Information Retrieval, 2, 337–360, 2000
© 2000 Kluwer Academic Publishers. Manufactured in The Netherlands.

Matching Index Expressions for Information Retrieval

B.C.M. WONDERGEM bernd@cs.kun.nl
P. VAN BOMMEL
TH.P. VAN DER WEIDE
Computing Science Institute,University of Nijmegen Toernooiveld 1, NL-6525 ED, Nijmegen, The Netherlands

Received January 12, 1999; Revised October 4, 1999; Accepted January 25, 2000

Abstract. The INN system is a dynamic hypertext tool for searching and exploring the WWW. It uses a dynamically built ancillary layer to support easy interaction. This layer features the subexpressions of index expressions that are extracted from rendered documents. Currently, the INN system uses keyword based matching. The effectiveness of the INN system may be increased by using matching functions for index expressions. In the design of such functions, several constraints stemming from the INN must be taken into account. Important constraints are a limited response time and storage space, a focus on discriminating (different notions of) subexpressions for index expressions, and domain independency. With these contextual constraints in mind, several matching functions are designed and both theoretically and practically evaluated.

Keywords: information retrieval, similarity, index expressions, matching

1. Introduction

Searching large and dynamic information spaces is a difficult task. Many authors have opted for an approach combining browsing and searching in a hypertext environment (see e.g. Lucarella and Zanzi (1993), Wilkinson and Fuller (1996) and Bruza and van der Weide (1992)). The INdex Navigator (INN) (Wondergem et al. 1999) is a dynamic hypertext tool offering this combined approach for the WWW. The INN uses index expressions (Bruza and van der Weide 1992) to construct a topic overview based on linguistically motivated refinements. In this overview, the hyperindex, navigation is used for query formulation. After formulation, the query (an index expression) is used to render relevant documents. The INN currently uses existing search engines for rendering, utilising only the terms (keywords) in the query. Thus, relations between terms, as modeled in the query, largely get lost. Exploiting similarity measures for index expressions in rendering, e.g. in re-ranking the documents, would alleviate this problem.

The intended use of the similarity measures in the INN poses a number of constraints on their design. The documents that are rendered by the search engines are likely to contain (most of) the required keywords from the query. However, *inter-term relations*, as modeled in index expressions, do not have to be satisfied since matching is keyword-based. This may be repaired by re-ranking the rendered set of documents according to similarity measures that go beyond keywords. Thus, the similarity measures should focus on inter-term relations and, consequently, on the *hierarchical composition* of index expressions.

The WWW contains documents on a very wide range of topics. Therefore, the INN has been designed as a *domain independent* exploration and search tool. This means that domain dependent approaches, such as conceptual IR (Mauldin 1989) and coordinated concepts (van der Vet and Mars 1998), which make heavy use of domain specific knowledge bases, are not adequately general for our purposes.

The use of thorough (linguistic) analysis is not viable for the INN due to the need for quick responses and a limited amount of available memory. Since the INN runs on a web server, it potentially serves many users which may cause an explosion of the required memory. This especially holds if considerable (linguistic) knowledge is used, for instance in detailed parsing. Similarity values have to be computed real time, next to rendering documents form search engines, locating and extracting index expressions from these documents and parsing them. We thus focus on matching of *basic syntactic refinement structure* of phrases, as captured in index expressions. In turn, this means that not all ambiguities encountered in content analysis can be resolved.

The refinements in the hyperindex directly stem from a particular view on the decomposition of index expressions, or, a *notion of subexpressions*. For instance, the order of subexpressions may be deemed relevant in computing the refinements of the hyperindex. In addition, there are different views on embedding of index expressions. It is envisaged that the user may select his preferred notion of subexpression for constructing the hyperindex. Otherwise, the preferred notion may be derived by analysing the (types of) refinements the user makes during navigational query formulation. In order to render documents consistent with the user's preferences, the similarity measure used for re-ranking should be in accordance with the preferred notion of subexpressions.

