Information Retrieval, 4, 231–255, 2001

ExpansionTool: Concept-Based Query Expansion and Construction

KALERVO JÄRVELIN likaja@uta.fi
JAANA KEKÄLÄINEN lijakr@uta.fi
Department of Information Studies, University of Tampere, Finland

TIMO NIEMI tn@uta.fi
Department of Computer and Information Sciences, University of Tampere, Finland

Received December 26, 2000; Revised April 30, 2001; Accepted May 24, 2001

Abstract. We develop a deductive data model for concept-based query expansion. It is based on three abstraction levels: the conceptual, linguistic and string levels. Concepts and relationships among them are represented at the conceptual level. The linguistic level gives natural language expressions for concepts. Each expression has one or more matching patterns at the string level. The models specify the matching of the expression in database indices built in varying ways. The data model supports a declarative concept-based query expansion and formulation tool, the ExpansionTool, for heterogeneous IR system environments. Conceptual expansion is implemented by a novel intelligent operator for traversing transitive relationships among cyclic concept networks. The number of expansion links followed, their types, and weights can be used to control expansion. A sample empirical experiment illustrating the use of the ExpansionTool in IR experiments is presented.

Keywords: query formulation, conceptual models, ontology, deductive data model, concept-based information retrieval

1. Introduction

Retrieval of digital text documents is based on character string matching rather than retrieving meanings. The searcher is encumbered with the selection of strings that accurately represent the needed information and match documents carrying that information. Solutions for this problem, sometimes referred to as the vocabulary problem, have been sought either at the storage or at the retrieval phase. At the storage phase, documentation languages, like thesauri, may be used to control vocabulary. Because intellectual document description is too expensive in most cases, more attention has been devoted to the retrieval phase. Relevance feedback and different query expansion (QE for short) methods are typical solutions. Relevance feedback has often proved to be beneficial, but its effectiveness depends on search string selection of the initial queries, ranking function and the number of relevant items known, i.e., on the quality of the search results (Beaulieu et al. 1997, Buckley et al. 1995, Harman 1992, Xu and Croft 1996). QE based on knowledge structures (e.g., thesauri) does not depend on search output, but it has not been found unambiguously useful (e.g. Voorhees 1994, Jones et al. 1995, Crestani et al. 1997).

Our starting point for QE is different because we aim, instead of search strings, to start query formulation from concepts. We believe that information needs may be represented as sets of concepts, which in turn have several different search string representations depending on the search environment. Our aim is to equip the searcher with a conceptual model representing semantic relationships among concepts and giving for each concept a set of search strings that may represent concepts in different search environments. The thesaural structure controlling hierarchies, associative relations and synonymy suits well for this kind of conceptual model. The model is managed by a tool that supports (1) searchers to automatically construct and expand effective queries without prior understanding about query structures and their interaction with expansion in various retrieval environments, and (2) QE experimentation with query structures, expansion and other query construction parameters.

Thesaurus modeling and software have received notable attention in IR literature. Jones and others (1993, 1995) introduce a thesaurus data model, based on the relational data model (RDM) and investigate the feasibility of incorporating intelligent algorithms into software for thesaurus navigation. Paice (1991) proposed a spreading activation method for thesaurus based QE. Term nodes, which are sufficiently loaded by spreading activation, are used to expand queries. Järvelin and others (1996) proposed a deductive data model (see, e.g., Ullman 1988) for thesaurus representation, query construction and expansion. Their deductive query language allowed navigation of transitive relationships in thesauri, which were represented as acyclic graphs. In it hierarchical relationships were processed by deductive operations, e.g., by expanding an abstract concept step by step to all of its descendants. It was not possible to limit expansion by the number of expansion steps. Associative relationships, due to their cyclic (symmetric) nature, could not be processed transitively. They were exploited by a single step only through traditional relational processing. Although unrestricted expansion over associative relationships is bound to impair performance, the single step limitation is often too strict in practice. Several thesauri, e.g., statistical thesauri, may only have "associative" relationships. Also spreading activation methods require uniform processing of all terminological relationships. Therefore it is desirable to have a uniform representation for all conceptual (or terminological) relationships and an expansion operator, which supports expansion from selected concepts toward selected (semantic) directions, to an adjustable distance, and/or until (as long as) an adjustable weighting criterion is fulfilled. In this paper we propose such a representation and describe such an expansion operator.

Our data model contains three levels of abstraction (Järvelin et al. 1996). The *conceptual level* represents concepts and conceptual relationships (e.g., hierarchical relationships). The *linguistic level* represents natural language expressions for concepts and their relationships (synonymy). Typically there are many expressions—including single words, compounds and phrases—for each concept. Each expression may have one or more matching patterns at the *string level.* Each matching pattern represents, in a query-language independent way, how the expression may be matched in texts or database indices built in varying ways, e.g., with or without stemming, morphological normalization, and compound word splitting into their component words. Query expansion is performed at all levels of abstraction.

Many languages are rich in compound words and have more complex inflectional properties than English (Alkula 2000 and Pirkola 2001). These properties may be handled in

several ways in database indexing. Thus a desirable feature of a query construction tool is to take automatically into account target database indexing (stemming, normalization, compound splitting) in the formulation of individual search keys. Our query construction and expansion tool is capable of this.

In modern IR environments both ordinary users and researchers often need to utilize or test several different IR systems. Their query language paradigms, operators and expressive power may vary strongly. There are, e.g., the probabilistic and Boolean paradigms. The sets of operators may vary in operator names (e.g., "and", "#and", and "*"), syntax (e.g., prefix form as in InQuery, or infix) and property details (e.g., "phrase", "Wn", and "ADJ"). One language may allow disjunctive Boolean clauses within a proximity operator whereas another does not. Therefore it is desirable that the tool for query construction and expansion automatically converts the query into the required target query language. If precise conversion is not possible, the nearest equivalent should be used. Our query construction and expansion tool supports such conversions.

We introduce the query construction and expansion tool, called the ExpansionTool, and demonstrate its use in the evaluation of query structuring and expansion in text retrieval. Section 2 presents the basic data model and the new QE operator. A sample knowledge base is also given. Section 3 gives a formal account of QE using the ExpansionTool and presents its interface. Section 4 demonstrates the use of the ExpansionTool for query construction and expansion in a test environment. Sections 5 and 6 contain discussion and conclusions.

2. The data model

2.1. Three abstraction levels

The three abstraction levels: conceptual, linguistic and string level are well founded in the IR literature (Croft 1986, Paice 1991, UMLS 1994). Thus we can differentiate concepts and relationships (e.g., the generic, partitive and associative relations) among them at the conceptual level, concept expressions and their relationships (the equivalence relation) at the linguistic level, and matching patterns (e.g., full-word strings, stems, string patterns involving wildcards) indicative of linguistic expressions at the string level. Expressions represent concepts and each concept may have several expressions in several natural and artificial languages. The expressions may be basic words, compound words, phrases or larger linguistic constructs, or common codes and abbreviations (e.g., USD49.90).

Figure 1 illustrates the roles of the three levels in query formulation. Search concepts are first translated into search keys, which are (thesaurus) terms, common codes and/or natural language expressions. Thereafter the search keys are translated into matching patterns. Language-dependent aspects are represented at the linguistic and string levels.[1] At the string level all retrieval system dependent aspects are embedded in translators specific to query languages, not in the matching patterns.

In the ExpansionTool, we use a relational database (e.g., Ullman 1988) to represent concepts, their expressions (and relationships among expressions) and matching patterns. For transitive processing, a collection of ternary (three-column) relations is used to represent concept relationships. In many applications it is sufficient to use binary relations to represent

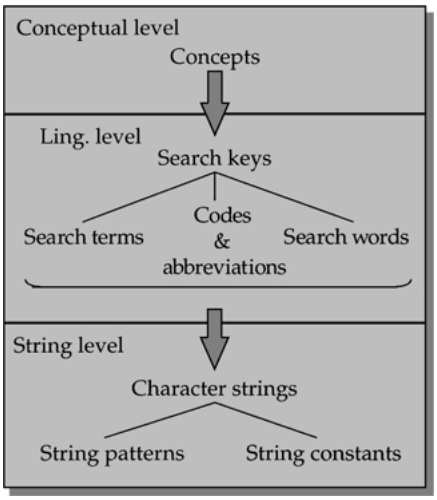

Figure 1. The abstraction levels of query formulation.

data for transitive processing. However, for QE we need to attach a strength score to each immediate connection between concepts. We therefore use ternary relations for modeling concept relationships. They are chosen according to the application area and are either generally hierarchic or associative. Different relations may represent different subtypes of these relationships (e.g., generalization and partitive relationships).

The matching pattern language has, among others, the following features (Järvelin et al. 1996):

- Representing atomic basic words by their morphological basic forms, e.g., bw(accident).
- Representing compound words by their morphological basic forms, e.g., cw(<bw(jet), bw(lag)>). The basic form matching patterns take into account that the database index may or may not recognize the compound word components. Thus the matching patterns are able to generate both whole compound word in the basic form and each of its components.
- Representing phrases with a specified word order through morphological basic forms: for example, 'information retrieval' is modeled by phra(2, < bw(information), bw(retrieval)>) indicating two components and listing them.
- Representing word proximity in a specified order, with intervening words allowed, through morphological basic forms or stems. For example, 'information retrieval' would be modeled by prox(2, <bw(information), bw(retrieval)>, 3) indicating two components, listing them, and allowing for distance of 0–3 words.