The presented topic overview in the INN is based on a small subset (10–100 documents) of the complete collection. Statistics thus only apply to a very small part of the information space. As a consequence, (domain dependent) collection statistics cannot be globally computed. In addition, term weights for the query are not supplied by the INN system. In this article, therefore, we focus on the core function of our similarity, the structure of index expressions and provide abstract functions for comparing terms.

The goal of this article is the design of (relatively simple and efficient) similarity measures that focus on the structure of index expressions. In particular, the order of subexpressions and different notions of embedding are to be modeled in the similarity measures.

The structure of this article is as follows. In Section 2, our approach is put in the context of other work on linguistically motivated indexing and structural matching. Section 3 introduces two representations of index expressions. In Section 4, elaborates several topics concerning the contents and structure of index expressions. These topics lead to the design of several similarity measures for index expressions in Section 5. Section 6 provides an experimental evaluation of the proposed measures. Finally, Section 7 provides concluding remarks and directions for further research.

2. Related work

2.1. Obtaining index expressions from text

Currently, the INN system directly extracts index expressions from the titles of the rendered documents. This produces acceptable results since titles are often in a form which permits a ready transformation to index expressions. A similar point of departure is taken in a closely resembling system, the hyperindex browser (Iannella et al. 1995). A simple parsing method based on a two-level priority scheme for connectors (Bruza 1993) is used. All allowed connectors are listed together with their priority. All other words encountered are interpreted as terms (keywords). A stoplist (list of stopwords) filters out unwanted words like, for instance, articles and determiners. During parsing, the priority of the connector at hand determines if the already parsed part of the index expression has to be broadened or deepened. Thus, no (explicit) linguistic grammar is used by the parser. The parser results in one parse tree only, thus possibly neglecting different interpretations. A test on the CACM document collection showed that this parsing method was accurate in approximately 90 percent of the cases.

No additional sources of linguistic knowledge are used in the INN. Although our approach is rather simplistic and although "syntax by itself cannot resolve the many ambiguities that complicate the content analysis task" (Salton and Smith 1989), in our view, the advantages offered by the INN system, augmented with similarity measures for index expressions, produce a workable balance between keyword based approaches and sophisticated linguistic analysis.

A more general approach to indexing documents with index expression could make use of a part-of-speech (POS) tagger and an extractor. For closely related descriptors, called phrase frames, this approach is described in Arampatzis et al. (1998). Phrase frames are head-modifier structures in which connectors are (eventually) omitted. After preprocessing the documents, filtering non-textual parts and moulding the text into acceptable format, the POS tagger labels words with their part of speech. A well performing POS tagger is described in Brill (1994). The extractor, which is applied next, uses the POS labels in identifying the linguistic constructs that should be parsed as index expressions. The extractor uses pattern matching with regular expressions. An extractor for index expressions would constitute a subset of the patterns used for phrase frames, albeit augmented with connectors. Finally, the same parser as above can be used, although more sophisticated parsers are available. This is, however, not in the scope of this article.

The general architecture of an NLP-IR system advocated by Strzalkowski (1995) also includes a stemmer. Normalisation at term level (e.g. stemming) is largely orthogonal to our approach, so possible to include, but not considered in this article. However, we provide abstract similarity functions for terms and connectors that could be instantiated by measures that do take, for instance, stemming into account.

Regularisation or syntactical normalisation at phrase level aims at recall enhancement by reducing syntactical variation (Strzalkowski 1995). For phrase frames, syntactic normalisation is described in Arampatzis et al. (1998). This approach can, mutatis mutandis, also be applied for index expressions since phrase frames closely resemble index expressions. However, this requires the use of additional linguistic knowledge. We do not elaborate on this issue here.

Instead of normalising descriptors, systematically generating (all) linguistically motivated alternative variants (Sparck Jones and Tait 1984) has a similar aim. Semantic linguistic knowledge is exploited not to generate invalid variants. Although this restricts the number of generated variants, this number may explode, especially for complex composed descriptors. Matching is done via translation of the generated variants into a Boolean query.