We will not discuss the relational database in this paper but, instead, its formal representation for QE. The formalization of the conceptual model is based on the set-theoretic description of a thesaurus database by Sintichakis and Constantopoulos (1997), and notations and formal representation conventions by Järvelin and Niemi (1993). The present formalization is based on Kekäläinen's (1999) definitions but modified for the new cyclic

network based expansion. This formalization is a compact and exact way to define the query expansion process in the ExpansionTool. However, the formalization is simplified: all details of the application (e.g., the reliability figures for expressions and matching patterns) are not fully covered.

We discuss conceptual models based on nuclear waste management as a sample domain. The following table (Table 1) describes the domain. It gives for each concept its identifier (cno) and name (cname), its related concepts in specialization, generalization and association relationships with their relationship strengths. The latter are real numbers in the range (0.0, 1.0] and used for controlling expansion. Moreover, Table 1 gives for each concept its expression identifier (eno) and expression with an indicator whether it is a term or not. The final column abstracts the matching pattern for each expression with the number of dots indicating the allowed distance between phrase components and an indicator (yes/no) whether the model is precise or not. For example, concept c6 has c5 as a generalization and c8 and c9 as associations. The term for c6 is t60, low active waste, which has two matching

Table 1. The sample domain of nuclear waste management.

Cno, Cname	Specializations	Generalizations	Associations	Eno, expression, term	Pattern, precise
c4, radioactive waste	c5, 1.0	–	c8, 0.7 c9, 0.6	t40, radioactive waste, yes;	"radioactive waste," yes "radioactive...waste," no
c5, nuclear waste	c6, 1.0 c7, 1.0	c4, 0.5	c8, 0.8 c9, 0.8	t50, nuclear waste, yes	"nuclear waste," yes; "nuclear...waste," no
c6, low active waste		c5, 0.5	c8, 0.8 c9, 0.8	t60, low active waste, yes	"low active waste," yes; "low ... active ... waste," no
c7, high active waste	–	c5, 0.5	c8, 0.8 c9, 0.8	t70, high active waste, yes	"high active waste," yes; "high ... active ... waste," no
c8, fission product	–	–	c4, 0.7 c5, 0.8 c6, 0.8 c7, 0.8	t80, fission product, yes	"fission product," yes "fission...product," no
c9, spent fuel	–	–	c4, 0.6 c5, 0.8 c6, 0.8 c7, 0.8	t90, spent fuel, yes	"spent fuel," yes; "spent...fuel," no
c10, storage	c11, 1.0		–	t100, storage, yes; nt101, store, no; nt102, stock, no	"storage," yes; "store," no "stock," no
c11, repository		c10, 0.5	–	t110, repository, yes	"repository," yes
c12, process	–	–	c13, 0.5 c14, 0.6	t120, process, yes	"process," yes
c13, refine	–	–	c12, 0.5	t130, refine, yes	"refine," yes
c14, treat	–	–	c12, 0.6	t140, treat, yes	"treat," yes

patterns, the precise one being "low active waste", and the other "low ... active ... waste" which allows three intervening words.

Tuples and trees are commonly used data structures in information systems specification. We need them in the formal representation of the conceptual model. We apply the following notational conventions for their representation.

Notational convention 1: Sequences consisting of structurally homogeneous objects are represented as n-tuples. Finite n-tuples are denoted between angle brackets, e.g., t1 = <a,b,c>. The *i*th component of a tuple t is denoted by t[i]. For example, t1[3] = c.

Notational convention 2: Sequences consisting of structurally heterogeneous objects are represented as trees. Trees are denoted between parenthesis. For example $s1 = (a,\ b,\ \{3, 7\})$ is a tree consisting of two atomic components and one set-valued component. Let s be any finite *tree* with n components. The selector function σ_i $(i = 1, \ldots, n)$, selects the ith component of s. For example, if $s1 = (a, b, \{3, 7\})$, then $\sigma_3(s1) = \{3,\ 7\}$.

Our conceptual models consist of seven main components. Therefore the conceptual model CM has the components CM = (*c-term, e-strict, e-all*, SYN, SPEC, GEN, ASS) where

- *c-term* is a function which maps concepts at the conceptual level to their terms at the linguistic level, i.e., concepts from the concept set C to the terms of the term set T. The function *c-term1* = {<c4, t40> <c5, t50>, <c6, t60>, <c7, t70>, <c8, t80>, <c9, t90>, <c10, t100>, <c11, t110>, <c12, t120>, <c13, t130>, <c14, t140>} is a sample function and gives t70 as the term for concept c7.
- *e-strict* is a function which maps each term expression to its precise (most reliable) matching patterns. The sample function *e-strict1* = {(t40, {phra(2,<bw(radioactive), bw (waste)>)}), (t50, {phra(2, <bw(nuclear), bw(waste)>)}), (t60, {phra(2, <cw(<bw(low), bw(active)>), bw(waste)>)}), (t70, {phra(2, <cw(<bw(high), bw(active)>), bw(waste) >)})} gives for the term t70 a single element set which models the phrase *high-active waste* as the compound *high-active* and the basic word *waste*.
- *e-all* is a similar function but maps each (term and non-term) expression to all its matching patterns. The sample function *e-all1* = {(t40,{phra(2, <bw(radioactive), bw(waste)>), prox(2, <bw(radioactive), bw(waste)>, 3)}), (t70, {phra(2, <cw(<bw(high), bw (active) >), bw(waste)>), prox(2, <cw(<bw(high), bw(active)>), bw(waste)>, 3)}), (nt101, {bw(store)}> , <nt102, {bw(stock)})} gives, in addition to the above phrase model *high-active waste* for the term t70, also the less reliable proximity model allowing three word distance between the compound *high-active* and the basic word *waste*. The non-terms nt101 and nt102 are given basic word models for *store* and *stock*, respectively.
- SYN is a binary equivalence relation between the set of term expressions T and the set of non-term expressions NT. It gives for each term identifier the set of identifiers of synonymous expressions. For example, the relation SYN1 = {(t100, {nt101, nt102})} gives nt101 and nt102 as synonymous expressions for t100.
- SPEC is a collection of specialization relations SPEC = $\{SR_1, SR_2, \ldots, SR_n\}$. Each specialization relation SR_i consists of tuples <x, y, s> where y is the hierarchically narrower

concept of x by the strength s. SPEC may contain more than one specialization relations based on different specialization principles, e.g., genus–species, part–whole. These are defined according to the application area. For example, SPEC1 = {{<c4, c5, 1.0>, <c5, c6, 1.0>, <c5, c7, 1.0>, <c10, c11, 1.0> }} represents a genus-species relationship given in Table 1 specifying, among others, c6 (low active waste) and c7 (high active waste) as narrower concepts of c5 (nuclear waste), all with strength 1.0.

- GEN is a collection of generalization relations GEN = {GR_1, GR_2, . . . ,GR_n}. Each generalization relation GR_i consists of tuples < x, y, s > where y is the hierarchically broader concept of x by the strength s. GEN may contain more than one generalization relations based on different generalization principles, e.g., genus–species, part–whole. These are defined according to the application area. For example, GEN1 = {{<c5, c4, 0.5>, <c6, c5, 0.5>, <c7, c5, 0.5>, <c11, c10, 0.5>}} represents a genus–species relationship specifying, among others, c5 (nuclear waste) as a broader concept of c6 (low active waste) and c7 (high active waste) with strength 0.5. Obviously, GEN represents the inverted relationships of SPEC. Representing them separately has two advantages: firstly, the strengths may be unsymmetrical, and secondly, the direction of conceptual expansion may be controlled.
- ASS is a collection of association relations ASS = {AR_1, AR_2, . . . ,AR_n}. Each association relation AR_i consists of tuples <x, y, s> where y is the semantically but non-hierarchically associated with concept of x by the strength s. ASS may contain more than one association relations based on different association principles, e.g., process–outcome, object–instrument. These are defined according to the application area. For example, ASS1 = {{<c4, c8, 0.7>, <c8, c4, 0.7>, <c4, c9, 0.6>, <c9, c4, 0.6>, <c5, c8, 0.8>, <c8, c5, 0.8>, <c5, c9, 0.8>, <c9, c5, 0.8>, . . .}} represents, among others, c5 (nuclear waste) as associated with c8 (fission product) and c9 (spent fuel) with strength 0.8. Each direction of the association is represented separately allowing for unsymmetrical strenghts.

Now we may refer to the components of the conceptual model as follows: σ_1(CM) = *c-term,* . . . , σ_7(CM) = ASS. For simplicity, *c-term, e-strict, e-all*, SYN, SPEC, GEN, ASS, C, T and EXP will be used below to denote the conceptual model components. Our sample conceptual model is the model CM1 = (*c-term1, e-strict1, e-all1*, SYN1, SPEC1, GEN1, ASS1) which is given in the appendix.

We use several functions to process the conceptual model. These are introduced below briefly and informally. Full formal definitions are given in Järvelin et al. (2001). The functions *c-term, e-strict* and *e-all* were already defined as components of the conceptual models. They may applied in the normal functional way, e.g., *c-term1*(c4) = t40, and *e-all1*(t40) = {phra(2, <bw(radioactive), bw(waste)>), prox(2, <bw(radioactive), bw(waste) >3)}.