Due to the need for efficient matching in the INN system, we do not generate all variants of index expressions.

The Constituent Object Parser (Metzler and Haas 1989) also uses syntactic structure for matching descriptors. It produces binary dependency trees, which indicate (directed) dependency and scope of relationships between terms. Many different syntactic forms are represented by this single dependency relation. As a result, ambiguities are retained in the representation (Strzalkowski 1995). A comparable approach is that of using tree structured analytics (Smeaton and Sheridan 1991). Both approaches use external sources of linguistic knowledge and offer at best reasonable efficiency. Index expressions make a distinction between the "types" of dependency, as given by connectors. Not taking the actual (contents of) connectors into account, e.g. by allowing wildcard connectors, in matching normalised index expressions effectuates a similar (directed) notion of dependency. An intermediate form of dependency, colliding connectors of the same type, is described in Section 4.1.2.

The CLARIT system (Evans et al. 1991, Evans et al. 1992) integrates several NLP techniques with the vector space retrieval model. These techniques include morphological analysis, robust noun-phrase parsing, and automatic construction of first order thesauri. Construction of the thesauri, used to support the selection of appropriate terms, requires a substantial (minimally 2GB) sample of documents about a certain topic. Because of limited response times and available memory, we expect this approach too demanding for our application.

2.2. *Comparing structured descriptors*

The similarity measures for index expressions described in Section 5 focus on the structure of index expressions, i.e., on the way subexpressions are nested. Three similarity measures are provided: skeleton content, full-product, and a twig-based measure. These measures represent different views on the order of subexpressions and embedding. Headedness, or concept modification in head-modifier structure, is considered as well.

In Kilpelaïnen and Mannila (1993), a language for querying structured text based on tree inclusion is described. Their approach, which exploits inclusion patterns to ensure preservation of binary properties between nodes, takes both structure and content into account. Example inclusion patterns are L for labels, A for ancestorship, and O for (left-to-right) ordered tree inclusion.

Our skeleton-content approach resembles their $\{LAO\}$-embedding. That is, ancestorship and ordering are preserved and labels are taken into account. However, our approach does not require labels to be equal but supports approximate matching by offering abstract similarity functions for terms and connectors. In the introduction of the mentioned article the authors indicate that, indeed, such "standard IR techniques should be added to the language". Our similarity measures for index expressions capture both aspects by approximation: they are sensitive to both structure and, by using abstract term comparison functions, content.

Our skeleton-content approach thus 'preserves' labels by taking into account their similarity. Essentially, it searches for the best $\{L'AO\}$-embedding and delivers the degree of embedding. A similar line of reasoning shows that our full product approach computes the degree of $\{LA\}$-embedding. Since in the full product measure the order of subexpressions is considered irrelevant, the corresponding inclusion pattern O is not satisfied.

In Arampatzis et al. (1998), an approach to compare head-modifier structures is described. Lexico-semantical relations are used in computing the similarity between their binary terms. Our set based similarity measures for twigs do not explicitly make use of a semantical network. However, this may be incorporated in our abstractly modeled comparison functions for terms and connectors. In addition, twigs are tertiary structures, also including information on the type of connection between head and modifier. Furthermore, we focus on similarities between *sets of* twigs, rather than on their individual similarities.

3. Index expressions

Index expressions can be represented in several ways (see e.g. Wondergem et al. (to appear)). Each representation has its own advantages and disadvantages. We use the *inductive* and the *structural* representations. These representations describe the same language of index expressions but differ in denotational properties. Index expressions are based on terms and connectors. Terms denote keywords, concepts names, adjectives, gerunds, and attribute values. Connectors denote relations between terms in the form of prepositions (showing place, position, time, or method) and some present participles (e.g. using, having, and being).

3.1. Inductive representation

The inductive representation, as described in Definition 3.1, is used since it most basically describes the (de)composition of index expressions. The underlying idea is that index expressions can be augmented with subexpressions through connectors, as illustrated in figure 1. An advantage of such an elementary representation is that it allows several auxiliary functions to be readily designed.