The function *t-syns* is based on the equivalence relation SYN and gives for an argument term t its synonymous expressions. For example, *t-syns*(t100, SYN1) = {nt101, nt102}.

The function *c-expr* gives the set of all expressions related to an argument concept c. It is based on the functions *c-term* and *t-syns*, and the equivalence relation SYN. The function *c-expr* finds for an argument concept c its term and non-term expressions. For example, *c-expr*(c10, *c-term1*, SYN1) = {t100, nt101, nt102}.

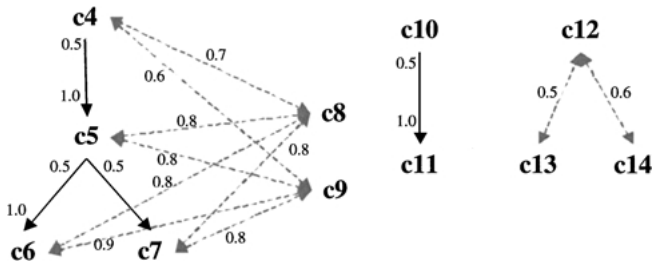

Figure 2. A sample concept graph.

2.2. Network based concept expansion

The main operator of our tool is a conceptual QE operator intended for manipulation transitive relationships. Basically, the QE operator is a novel generalized operator for traversing collections of cyclic graphs, consisting of concept nodes and of links between them.[2] Figure 2 illustrates the concept graph of our conceptual model. The conceptual QE operator is used for computing paths of nodes that satisfy the criteria given by the user. These may concern:

- the starting nodes from which the paths start
- the target nodes at which the paths end
- the intermediate nodes, which must belong to the paths
- the maximum number of nodes that is allowed to belong to a path
- the minimum (maximum) weight a path is required to have (by link weight multiplication).

This operator is a generalized operator for traversing cyclic graphs and suitable for many application areas. In the present paper, we shall consider its application in QE and call it the CQE operator (for conceptual query expansion operator). Therefore, for our CQE operator, the nodes are *concept nodes* and the *links* immediate concept relationships, with *strengths* in the range (0, 1]. Different types of relationships are represented by different graphs.

Consider the sample concept graph of figure 2. They represent two subgraphs containing hierarchies starting at concept nodes c4 and c10, and one graph based on associations only. Hierarchical relationships (in black arrows), their inverted relationships (black arrows inverted) and associations (gray dashed arrows in both directions) are represented with their *strengths*. The graph is cyclic, because the links can be traversed to both directions.

Graphs can formally be represented by ternary relations $R \subseteq \mathbf{C} \times \mathbf{C} \times \mathbf{R}$, where $\mathbf{C}$ denotes the set of concept nodes and $\mathbf{R}$ real numbers. Each element, say $< c4, c5, 1.0 >$, is a tuple representing that two concept nodes (c4, c5) are related by the strength 1.0. Different relationship types (or directions) may be represented in different relations. The specialization, generalization and association relations SPEC, GEN and ASS of the preceding section are such collections of graphs. The sample relations SPEC1, GEN and ASS1 correspond to figure 2. The tuple <c4, c5, 1.0> belongs to SPEC1 and the tuple <c5, c4, 0.5> to GEN1. Both <c12, c13, 0.5> and <c12, c13, 0.5> belong to ASS1.

Using the CQE operator, any concept node may be expanded by other nodes that are within a required distance, lead toward a required node, can be reached via some specified nodes, or can be reached while keeping the path weight above a required minimum. The following are some paths the CQE operator may compute. The tuple <c4, c5, c6> represents a path in SPEC1 and has *length* 3. Its *weight* is 1, calculated by *multiplying* the strength figures (at the arrow heads in figure 2). The tuple <c7, c5, c4> represents a path in GEN1 and has weight 0.25, again calculated by multiplying the strength figures (at the arrow ends in figure 2). Moreover, the tuple <c4, c5, c7, c9> represents a path in the *union* of SPEC1 and ASS1 with length 4 and weight 0.8. For query expansion, it is useful to constrain concept paths by their length and weight, the latter computed by multiplication of strength values, given in the range (0, 1), of the links and constrained by a minimum value.

Below we shall consider the CQE operator formally by defining the function *expand* which expands a concept node to a set of all concept paths that are constructable from this given start node without specifying any target or intermediate nodes, under length and/or weight constraints. The function *expand* finally disassembles the paths and produces the union of their constituent concept nodes, all passing the path length and/or weight constraints. This is the basic way of using CQE operator. The other ways of using the CQE operator lead to analogous definitions but fall beyond the scope of this paper. Some further notational conventions are introduced before the definitions of the functions.

Notational convention 3: Finite n-tuples are assembled by the *catenation* operator $\leftrightarrow$. If t1 = <a, b> and t2 = <c, d, e, f> are tuples, then t1 $\leftrightarrow$ t2 = t = <a, b, c, d, e, f>. The *length* of a tuple t is given by *len*(t). For example, *len*(t2) = 4. Analogously to set membership, we use the notations c $\underline{\in}$ t, and c $\underline{\notin}$ t, to test whether a given component c belongs or does not belong to a given tuple t. For example, the expressions e $\underline{\in}$ t2, and a $\underline{\notin}$ t2 are true.

Notational convention 4: The *power set* of a set S is denoted by P(S). For example, if S = {a,b,c} the P(S) = {{},{a}, {b}, {c}, {a,b},{a,c},{b,c},{a,b,c}}. If S is any subset of a set **D**, it belongs to the set P(**D**), because S $\subseteq$ **D**. For example, {1, 2, 3, 4} $\in$ P(**I**) where **I** denotes the set of integers. The set of tuples consisting of elements, which belong to the same set **D**, is denoted as T(**D**). Thus T(**D**) is the set of tuples constructable from the elements of **D**. For example, <1, 2, 3, 4> $\in$ T(**I**).

Notational convention 5: Let f : D $\rightarrow$ R be any function. In its *signature* f is a *function symbol*, D is a *domain*, i.e., it defines a set of values to which the function can be applied, and R is a *range*, i.e., it defines a set of values, to which the results of function applications belong. In complex cases a domain set may be a Cartesian product or its subset (mathematically a relation). If the function f has the signature f : D $\rightarrow$ R then *dom*(f) = D denotes the domain of f and *rng*(f) $\subseteq$ R the range of f.

In the definition of the function *expand* we need two auxiliary functions, *weight* and *path-expansion*. The former gives, for an immediate pair of concept nodes, the weight related to their relationship in the underlying ternary relation. The latter gives, for a given initial path Path, all concept paths ExpPath that extend the initial path and are (1) connected to the initial path in the concept network, (2) do not repeat the nodes of the path (thus avoiding

unterminating computation), and satisfy the given (3) length and (4) weight constraints LC and WC. The definition is declarative, we bypass all implementation-related issues for simplicity.

Definition 1. Let Scope be a ternary relation containing immediate concept relationships, LC a given length constraint ($LC \in \mathbf{I}^+$), WC a given weight constraint ($WC \in \mathbf{R}$, $0 < LC \leq 1$), and Path be an original path to be expanded. The set of paths under the length and weight constraints LC and WC extendable from Path are given by the function *path-expansion*:

$$
\begin{aligned}
&\textit{path-expansion}\colon \mathrm{P}(\mathbf{C} \times \mathbf{C} \times \mathbf{R}) \times \mathbf{I} \times \mathbf{R} \times \mathrm{T}(\mathbf{C}) \rightarrow \mathrm{P}(\mathrm{T}(\mathbf{C}))\\
&\textit{path-expansion}(\mathrm{Scope}, \mathrm{LC}, \mathrm{WC}, \mathrm{Path}) =\\
&\quad \{\mathrm{ExpPath} \mid \mathrm{Path1} \in \mathrm{T(C)}\colon \mathrm{ExpPath} = \mathrm{Path} \leftrightarrow \mathrm{Path1}\\
&\qquad \wedge\, \forall \mathrm{i} \in \{\textit{len}(\textit{Path}), \ldots, \textit{len}(\mathrm{ExpPath}) - 1\}\colon\ <\mathrm{ExpPath[i]}, \mathrm{ExpPath[i+1]},\\
&\qquad\quad \mathrm{w}> \in \mathrm{Scope}\\
&\qquad \wedge\, \forall \mathrm{i}, \mathrm{j} \in \{1, \ldots, \textit{len}(\mathrm{ExpPath})\}, \mathrm{i} \neq \mathrm{j} \Rightarrow \mathrm{ExpPath[i]} \neq \mathrm{ExpPath[j]}\\
&\qquad \wedge\, \textit{len}(\mathrm{ExpPath}) \leq \mathrm{LC}\\
&\qquad \wedge \prod_{\mathrm{i}=1,\ldots,\textit{len}_{(\mathrm{ExpPath})}} \textit{weight}(\mathrm{ExpPath[i]},\ \mathrm{ExpPath[i+1]}, \mathrm{Scope}) \geq \mathrm{WC}\}
\end{aligned}
$$

where $\textit{weight}(\mathrm{c1}, \mathrm{c2}) = \mathrm{w}$, when $<\mathrm{c1}, \mathrm{c2}, \mathrm{w}> \in \mathrm{Scope}$.

Consider the sample concept network of figure 2. Let Scope1 be the union of the ternary relations, Scope1 = SPEC1 ∪ ASS1. If we construct expansion paths for c4 without length constraints by weight constraint 0.7 we use the expression path-expansion(Scope1, ∞, 0.7, <c4>) which yields the path-set PS1 =

```
{<c4, c5>,          /*with weight 1.0 and length 2*/
 <c4, c8>,          /*with weight 0.7 and length 2*/
 <c4, c5, c6>,      /*with weight 1.0 and length 3*/
 <c4, c5, c7>,      /*with weight 1.0 and length 3*/
 <c4, c5, c8>,      /*with weight 0.8 and length 3*/
 <c4, c5, c9>,      /*with weight 0.8 and length 3*/
 <c4, c5, c6, c8>,  /*with weight 0.8 and length 4*/
 <c4, c5, c6, c9>,  /*with weight 0.8 and length 4*/
 <c4, c5, c7, c8>,  /*with weight 0.8 and length 4*/
 <c4, c5, c7, c9>}  /*with weight 0.8 and length 4*/
```

The function *expand* constructs all possible paths from a single starting concept node sn within a scope set SS consisting of a collection of concept relationships {R1, R2, ..., Rn}, under given weight and path length constraints. The target and intermediate nodes are not specified or limited in any way, and weight computation is based on multiplication and constraining the score by minimum. The function *expand* unites the concept relationships into a single scope set and constructs an elementary path <sn> consisting of the starting concept-node. It then applies the function *path-expansion* in constructing the required paths.

Definition 2. Let sn be the start node, SS the scope set or SS = {R1, R2, . . . , Rn}, LC the length constraint, and WC the weight constraint for concept expansion. The expansion result is defined by the function *expand* as follows:

$$expand: \mathbf{C} \times P(P(\mathbf{C} \times \mathbf{C} \times \mathbf{R})) \times \mathbf{I} \times \mathbf{R} \rightarrow P(T(\mathbf{C}))$$
$$expand(sn, SS, LC, WC) = path\text{-}expansion(\cup_{R \in SS} R, LC, WC, <sn>)$$

For example, the expression *expand*(c4, {SPEC1, ASS1}, ∞, 0.8) yields the path set PS2 = {<c4, c5>, <c4, c5, c6>, <c4, c5, c7>, <c4, c5, c8>, <c4, c5, c9>, <c4, c5, c6, c8>, <c4, c5, c6, c9>, <c4, c5, c7, c8>, <c4, c5, c7, c9>}. To obtain the set of concept nodes that expand the start node sn from the result of the function *expand*, one takes the union of the path components by the function *nodes*.

Definition 3. Let PS be a set of paths. The set of nodes forming the paths in PS is given by the function *nodes*:

$$nodes: P(T(\mathbf{C})) \rightarrow P(\mathbf{C})$$
$$nodes(PS) = \cup_{path \in PS} \{c \mid c \subseteq path\}$$

For example, the expression *nodes*(PS2) = CE1 = {c4, c5, c6, c7, c8, c9}. Any conceptual expansion from a given node sn in the scope set SS with the length and weight constraints LC and WC is expressed by the function:

$$nodes(PathSet), \quad \text{where} \quad PathSet = expand(sn, SS, LC, WC),$$

by adjusting the expansion scope SS, length constraint LC and weight constraint WC suitably. Note that by a slightly modified definition of *path-expansion* it would be possible to get the weight for each path node individually. Such weight could be used for weighting the concepts individually. Now there only is the guarantee that all nodes found have a path exceeding the required minimum weight, although some do this by a much greater marginal than others.

The following four conceptual expansion functions perform conceptual expansion specifically within the specialization, generalization, and association relationships, and generally within all conceptual relationships.

Definition 4. Let SPEC, GEN and ASS be any collections of specialization, generalization and association relationships (respectively), RELS = SPEC ∪ GEN ∪ ASS, c a concept ($c \in \mathbf{C}$) and s a real number indicating minimum concept weight constraint. All narrower, broader and associated concepts of c, with the minimum weight s, without path length constraints, are obtained by the functions c-spec, c-gen, c-asso, and c all, respectively. These functions have the same signature or $\mathbf{C} \times P(P(\mathbf{C} \times \mathbf{C} \times \mathbf{R})) \times \mathbf{R} \rightarrow P(\mathbf{C})$ and are defined analogously as follows:

$$c\text{-}spec(c, SPEC, s) = nodes(expand(c, SPEC, \infty, s))$$
$$c\text{-}gen(c, GEN, s) = nodes(expand(c, GEN, \infty, s))$$
$$c\text{-}asso(c, ASS, s) = nodes(expand(c, ASS, \infty, s))$$
$$c\text{-}all\ (c, RELS, s) = nodes(expand(c, RELS, \infty, s)).$$

3. Query expansion in the ExpansionTool

3.1. Conceptual expansion

Unexpanded—or original—queries are formulated from concepts selected from the conceptual model. Further, these concepts are interpreted as belonging to conjunctive facets representing aspects of the information need. The concepts of each facet are alternative (or disjunctive) interpretations of the facet.

Notational convention 6: Let $c_{11}, \ldots, c_{1n1}, c_{21}, \ldots, c_{2n2}, \ldots, c_{k1}, \ldots, c_{knk}$ be concept identifiers which belong to facets $F_1 = \{c_{11}, \ldots, c_{1n1}\}$, $F_2 = \{c_{21}, \ldots, c_{2n2}\}$ and $F_k = \{c_{k1}, \ldots, c_{knk}\}$. A conceptual query Q is represented as a set of facets $Q = \{F_1, F_2, \ldots, F_k\} = \{\{c_{11}, \ldots, c_{1n1}\}, \{c_{21}, \ldots, c_{2n2}\}, \ldots, \{c_{k1}, \ldots, c_{knk}\}\}$.

In principle, there is an 'AND' between the facets $F_1, F_2, \ldots, F_k$ and an 'OR' between the concepts within each facet, e.g., between $c_{11}, \ldots, c_{1n1}$. This high-level structure is maintained throughout query construction and rejected only in the matching pattern translation phase if the query structure requires this (e.g., through the use of a single probabilistic operator instead of Boolean operators).

Query formulation from concepts to a query is illustrated by the following example. Assume that the test request is about the processing and storage of radioactive waste. In the sample conceptual model CM1, the concepts c4, c10, and c12 representing the terms *c-term1*(c4) = t40 (for 'radioactive waste'), *c-term1*(c10) = t100 (for 'storage') and *c-term1*(c12) = t120 (for 'process') represent this information need. Two facets are identified: $F_1 = \{c_4\}$, $F_2 = \{c10, c12\}$. They form the concept query Q1 = {{c4}, {c10, c12}}.

In conceptual query expansion, each concept is expanded to a disjunctive set of concepts on the basis of conceptual relationships selected. For an original query $Q = \{F_1, \ldots, F_k\}$, the expansion result is in each case an expanded concept query $Q' = \{F'_1, \ldots, F'_k\}$ where each facet $F'_i = \{c_{i1}, c_{i11}, c_{i12}, \ldots, c_{i1m1}, \ldots, c_{in}, c_{in1}, c_{in2}, \ldots, c_{inmn}\}$ contains the *original concept identifiers* $\{c_{i1}, \ldots, c_{in}\}$, and the *expansion concept identifiers*, $\{c_{i11}, c_{i12}, \ldots, c_{i1m1}, \ldots, c_{in1}, c_{in2}, \ldots, c_{inmn}\}$.

Conceptual query expansion is performed within a selected collection of concept relationships RELS. Within this collection, all derivable conceptual expansions are performed and combined, i.e., all original concepts, their narrower, broader and associative concepts are collected if these concept relationships are included in RELS. The function for the full expansion of a concept is defined as follows:

Definition 5. Let CM be a conceptual model CM = (*c-term*, *e-strict*, *e-all*, SYN, SPEC, GEN, ASS), RELS be any collection concept relationships (RELS $\subseteq$ SPEC $\cup$ GEN $\cup$ ASS), $Q = \{F_1, \ldots, F_k\}$ be any query with $F_1, \ldots, F_k$ facets and s the weight constraint. The conceptually expanded query for the original query Q is obtained by the function:

$$\textit{cons-q-expand}: \mathrm{P}(\mathrm{P}(\mathbf{C})) \times \mathrm{P}(\mathrm{P}(\mathbf{C} \times \mathbf{C} \times \mathbf{R})) \times \mathbf{R} \rightarrow \mathrm{P}(\mathrm{P}(\mathbf{C}))$$

$$\textit{cons-q-expand}(Q, \mathrm{RELS}, s) = \{\textit{cons-f-expand}(F, \mathrm{RELS}, s) \mid F \in Q\}$$

when *cons-f-expand*: $P(\mathbf{C}) \times P(P(\mathbf{C} \times \mathbf{C} \times \mathbf{R})) \times \mathbf{R} \rightarrow P(\mathbf{C})$

cons-f-expand(F, RELS, S) $= \cup_{c \in F}$ *c-all*(c, RELS, S).

By selecting RELS suitably, specific types of conceptual expansions are obtained. For example, in the case of sample query Q1 the narrower concept expansion *cons-q-expand*(Q1, SPEC1, 0.8) yields the expanded query $Q1_n$ = {{c4, c5, c6, c7}, {c10, c12, c11}}. The new concepts are c5 ('nuclear waste'), c6 ('low-active waste'), c7 ('high-active waste'), and c11 ('repository').

The associative concept expansion for Q1 is *cons-q-expand*(Q1, ASS1, 0.5) gives the expanded query $Q1_a$ = {{c4, c8, c9}, {c10, c12, c13, c14}}. The new concepts are c8 ('fission product'), c9 ('spent fuel'), c13 ('refine') and c14 ('treat').