Definition 3.1. Let T be a nonempty set of terms and C be a set of connectors such that $T \cap C = \emptyset$. Then, the *language of non-empty index expressions* is defined as:

1. if $t \in T$, then t is a non-empty index expression, and
2. if I and J are non-empty index expressions and $c \in C$ is a connector, then add (I, c, J) is also a non-empty index expression, and
3. no other non-empty index expressions exist.

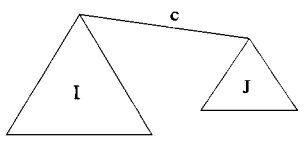

Figure 1. Basic setup of inductive representation.

$$\begin{array}{lcl}
\mathsf{Terms}(t) & = & \{t\} \\
\mathsf{Terms}(\mathsf{add}(I,c,J)) & = & \mathsf{Terms}(I) \cup \mathsf{Terms}(J) \\
\\
\mathsf{Conns}(t) & = & \emptyset \\
\mathsf{Conns}(\mathsf{add}(I,c,J)) & = & \mathsf{Conns}(I) \cup \{c\} \cup \mathsf{Conns}(J) \\
\\
\mathsf{Head}(t) & = & t \\
\mathsf{Head}(\mathsf{add}(I,c,J)) & = & \mathsf{Head}(I)
\end{array}$$

Figure 2. Auxiliary functions on index expressions.

Example 3.1. Single word queries or document representations are modeled by terms. Example terms are conference, biology, and Holland. Composed index expressions can be constructed through the add operator. This also exploits connectors, such as in, with, and on. For instance, the composed index expression add(conference, on, biology) represents information on a conference on biology. As a more complex example, consider add(add(conference, on, biology), in, Holland) denoting information on a conference on biology held in Holland. The semantics of index expressions depends on their structure. That is, differences in nested subexpressions may cause differences in semantics. As an example of this, compare the last index expression with the slightly different add(conference, on, add(biology, in, Holland)). The last one denotes a conference about biology as far as it is practiced in Holland.

Three auxiliary functions on index expressions are introduced in figure 2. These functions are used in defining similarity measures. The definitions are formulated in terms of the inductive definition of index expressions. The functions result in the terms of an index expression, its connectors, and its head, respectively.

3.2. *Structural representation*

The inductive representation provides a horizontal decomposition of index expressions: subexpressions are added on the right. Another representational formalism for index expressions, called the structural representation, serves better if all subexpressions at a certain depth have to be addressed at the same time. The structural representation provides a vertical decomposition of index expressions, allowing a direct and clear look on their structure. The structural representation is exploited in cases where the order of subexpressions is to be taken into account. In the structural representation, a composed index expression is denoted by

$$h \otimes_{i=1}^{k} c_i(I_i) \equiv h c_1(I_1)..c_k(I_k)$$

where h is the head and the k subexpressions I_i are connected with the head by connectors c_i. The subexpressions are denoted by the structural representation as well. The composition operator $\otimes$ provides a notational shorthand.

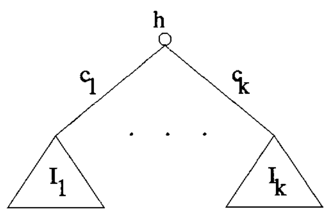

Figure 3. Structural representation.

Figure 3 gives a schematic view on the structural representation.

Example 3.2. The two last mentioned index expressions of Example 3.1 are denoted in the structural representation as conference on (biology) in (Holland) and conference on (biology in (Holland)), respectively. Note that in the structural representation the differences in semantics are modeled by brackets.

4. Issues concerning semantics

The index expression

$$hc_1(I_1)..c_k(I_k)$$

can be seen as a description of a concept named h being further refined by relations called $c_1..c_k$ and concepts $I_1..I_k$, respectively.

As a consequence, comparison of both concepts names (contents) as well as the refining mechanism (structure) will be important issues when matching index expressions. This section presents several topics concerning contents and structure that refine the abovementioned view on the semantics of index expressions.