The combined narrower and associative concept expansion for Q1 is *cons-q-expand*(Q1, SPEC1 ∪ ASS1, 0.5) and gives the expanded query $Q1_{n\&a}$ = {{c4, c5, c6, c7, c8, c9}, {c10, c12, c11, c13, c14}}. The new concepts are as above.

3.2. *Query expansion to terms and synonyms*

After conceptual expansion, the next step in query construction is finding the terms and their equivalent expressions (synonyms) for the (expanded) conceptual query. The following definition gives two functions *q-terms* and *q-syns*, which perform these expansions. The former gives only the terms for the concepts, the latter both the terms and their synonyms. Below, we denote the set of terms by **T**, the set of non-term expressions by **NT** and all expressions by **EXP** = **T** ∪ **NT**.

When terms only are used, the original concept facets are represented by expression facets $\{E_1, \ldots, E_k\}$, where each facet E_i is derived from the corresponding concept facet F_i by replacing each concept identifier in F_i by the identifier of the corresponding term. In the synonym expansion, the set of concept facets $\{F_1, \ldots, F_k\}$ is translated and expanded into synonyms by adding all equivalent expressions of the terms of the original concepts to the query. Again, the result is an expanded set of expression facets $\{E_1, \ldots, E_k\}$.

Definition 6. Let CM be a conceptual model CM = <*c-term*, *e-strict*, *e-all*, SYN, SPEC, GEN, ASS>, and Q = $\{F_1, \ldots, F_k\}$ be any conceptual query with $F_1, \ldots, F_k$ facets. The term and synonym expansions of the original query Q are obtained by the functions, respectively:

q-terms: $P(P(\mathbf{C})) \times \mathbf{CM} \rightarrow P(P(\mathbf{T}))$

q-terms(Q, CM) = {*f-terms*(F, CM) | F ∈ Q}

when

f-terms: $P(\mathbf{C}) \times \mathbf{CM} \rightarrow P(\mathbf{T})$

f-terms(F, CM) $= \cup_{c \in F}$*c-term*(c)

q-syns: $P(P(\mathbf{C})) \times \mathbf{CM} \rightarrow P(P(\mathbf{EXP}))$

q-syns(Q, CM) = {*f-syns*(F, CM) | F ∈ Q}

when
f-syns: $P(\mathbf{C}) \times \mathbf{CM} \rightarrow P(\mathbf{EXP})$
f-syns$(F, CM) = \cup_{c \in F}$ *c-expr*$(c,$ *c-term*, SYN$)$.

For example, *q-syns*(Q1, CM1) gives the synonymous expressions of the original unexpanded Q1 query as the identifier set {{t40}, {t100, nt101, nt102, t120}}. On the other hand, *q-terms*($Q1_n$, CM1) gives the terms of the narrower concept expanded query $Q1_n$ as the identifier set {{t40, t50, t60, t70}, {t100, t110, t120}}.

3.3. Query expansion to matching patterns

After expression expansion, the next step in query construction is finding the matching patterns for the (expanded) query. The following definition gives the functions *q-strict-patterns* and *q-patterns*, which perform these expansions for any query, represented as a faceted structure of expression identifiers. The former gives only the strict matching patterns for the expressions, while the latter all matching patterns. Below, we denote the set of matching patterns by **MM**.

In the matching pattern expansion all matching patterns of all expressions (exceeding the weight constraint) are added to the query. The set of expression identifier facets $\{E_1, \ldots, E_k\}$ is translated and expanded into matching patterns by adding all applicable patterns to the query. The result is an expanded set of matching pattern facets $\{P_1, \ldots, P_k\}$, where each facet P_i is derived from the corresponding expression facet E_i by representing each expression identifier in E_i by its matching patterns.

Definition 7. Let CM be a conceptual model CM = (*c-term*, *e-strict*, *e-all*, SYN, SPEC, GEN, ASS), and $Q = \{E_1, \ldots, E_k\}$, be any expression level query with $E_1, \ldots, E_k$ facets. The strict and all matching pattern expressions of the original query Q are obtained by the functions, respectively:

q-strict-patterns: $P(P(\mathbf{T})) \times \mathbf{CM} \rightarrow P(P(\mathbf{MM}))$
q-strict-patterns$(Q, CM) = \{$*f-strict-patterns*$(E, CM) \mid E \in Q\}$
when
f-strict-patterns: $P(\mathbf{T}) \times \mathbf{CM} \rightarrow P(\mathbf{MM})$
f-strict-patterns$(E, CM) = \cup_{e \in E}$ *e-strict*(e)

q-patterns: $P(P(\mathbf{EXP})) \times \mathbf{CM} \rightarrow P(P(\mathbf{MM}))$
q-patterns$(Q, CM) = \{$*f-patterns*$(E, CM) \mid E \in Q\}$
when
f-patterns: $P(\mathbf{EXP}) \times \mathbf{CM} \rightarrow P(\mathbf{MM})$
f-patterns$(E, CM) = \cup_{e \in E}$ *e-all*(e)

For example, *q-strict-patterns*(*q-syns*(Q1, CM1), CM1) gives the matching patterns for the synonymous expressions of the original unexpanded Q1 query as the set Q1P = {{phra(2, <bw(radioactive), bw(waste)>)}, {bw(storage), bw(store), bw(stock), bw(process)}}. On the other hand, *q-patterns*(*q-terms*($Q1_n$, CM1), CM1) gives the matching patterns for the

terms of the narrower concept expanded query $Q1_n$ as the set $Q1P_n =$

{{phra(2, <bw(radioactive), bw(waste)>),
prox(2, <bw(radioactive), bw(waste)>, 3),
phra(2, <bw(nuclear), bw(waste)>), prox(2, <bw(nuclear), bw(waste)>, 3),
phra(2, <cw(<bw(low), bw(active)>), bw(waste)>),
prox(2, <cw(<bw(low), bw(active)>), bw(waste)>, 3),
phra(2, <cw(<bw(high), bw(active)>), bw(waste)>),
prox(2, <cw(<bw(high), bw(active)>), bw(waste)>, 3)},
{bw(storage), bw(repository), bw(process)}}.

All conceptual, expression level and matching pattern expansions of an original conceptual query are obtained by applying and nesting the functions *cons-q-expand*, *q-strict-patterns*, *q-patterns*, *q-terms* and *q-syns* with suitable parameters. The queries represented as matching pattern facet sets are still query language independent, and need now be translated into query language specific expressions for execution.

3.4. Matching pattern translation

This step translates the query language independent expression into an expression of a given language. The starting point of matching pattern translation is the expansion result constructed above. Matching pattern translation is implemented on the basis of logic grammars (Abramson and Dahl 1989, Pereira and Warren 1980). Each grammar is a set of logical rules, which generate well-formed expressions of a specified query language. Each query language has its own logic grammar, which generates its specific expression types and structures.

The parameters of matching pattern translation are (1) a query structure indicator, (2) the database index type indicator, (3) the target query language indicator, and (4) the key reduction parameter. The first one is used to express how the facets are combined with each other, and how the keys are combined with each other. Kekäläinen and Järvelin (1998, 2000) and Kekäläinen (1999) have shown the effect of query structure for the effectiveness of expanded queries (see also Section 4.1). Therefore query structure is an important construction parameter.

A query structure is the syntactic structure of a query expression, as expressed by the query operators and parentheses. The structure of queries may be described as weak (queries with a single operator or no operator, no differentiated relations between search keys) or strong (queries with several operators, different relationships between search keys). More precisely, typical strong query structures are based on facets. Each facet indicates one aspect of the request, and is represented by one or more concepts, which, in turn, are expressed by a set of search keys (Kekäläinen 1999).

The ExpansionTool provides several alternatives for query structures, ranging from the conjunctive Boolean structure to probabilistic sum and weighted sum structures (all not available to all target query languages), including:

- and for facet conjunction,
- para for facet paragraph proximity,
- sum for probabilistic facet sums,

- syns for the probabilistic structure:[3]

  #sum(#syn($term_1$ syn_{11} syn_{12} ...,), #syn($term_2$ $syn_2$1...)...),

- wsyns(QW, FacetWList) for the probabilistic structure:

  #wsum(1 w_1 #syn($term_1$ syn_{11} syn_{12}...) w_2 #syn($term_2$ syn_{21} ...)...), where FacetWList = <w_1, w_2, ...> gives the weight of facets as a list of integers.

The *database index type* indicators bw, cw and iw indicate index types "basic words with compound words split," "basic words with compounds" and "inflected words," respectively. Allowed *query language identifiers* currently are *inquery* (for InQuery v3.1 by the Center for Intelligent Information Retrieval, University of Massachusetts), *iso* (for the ISO standard query language; ISO 1993), *topic* (for TOPIC by Verity Inc.), or *trip* (for TRIP by PSI Inc.).

The parameter *reduction* specifies whether keywords for matching are reduced or not. If reduced, duplicates or expressions logically covered by other expressions within each facet are reduced from the query. For example, the key 'industry' covers all proximity expressions containing the key 'industry' and other keys. Otherwise they are allowed. The options are 'reduce' for reduced keys and 'duplicates' for redundant keys. In the InQuery language, #sum(industry) logically covers #sum(industry #uw3(forest industry) #20(wood industry)) and #uw10(forest industry) covers #uw3(forest industry). Note however that the expressions may return different document sets. Reduction should always be used for Boolean queries.

If the target language of matching pattern translation does not support some specific feature of matching patterns or logical structure, then either the obvious closest or alternative construct of the target language is generated or query construction terminates with an error message. For example, the InQuery retrieval system (v3.1) does not have grammatical proximity operators (e.g., "sentence") but supports proximity conditions based on numeric word distance. Therefore the sentence proximity condition is translated an adjustable numeric proximity expression, e.g., #10 allowing 10 intervening words. InQuery neither supports disjunctions within proximity operations, i.e., the structure #10(#or(a b) #or(d e)). Therefore such structures are automatically converted into DNF, i.e., #or(#10(a d) #10(a e) #10(b d) #10(b e)) which is supported. The TOPIC query language provides the proximity operators "phrase," "sentence" and "paragraph." All such transformations are handled by the logic grammars.

Järvelin and others (1996) describe the matching pattern translation phase in more detail. The result of matching pattern translation is an expression of the target query language for an index of the specified type. The query may be very long, if it contains many broad concepts expanded by loose criteria, and if a proximity condition is applied between the facets.

Examples of query structures are given in Section 4.

4. Test on expansion effects

We demonstrate the properties of the ExpansionTool by constructing and testing queries with different structures and expansion types. The interaction and effects of the following variables are considered: the number of search concepts, the number of search strings representing concepts, query expansion (QE) with different semantic relationships, and query structures. The test demonstrates that by the use of the ExpansionTool one may

automatically construct queries, which differ in structural and expansion aspects and yield quite different effectiveness. Because it is fairly easy to define new query structures for ExpansionTool, it is a flexible tool for experimentation in varying retrieval environments. It has been used in IR experiments reported by Järvelin and others (1996), Kekäläinen (1999), and Kekäläinen and Järvelin (1998, 2000).

4.1. Test environment and query structures

The test environment is a text database containing Finnish newspaper articles operated under the InQuery retrieval system (see Turtle 1990, Turtle and Croft 1991). The database contains 54,000 articles published in three Finnish newspapers. The average article length is 233 words, and typical paragraphs are two or three sentences in length. The database index contains all keys in their morphological basic forms. In the basic word form analysis all compound words are split into their component words in their morphological basic forms. We use a collection of 30 requests, which are 1–2 sentences long, in the form of written information need statements. For these requests there is a recall base of 16,540 articles.

The total number of concepts in the test thesaurus was 832 and the number of expressions in the thesaurus is 1,345. The number of matching patterns in the thesaurus is 1,558. The values of concept association strength were based on the judgments of the researchers. Hierarchical relationships had strength 1.0 to the narrower concept direction and 0.5 to the broader concept direction. The justification is that narrower concepts represent a given concept more reliably than broader concepts.

InQuery was used in the demonstrative experiment. It supports both exact and best match retrieval, allows search key weighting, and has a wide range of operators, including Boolean operators and 'probabilistic' operators. The operators '#and,' '#or,' and '#not' and proximity searching by the operator '#*n*,' where *n* is an integer, allow probabilistic retrieval by Boolean operators, while '#band' allows ordinary Boolean retrieval (with the result set ranked). The proximity operator '#*n*' spans over sentence and paragraph boundaries. The probabilistic sum operators '#wsum' and '#sum' are also available.

In query formulation, search concepts are first selected from the ExpansionTool database on the basis of a request. Then the parameters structure, the databse index type indicator ('bw' in the experiment), the target query language identifier (here 'inquery'), and the key reduction parameter (here 'no') are given. In the present experiment, the path length is always '∞,' and the weight constraint for conceptually unexpanded queries is 1.0, and for expanded queries 0.3. We formulate queries for the test requests using four query structures: Boolean operators ('and'), paragraph proximity operators ('para', allowed word distance 40) and the 'probabilistic' structures 'ssyn' and 'sum.' Queries are either unexpanded, when each concept is represented by its term and the corresponding matching pattern, or expanded 1) by synonyms and narrower concepts (s + n expansion; 2) by synonyms, narrower and associated concepts (full expansion). Next, we shall give examples of the structures and expansions (unexpanded and synonym + narrower concept expanded queries).[4] Our sample request is "*processing and storage of radioactive waste*." The following concepts are selected from the thesaurus: *c4* (radioactive waste), *c10* (storage) and *c12* (process) and form the concept query Q1 = {{c4}, {c10, c12}} of Section 3. The expansions are also described in Section 3.

Proximity operators put more strict demands on the occurrence of search keys than the AND operator. However, when the proximity constraints are strict, the number of alternative search keys becomes decisive in order to keep even precision acceptable (Kristensen 1993). We reduced the number of search concepts per request in order not to set too strict conditions. Two operators are used to combine the search key alternatives for each concept: disjunction (*ProxOr*) and synonymity (*ProxSyn*). Because the queries are identical except for the initial operator, examples of the *ProxOr* queries only are given. In the examples, expressions like #0(low active) denote compound words (with hyphen).

- *ProxOr*, *no expansion:*

```
#or(#uw40(#1(radioactive waste) process) #uw40(#1(radioactive waste) storage))
```

- *ProxOr*, $s + n$ *expansion*:

```
#or(#uw40(#1(radioactive waste) process) #uw40(#1(nuclear waste) process)
    #uw40(#1(#0(high active) waste) process) #uw40(#1(#0(low active) waste)
          process)
    ...
    #uw40(#1(radioactive waste) storage) #uw40(#1(radioactive waste) stock)
    #uw40(#1(radioactive waste) store) #uw40(#1(radioactive waste) repository)
    ...
    #uw40(#0(low active) waste) storage) #uw40(#0(low active) waste) store)
    #uw40(#0(low active) waste) stock) #uw40(#0(low active) waste) repository))
```

A query with the Boolean operators (*BOOL*) is constructed using the block search strategy. In InQuery, a query with the 'true' Boolean operators[5] retrieves a set of documents that agree with the Boolean constraints. However, within the set the documents are ranked according to the weighting scheme of the system. Only the expanded query is given below.

- *BOOL*, $s + n$ *expansion*:

```
#band(#or(#1(radioactive waste) #1(nuclear waste)
          #1(#0(high active) waste) #1(#0(low active) waste)
    #or(process storage stock store repository))
```

In a SUM-of-synonym-groups-query (*SSYN*) each facet forms a clause with the #syn operator. The #syn clauses are combined with the #sum operator. Only the expanded query is given below.

- *SSYN*, $s + n$ *expansion*:

```
#sum(#syn(#1(radioactive waste)#1(nuclear waste)
          #1(#0(high active) waste) #1(#0(low active) waste)
    #syn(process storage stock store repository))
```

SUM (average of the weights of keys) queries represent weak structures. An unexpanded SUM query a single key, or a set of single keys corresponding to the term, represents each original concept. In expansions all expressions were added as single words, i.e., no phrases were included.

- *SUM, s + n expansion*:

  #sum(radioactive waste nuclear waste #0(high active) waste #0(low active) waste process storage stock store repository)

4.2. Test results

The average number of facets in the strongly structured queries was 3.7. In the proximity queries the number of facets was pruned to 2.3. The average number of concepts in the unexpanded queries was 4.9, and in the unexpanded proximity queries 2.7. The average number of search keys (i.e., matching patterns) in the unexpanded queries when no phrases were marked (i.e., SUM queries) was 6.1, and in the expanded queries without phrases as follows: s + n expansion 30.6; full expansion 62.3. The number of search keys with phrases was as follows: no expansion 5.4; s + n expansion 24.4; full expansion 52.4, on average. For the proximity queries the corresponding average figures were 3.3 search keys in the unexpanded queries; 13.4 in the s + n expanded queries; 34.3 in the fully expanded queries. The test results are given as P-R curves (average precision at 10 recall points [10–100]).

Figure 3 shows the P-R curves for the proximity queries. Without expansion the performance of the ProxOr and ProxSyn queries is equal, which is not surprising because most of

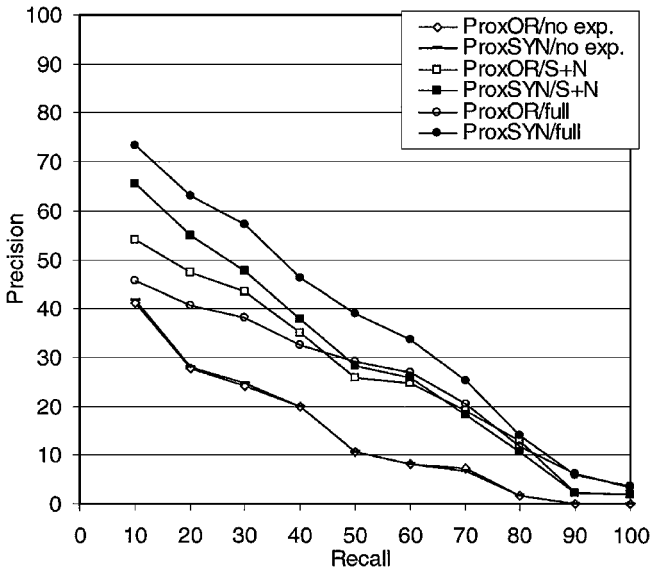

Figure 3. P-R curves for proximity queries with and without expansion.