4.1. Contents

The contents of index expressions is given by their terms and connectors.

4.1.1. Terms. In order to match index expressions, their terms should be compared. Therefore, we assume a similarity function between terms, denoted $\mathsf{sim}_T : T \times T \rightarrow [0..1]$. The expression $\mathsf{sim}_T(t, t')$ denotes the similarity between the concepts referred to by terms t and t'.

The similarity between terms can be obtained in several ways. For instance, it can be computed by string comparison algorithms such as n-grams, the overlapping substrings of terms of length n. Furthermore, additional lexico-semantical knowledge can be used taking, for instance, hypernyms and synonyms into account. In IR, stemming is often performed to reduce morphological variance. We abstract from the particular techniques used in computing the similarity and concentrate on using this in computing similarity between index expressions.

4.1.2. Connectors. Similar to terms, we also assume a similarity function $\mathsf{sim}_C : C \times C \to [0..1]$ between connectors. This similarity expresses the strength of the relation between connectors.

The similarity function for connectors can take several aspects into account. For instance, it can be based on the types of connectors as identified by Farradane (see Farradane (1980a, b)). In this approach, called relational indexing, connectors model the relationships between terms. Connectors that model the same relationship could be given a high similarity value. Furthermore, the priority of connectors within index expressions can be exploited. In addition, occurrence-frequencies of connectors could be used. Again, we abstract from the different approaches and focus on exploiting similarity between connectors for matching index expressions.

4.2. Structure

The structure of index expressions partly determines their semantics and should, therefore, be taken into account in matching. Three issues concerning structure are considered in this article: the *order of subexpressions*, *embedding*, and *headedness*.

4.2.1. Order of subexpressions. An important question considering the semantics of index expressions is whether the order of subexpressions is relevant. Consider for example the following index expressions.

hiking in mountains with friends
hiking with friends in mountains

One may argue that their meaning is equivalent. In other situations, however, there may be cases in which the order of subexpressions is relevant, i.e., causes a different meaning. For instance, if the sequential order of paragraphs in a text is represented in an index expression.

The notion of order of subexpressions is formalised by the relation `EqOrder` as follows.

Definition 4.1. We call two index expressions $I = h \otimes_{i=1}^{k} c_i(I_i)$ and $J = h' \otimes_{j=1}^{l} d_j(J_j)$ equal modulo order, denoted by $\mathtt{EqOrder}(I, J)$, iff

1. $h = h'$, and
2. there exists a projection π of $[1..k]$ to $[1..l]$ such that for all $1 \leq i \leq k$ it holds that $c_i = d_{\pi(i)}$ and $\mathtt{EqOrder}(I_i, J_{\pi(i)})$.

Two index expressions are equal modulo order iff their heads are equal and (recursively) the subexpressions of the first are equal modulo order to one of the subexpressions of the second. The second criterion holds, for instance, for index expressions of which the subexpressions (recursively) are a permutation of the other's. In addition, it holds for index expressions which contain several copies of the same subexpressions. This is, for instance, illustrated by the index expressions retrieval of information and retrieval of (information) of (information). Furthermore, equality modulo order is reflexive. Later in this article, we describe a matching strategy that computes maximal similarity for index expressions that are equal modulo order.

4.2.2. Embedding. Embedding or containment plays a prominent role in IR. For example, an often applied strategy to query-document matching is: if the query is contained or embedded in a document, then the document is deemed relevant to the query.

Different notions of embedding exist. The subexpression relation for index expressions, for example, defines a *connected* variant of embedding: a subexpression is a connected part of its superexpressions. For instance, surfing in Holland is a subexpression of surfing in Holland in November but not a subexpression of surfing in sunny Holland. The subexpression relation (see Wondergem et al. (to appear)) is denoted by $\preccurlyeq$. The expression $I \preccurlyeq J$ means that index expression I is a subexpression of J.

In the abovementioned case, sunny modifies the last term Holland of the path expression sunny Holland. In our view, surfing in Holland is therefore embedded in surfing in sunny Holland. To cater for these cases, we exploit a slightly more liberal version of embedment.