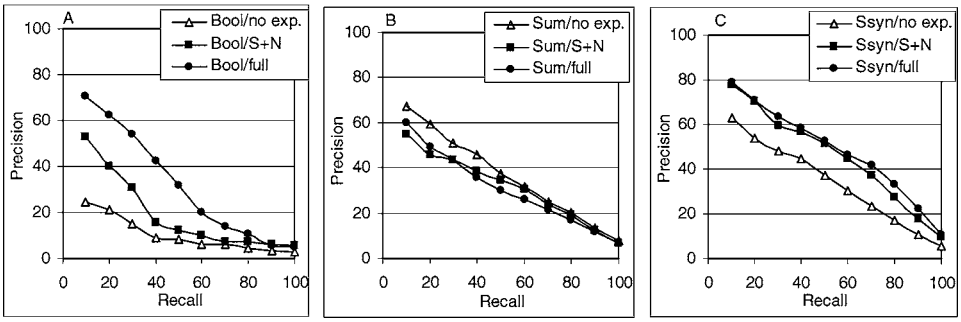

Figure 4. P-R curves for BOOL, SUM and SSYN queries with and without expansion.

the queries do not have more than one concept in a facet, and the query reduces to a simple proximity query. The S + N expansion enhances performance in both query types; the ProxSyn queries are slightly more effective at high precision levels. An interesting difference in performance between the query types arises with the full expansion: the performance of the ProxSyn queries still enhances while the performance of the ProxOr queries drops (when recall $<80\%$). One should bear in mind that the operator just ranks the result set obtained by a proximity search. The low precision figures after 80% recall show that the unexpanded proximity queries actually never retrieve all relevant documents.

For Boolean queries the expansion also proves useful (figure 4a). The difference between the unexpanded and fully expanded queries is considerable, especially at high precision levels (recall $<50\%$).[6] The unexpanded best match query types, SUM and SSYN, have almost similar performance (figure 4(b) and (c)). However, QE has different effects on them: the performance of the SUM queries decreases slightly but the performance of the SSYN queries increases markedly. The best results overall are achieved with the fully expanded SSYN queries.

5. Discussion

The proposed query construction and expansion tool, the ExpansionTool, is intended for use prior to the initial search for natural language text retrieval in heterogeneous document collections lacking intellectual indexing. It has the following desirable features:

- It supports automatic construction and expansion of queries with adjustable query structures and other expansion parameters. Thus queries may be expanded without requiring the users to understand query structures and their interaction with expansion in various retrieval environments.
- Concept-based query formulation and QE are performed at three levels of abstraction (the conceptual, linguistic and string levels). First, concepts representing requests are selected from a conceptual model, second, queries are formulated in which the number of concepts, the number of search keys representing the concepts, the query structure, assumed indexing, and query language may vary.

- It provides a uniform representation for all conceptual (or terminological) relationships and an expansion operator, which supports expansion from selected nodes toward selected (semantic) directions, to an adjustable distance, and/or until (as long as) an adjustable weighting criterion is fulfilled.
- It takes target database indexing into account in the specific formulation of individual search keys. Stemming, word normalization and/or compound splitting for the index are hidden from the user.
- It automatically converts the query into the requested query language (among available languages) and hides differences due to query language paradigms, operator names and expressive power from the user.

The query expansion and construction examples show that the ExpansionTool makes it easy to generate a range of quite differently behaving queries to a number of search environments which are heterogeneous with respect to the overall retrieval strategy, query language properties and database index construction strategies. Therefore the tool greatly supports experimentation with query structures, expansion and other query construction parameters. The ExpansionTool has been fully implemented in Prolog.

Construction of conceptual models is intellectual work and our sample model is hand crafted. However, many phases of the construction process could be automated or automatically supported. For example, candidates for semantic relations could be found through word co-occurrencies or linguistic analysis of texts, or from dictionaries. Construction of matching patterns could be supported by morphological analysis (i.e., by automatically producing word stems, basic forms or components of the compound words). Also, the integrity of relations could automatically be checked—as in any thesaurus tool. Conceptual models are viable if the subject area has a stable conceptual structure and users demand high recall.

In the ExpansionTool query construction is to be started from concepts. It is also possible to map user's own search keys to the model by matching keys to the expressions representing concepts. If there is no match, users could combine their own keys with the keys found in the model. The meaning of each term representing a concept becomes evident from the context the model gives to it. Because the ExpansionTool supports query construction for search environments without vocabulary control at storage phase, the ambiguity in texts cannot be eliminated. However, we believe that ambiguity is reduced through other search concepts (Kekäläinen 1999, and Järvelin 2000).

So far, we have only used the weights representing the strength of semantic relations to select the expansion concepts and their matching patterns. We are planning to test the effectiveness of these weights in calculating weights for the search keys for queries.

We have developed a limited web version of the ExpansionTool, called CIRI (for Concept-based Information Retrieval Interface). CIRI allows consulting several conceptual models, and presents them as a concept networks. The user chooses concepts by navigating the networks. When completed, the server component constructs and supplies the corresponding expanded query, which is then run on a document server. The resulting documents are presented in the CIRI interface. CIRI is fully implemented in Java and uses Java servlets and a relational database for conceptual model storage and query construction. At the moment CIRI constructs only Boolean queries. CIRI also has a subsystem for conceptual model creation and maintenance. Thus domain knowledge may be collected from the users interactively

(Paice 1991, Croft and Das 1990). The matching patterns are generated semi-automatically from expressions. Further automation would require integration with NLP-tools.

CIRI and ExpansionTool constitute a fully conceptual approach to query construction and bypass user's direct interaction with search keys and query expressions by concept network navigation. The main application areas of the ExpansionTool are query interfaces and filter agents for networked IR. Obviously, the ExpansionTool approach can be utilized in improving the parametrizability and matching expressions of information filter agents in networked heterogeneous database environments.

6. Conclusions

The ExpansionTool is a versatile tool for concept-based query expansion and construction for multiple heterogeneous database environments. It is based on three abstraction levels: the conceptual, linguistic and string levels. Concepts and relationships among them are represented at the conceptual level. The expression level gives natural language expressions for concepts. Each expression has one or more matching patterns at the string level. The patterns specify the matching of the expression in database indices built in varying ways. Conceptual expansion is implemented by a novel operator for traversing collections of cyclic concept networks. The number of links expanded, their types, and weights may vary.

The ExpansionTool is a powerful tool intended for end users to support automatic constructing and expanding effective queries so that they need not understand query structures and their interaction with expansion in various heterogeneous retrieval environments. It also is a research tool that supports experimentation with query structures, expansion and other query construction parameters in similar retrieval environments. The empirical sample retrieval experiment demonstrated that the ExpansionTool supports easy construction of quite differently behaving queries for IR experiments.

Appendix

Our sample conceptual model is CM1 = (*c-term1*, *e-strict1*, *e-all1*, SYN1, SPEC1, GEN1, ASS1) as follows.

```
CM1 =
({<c4, t40>, <c5, t50>, <c6, t60>, <c7, t70>, <c8, t80>, <c9, t90>,
    <c10, t100>, <c11, t110>, <c12, t120>, <c13, t130>, <c14, t140>},
 {(t40, phra(2, <bw(radioactive), bw(waste)>)),
      (t50, phra(2, <bw(<nuclear), bw(waste)>)),
      (t60, phra(2, <cw(<bw(low), bw(active)>), bw(waste)>)),
      (t70, phra(2, <cw(<bw(high), bw(active)>), bw(waste)>)),
      (t80, phra(2, <bw(fission), bw(product)>)>,
        <t90, phra(2, <bw(spend), bw(fuel)>)),
      (t100, bw(storage)), (nt101, bw(store)), (nt102, bw(stock)),
      (t110, bw(repository)), (t120, bw(process)), (t130, bw(refine)), (t140, bw(treat))},
 {(t40, {phra(2, <bw(radioactive), bw(waste)>),
        prox(2, <bw(radioactive), bw(waste)>, 3)}),
```

(t50, {phra(2, <bw(<nuclear), bw(waste)>),
prox(2, <bw(<nuclear), bw(waste)>, 3)}),
(t60, {phra(2, <cw(<bw(low), bw(active)>), bw(waste)>),
prox(2, <cw(<bw(low), bw(active)>), bw(waste)>, 3)}),
(t70, {phra(2, <cw(<bw(high), bw(active)>), bw(waste)>),
prox(2, <cw(<bw(high), bw(active)>), bw(waste)>, 3)}),
(t80, {phra(2, <bw(fission), bw(product)>),
prox(2, <bw(fission), bw(product)>, 3)}),
(t90, {phra(2, <bw(spend), bw(fuel)>), prox(2, <bw(spend), bw(fuel)>, 3)}),
(t100, {bw(storage)}), (nt101, {bw(store)}), (nt102, {bw(stock)}),
(t110, {bw(repository)}), (t120, {bw(process)}),
(t130, {bw(refine)}), (t140, {bw(treat)})},
{<t100, {nt101, nt102}>},
{{<c4, c5, 1.0>, <c5, c6, 1.0>, <c5, c7, 1.0>, <c10, c11, 1.0>}},
{{<c5, c4, 0.5>, <c6, c5, 0.5>, <c7, c5, 0.5>, <c11, c10, 0.5>}},
{{<c4, c8, 0.7>, <c8, c4, 0.7>, <c4, c9, 0.6>, <c9, c4, 0.6>, <c5, c8, 0.8>,
<c8, c5, 0.8>, <c5, c9, 0.8>, <c9, c5, 0.8>, <c6, c8, 0.8>, <c8, c6, 0.8>,
<c6, c9, 0.8>, <c9, c6, 0.8>, <c7, c8, 0.8>, <c8, c7, 0.8>, <c7, c9, 0.8>,
<c9, c7, 0.8>, <c12, c13, 0.5>, <c13, c12, 0.5>, <c12, c14, 0.6>,
<c14, c12, 0.6>}}).