Connectedness can, in the context of subexpressions, be described by direct ancestorship, or, parenthood. This means that for all terms in an index expression I, their parent must also be their parent in index expressions in which I is embedded. Instead of direct ancestorship we use the notion of (general) ancestorship. This means that index expression I is embedded in J iff for all terms in I their ancestors in I are also ancestors in J.

These considerations are reflected in our notion of embedding of index expressions which is formalised by an embedment $\ll$ relation as follows.

Definition 4.2. Embedding of index expressions is captured in the binary relation $\ll$, where $I \ll J$ means that I *is embedded in* J, which is defined as:

(Same)	$t \ll t$
(Sub)	$I \ll \mathsf{add}(J, c, K)$ if $I \ll J$ or $I \ll K$
(Split)	$\mathsf{add}(I, c, J) \ll \mathsf{add}(K, d, L)$ if $c = d$ and $I \ll K$ and $J \ll L$
(Stop)	no other cases apply

Terms are embedded in themselves, as described by case Same of Definition 4.2. Case Sub states that an index expression is embedded in a composed one $\mathsf{add}(J, c, K)$, if it is embedded in either subexpression J or K. The third case Split shows that a composed index expression $\mathsf{add}(I, c, J)$ is embedded in another composed index expression $\mathsf{add}(K, d, L)$ if their connectors are equal, the leftmost subexpression I is embedded in the other leftmost subexpression K, and a similar argument holds for the rightmost subexpressions J and L.

4.2.3. Headedness. The head of an index expression is considered to be its most important part. We will address this notion, implemented by head-modifier pairs in Strzalkowski (1995), as *headedness*. The subexpressions modify the main concept stated in the head. Consequently, the lower terms occur, i.e., deeper with respect to nested expressions, the less important they are. This can be taken into account in matching by exploiting the depth of terms. Multiplying by a factor that is inversely proportional to the depth gives the desired result. We will indicate how each similarity function described below can be equipped with such a depth factor.

5. Similarity measures for index expressions

In this section, several similarity measures for index expressions are designed. They vary from simple measures that only consider the content of index expressions (Section 5.1) to more comprehensive measures that consider both content and structure (Section 5.2).

5.1. Contents only

Measuring the similarity between index expressions by sets of terms and connectors considers only their contents, not their structure. Several standard measures for set similarity can be adopted to provide similarity measures for (the contents of) index expressions. Example set measures are the Dice, Cosine, and Jaccard measures (see e.g. Rijsbergen (1975)). These measures do not use similarity functions for terms and connectors as described in Sections 4.1.1 and 4.1.2, respectively. Instead, they use set primitives as union, disjunction, and cardinality which use equality of elements (terms and connectors). The Dice measure, which normalizes the intersection $A \cap B$ with the sum of constituents, the Cosine measure, relating the overlap of both sets to their geometric average, and the Jaccard measure, expressing the degree of overlap between two sets as the proportion of the overlap from the whole, are defined as

$$\mathsf{Dice}(A, B) = \frac{2|A \cap B|}{|A| + |B|} \qquad \mathsf{Cos}(A, B) = \frac{|A \cap B|}{\sqrt{|A| \times |B|}} \qquad \mathsf{Jacc}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

in case the denominator does not equal zero. In case it does, the measures return similarity value zero.

By considering the (sets of) terms and connectors of index expressions, the set based measures can be exploited for index expressions. As an example, consider the similarity measure for index expressions based on the Dice coefficient as shown in figure 4.

The factor α determines the relative influence of terms and connectors on the similarity value. For $\alpha = 1$, only terms are considered and for $\alpha = 0$ only connectors are taken into account. Note that a depth factor cannot be taken into account directly since no information about structure is available in sets of terms.

We say a measure is maximal for certain index expressions if maximal similarity is returned for the descriptors. The Dice measure is not maximal for embedded index expressions

$$\mathsf{Dice}_\alpha(I, J) \quad = \quad \alpha \times \mathsf{Dice}(\mathsf{Terms}(I), \mathsf{Terms}(J)) \quad + \quad (1 - \alpha) \times \mathsf{Dice}(\mathsf{Conns}(I), \mathsf{Conns}(J))$$

Figure 4. Dice's similarity measure for index expressions.

nor for subexpressions. Consider, for example, index expression $\mathsf{surfing\ in\ Holland}$ and its subexpression $\mathsf{Holland}$. Since the sets of terms and connectors are different, the Dice measure is not maximal for these index expressions.

5.2. *Contents and structure*

This section provides three similarity measures that take both content and structure of index expressions into account. The first measure, coined *full product*, adheres to the idea that the order of subexpressions is irrelevant for the meaning of index expressions. On the contrary, the second measure, called *embedded content*, deems the order relevant. In addition, embedded content is based on non contiguous embedding. Finally, the *twigs* measure is based on decomposing index expressions into elementary connections called twigs.

5.2.1. Full product. The *full product* similarity measure computes the degree to which an index expression is equal modulo order with another one. This means that the order of subexpressions is considered irrelevant. Since the structural representation appears most appropriate in this case, the full product measure, denoted by FP, is specified by it as depicted in figure 5.

The full product algorithm, as shown in figure 5, consists of four cases. They can be readily translated to a functional algorithm using pattern matching. The first case Terms computes the similarity between single terms. In case Top, the embedment of a single term t in a composed index expression $h \otimes_{i=1}^{k} c_i(I_i)$ is computed by comparing t with head h since both occur at the same depth.

The third case Tall computes the similarity between two composed index expressions. Correspondingly to Definition 4.1, this is the product of the similarity between the heads and the maximum similarity values of the subexpressions of I with some subexpression of J.

The final case Toll gives a penalty for the fact that a composed index expression cannot be equal modulo order with a term. The returned similarity value is smaller if the mismatch in size (number of terms) is larger.

Note that the full product measure computes a similarity value layer by layer. That is, only terms and connectors that occur at the same depth are compared. The full product

$$\begin{array}{lll}
(\mathsf{Terms}) & \mathsf{FP}(t,t') = \mathsf{sim}_T(t,t') \\
(\mathsf{Top}) & \mathsf{FP}(t, h \otimes_{i=1}^{k} c_i(I_i)) = \mathsf{sim}_T(t,h) \\
(\mathsf{Tall}) & \mathsf{FP}(h \otimes_{i=1}^{k} c_i(I_i), h' \otimes_{j=1}^{l} d_j(J_j)) = \\
 & \mathsf{sim}_T(h,h') \times \frac{1}{k}\Sigma_{i=1}^{k}\ \mathtt{max}_{1\le j\le l}\ \mathsf{sim}_C(c_i,d_j) \times \mathsf{FP}(I_i,J_j) \\
(\mathsf{Toll}) & \mathsf{FP}(h \otimes_{i=1}^{k} c_i(I_i), t) = \frac{\mathsf{sim}_T(h,t)}{|\mathsf{Terms}(h \otimes_{i=1}^{k} c_i(I_i))|}
\end{array}$$

Figure 5. Full product algorithm.

similarity measure is maximal for index expressions that are equal modulo order, as shown in the following theorem.

Theorem 5.1. $\mathtt{EqOrder}(I, J) \Rightarrow \mathsf{FP}(I, J) = 1$.

Proof: The proof is done by structural induction to index expressions. We only investigate the (interesting) case of composed index expressions. Suppose $I = h \otimes_{i=1}^{k} c_i(I_i)$ and $J = h' \otimes_{j=1}^{l} d_j(J_j)$ are equal modulo the order of their subexpressions. This means that for each $hc_i(I_i)$ the maximum value $\mathtt{max}_{1\le j\le l}\mathsf{FP}(hc_i(I_i), h'd_j(J_j)) = 1$ since

1. $h = h'$, meaning $\mathsf{sim}_T(h, h') = 1$, and
2. for each $1 \le i \le k$ there exists a $1 \le j \le l$ such that $d_j = c_i$ and $\mathtt{EqOrder}(I_i, J_j)$ meaning that $\mathsf{sim}_C(c_i, d_j) = 1$ and, by the induction hypothesis, $\mathsf{FP}(I_i, J_j) = 1$.