Acknowledgments

The InQuery retrieval system was used in part of this research. The InQuery software was provided by the Center for Intelligent Information Retrieval, University of Massachusetts, Amherst, MA, USA. The FIRE Research Group at the University of Tampere has contributed to the paper with many comments. This research was supported, in part, by the Academy of Finland Research Projects 44703 and 49157. This research was completed while the first author was on a leave at the Department of Information Studies, University of Sheffield, UK, fall 2000.

Notes

1. We agree that languages may have some differences at the conceptual level, too. These are not taken into account here.
2. Graphs consist of nodes of any kind and of links between them, and may be either directed or undirected. Connected sequences of links form paths. A directed graph is acyclic if no node can be reached by following any path leaving from it. Otherwise it is cyclic. Undirected graphs are always cyclic. Links between nodes may be weighted to represent their length. Our conceptual QE operator uses collections of cyclic graphs, i.e. sets of possibly overlapping graphs.
3. Using the InQuery query language operators (e.g., Rajashekar and Croft 1995; Kekäläinen and Järvelin 1998) in the structures syns and wsyns. The operators are #sum for the probabilistic sum, #wsum for the weighted probabilistic sum, and #syn for the synonym operator.
4. NB. Examples are translations of Finnish queries for an index containing the basic forms of words, and abbreviated, thus they are illustrative but do not necessarily work as English queries.
5. The Boolean operator AND is denoted by #band and OR by #or in the InQuery query language.

6. NB. The result set is strict Boolean though ranked. All relevant documents are not in the set, thus after 90% recall the curve is not very reliable.

References

Abramson H and Dahl V (1989) Logic Grammars. Springer-Verlag, Heidelberg.

Aho AV and Ullman JD (1992) Foundations of Computer Science. Computer Science Press, New York.

Alkula R (2000) Merkkijonoista suomen kielen sanoiksi. Doctoral Thesis, University of Tampere, Acta Electronica Universitatis Tamperensis, 51. URL: http://acta.uta.fi/pdf/951-44-4886-3.pdf.

Allan J, Callan J, Croft B, Ballesteros L, Byrd D, Swan R and Xu J (1998) INQUERY does battle with TREC-6. In: Voorhees EM and Harman DK, Eds., Proceedings of the Sixth Text REtrieval Conference (TREC-6). NIST Special Publication 500-240, pp. 169–206.

Beaulieu MM, Gatford M, Huang X, Robertson SE, Walker S and Williams P (1997) Okapi at TREC-5. In: Voorhees EM and Harman DK, Eds., Information Technology: The Fifth Text Retrieval Conference (TREC-5). National Institute of Standards and Technology, Gaithersburg, MD, pp. 143–166.

Belkin N, Cool C, Croft WB and Callan JP (1993) The effect of multiple query representations of information retrieval performance. In: Korfhage R, Rasmussen EM and Willett P, Eds., Proceedings of the 16th International Conference on Research and Development in Information Retrieval. ACM, New York, NY, pp. 339–346.

Belkin N, Kantor P, Fox EA and Shaw JA (1995) Combining evidence of multiple query representations for information retrieval. Information Processing and Management, 31:431–448.

Buckley C, Singhal A, Mandar M and Salton G (1996) New retrieval approaches using SMART: TREC 4. In: Voorhees EM and Harman DK, Eds., Proceedings of the 4th Text REtrieval Conference (TREC-4). NIST special publication 500-236, pp. 25–48.

Chang CL and Walker A (1986) A Prolog programming interface with SQL/DS. In: Kerschberg L, Ed., Expert Database Systems: Proceedings from the 1st International Workshop. Benjamin-Cummings, Menlo Park, CA, pp. 233–246.

Crestani F, Sanderson M, Theophylactou M and Lalmas M (1997) Short queries, natural language and spoken document retrieval: Experiments at Glasgow University. In: Harman DK and Voorhees E, Eds., Proceedings of the Sixth Text Retrieval Conference (TREC-6). NIST, Washington DC, pp. 667–686.

Croft WB (1986) User-specified domain knowledge for document retrieval. In: Rabitti F, Ed., Proceedings of the 9th International Conference on Research and Development in Information Retrieval. Pisa, Italy.

Croft WB and Das R (1990) Experiments with query acquisition and use in document retrieval systems. In: Vidick J-L, Ed., Proceedings of the 13th International Conference on Research and Development in Information Retrieval. ACM, Bruxelles, pp. 349–368.

Efthimiadis EN (1996) Query expansion. In: Williams ME, Ed., Annual Review of Information Science and Technology, Vol. 31. Information Today, Medford, NJ, pp. 121–187.

Harman DK (1992) Relevance feedback revisited. In: Belkin N, Ingwersen P and Mark Pejtersen A, Eds., Proceedings of the 15th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. ACM, New York, NY, pp. 1–10.

Hull DA (1997) Using structured queries for disambiguation in cross-language information retrieval. In: AAAI Spring Symposium on Cross-Language Text and Speech Retrieval Electronic Working Notes [online], Stanford University, March 24–26, 1997.

ISO (1986) ISO International Standard 2788. Documentation—Guidelines for the establishment and development of monolingual thesauri. International Organization for Standardization.

ISO (1993) ISO International Standard 8777:1993(E). Information and documentation—Commands for interactive text searching. International Organization for Standardization.

Järvelin K, Kekäläinen J and Niemi T (2001) EXPANSIONTOOL: Formal definition of concept-based query expansion and construction. Report DIS-2001-1, University of Tampere, Department of Information Studies, Finland. Available at http://www.info.uta.fi/julkaisut/pdf/et3.pdf.

Järvelin K and Niemi T (1993) An entity-based approach to query processing in relational databases. Part I: Entity type representation. Data & Knowledge Engineering, 10:117–150.

Järvelin K, Kristensen J, Niemi T, Sormunen E and Keskustalo H (1996) A deductive data model for query expansion. In: Frei H-P, Harman D, Schäuble P and Wilkinson R, Eds., Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, New York, NY, pp. 235–249.

Jing Y and Croft WB (1994) An association thesaurus for information retrieval. In: Proceedings of RIAO, '94, pp. 146–160.

Jones S (1993) A thesaurus data model for an intelligent retrieval system. Journal of Information Science, 19:167–178.

Jones S, Gatford M, Robertson S, Hancock-Beaulieu M and Secker J (1995) Interactive thesaurus navigation: Intelligence rules ok? Journal of the American Society for Information Science, 46:52–59.

Kekäläinen J (1999) The effects of query complexity, expansion and structure on retrieval performance in probabilistic text retrieval. Doctoral Thesis, University of Tampere, Acta Universitatis Tamperensis 678.

Kekäläinen J and Järvelin K (1998) The impact of query structure and query expansion on retrieval performance. In: Croft WB, Moffat A, van Rijsbergen CJ, Wilkinson R and Zobel J, Eds., Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, New York, pp. 130–137.

Kekäläinen J and Järvelin K (2000) The co-effects of query structure and expansion on retrieval performance in probabilistic text retrieval. Information Retrieval, 1:329–344.

Kristensen J (1993) Expanding end-user's query statements for free text searching with a search-aid thesaurus. Information Processing and Management, 29:733–745.

Niemi T and Järvelin K (1992) Operation-oriented query language approach for recursive queries—Part 1. Functional definition. Information Systems, 17:49–75.

Paice CD (1991) A thesaural model of information retrieval. Information Processing and Management, 27:433–447.

Pereira FCN and Warren DHD (1980) Definite Clause Grammars for language analysis—A survey of the formalism and a comparison with Augmented Transition Networks. Artificial Intelligence, 13:231–278.

Pirkola A (2001) Morphological typology of languages for IR. Journal of Documentation, 57(3):330–348.

Rajashekar TB and Croft WB (1995) Combining automatic and manual index representations in probabilistic retrieval. Journal of the American Society for Information Science, 46:272–283.

Sintichakis M and Constantopoulos P (1997) A method for monolingual thesauri merging. In: Belkin NJ, Narasimhalu AD and Willett P, Eds., Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, New York, NY, pp. 129–138.

Turtle HR (1990) Inference networks for document retrieval. Doctoral Thesis, COINS Technical Report 90-92, University of Massachusetts. Computer and information Science Department.

Turtle HR and Croft WB (1991) Evaluation of an inference network-based retrieval model. ACM Transactions on Information Systems, 9:187–222.

Ullman JD (1988) Principles of Database and Knowledge Base Systems, Vol. I. Computer Science Press, Rockville, MD.

UMLS (1994) UMLS Knowledge Sources, 5th Experimental edition. National Library of Medicine, Bethesda, MD.

Voorhees E (1994) Query expansion using lexical-semantic relations. In: Croft WB and van Rijsbergen CJ, Eds., Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, New York, NY, pp. 61–69.

Xu J and Croft WB (1996) Query expansion using local and global document analysis. In: Frei H-P, Harman D, Schäuble P and Wilkinson R, Eds., Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, New York, NY, pp. 4–11.