This means that, for the j of case (2), $\mathsf{sim}_T(h, h') \times \mathsf{sim}_C(c_i, d_j) \times \mathsf{FP}(I_i, J_j) = 1$. The total similarity measure then comes down to $\frac{1}{k}\sum_{i=1}^{k} 1 = 1$. □

As a direct consequence of Theorem 5.1, we conclude that the full product similarity measure is maximal for equal arguments, i.e. that $\mathsf{FP}(I, I) = 1$. In addition, full product computes maximal similarity for index expressions of which the subexpressions (recursively) are a permutation of the other's or occur multiple times.

The full product measure is not maximal for embedded index expressions. Consider, for example, index expressions surfing in Holland and surfing in sunny ∘ Holland. Since the full product measure computes similarity values layer by layer and the keyword Holland resides at different depths, the given index expressions are not maximally similar. In addition, the full product measure is not maximal for subexpressions. Illustrating this, consider Holland and surfing in Holland. Since the heads of both index expressions are unequal, the full product measure is not maximal.

$$\begin{aligned}&\mathsf{FP}_{\mathtt{d}}\big(h \otimes_{i=1}^{k} c_i(I_i), h' \otimes_{j=1}^{l} d_j(J_j)\big)\\ &\quad= \mathsf{sim}_{\mathtt{d},T}(h, h') \times \frac{1}{k}\sum_{i=1}^{k} \mathtt{max}_{1\le j\le l}\ \mathsf{sim}_{\mathtt{d+1},C}(c_i, d_j) \times \mathsf{FP}_{\mathtt{d+1}}(I_i, J_j)\end{aligned} \tag{1}$$

The depth factor d can be incorporated in the full product algorithm (figure 5) by replacing case Tall by Eq. (1). Note that it is only shown how the depth factor is to be correctly computed in this case. Multiplying each case by a factor inversely proportional to this depth factor takes headedness into account.

5.2.2. Embedded content. The *embedded content* measure (see figure 6) computes the best way in which an index expression can be embedded (as defined in Definition 4.2) in another one. This means that the order of subexpressions is considered relevant. This eases the use of the inductive representation of index expressions and thus enables an elementary decomposition of index expressions.

The cases considered by the embedded content measure are the same as for the full product measure. The case Terms of figure 6 is exactly the same as for full product and calls the similarity function for terms sim_T. The second case Top states that the degree to which a term

$$\begin{array}{lrcl}
\text{(Terms)} & \mathsf{EC}(t,t') & = & \mathsf{sim}_T(t,t') \\
\text{(Top)} & \mathsf{EC}(t,\mathsf{add}(I,c,J)) & = & \mathtt{max}\{\mathsf{EC}(t,I),\mathsf{EC}(t,J)\} \\
\text{(Tall)} & \mathsf{EC}(\mathsf{add}(I,c,J),\mathsf{add}(K,d,L)) & = & \\
 & \multicolumn{3}{l}{\mathtt{max}\{\mathsf{EC}(\mathsf{add}(I,c,J),K),\ \mathsf{EC}(\mathsf{add}(I,c,J),L),} \\
 & \multicolumn{3}{l}{\qquad \mathsf{EC}(I,K)\times\mathsf{sim}_C(c,d)\times\mathsf{EC}(J,L)\}} \\
\text{(Toll)} & \mathsf{EC}(\mathsf{add}(I,c,J),t) & = & \frac{\mathsf{sim}_T(\mathsf{Head}(I),t)}{|\mathsf{Terms}(\mathsf{add}(I,c,J))|}
\end{array}$$

Figure 6. Embedded content algorithm.

t is embedded in a composed index expression $\mathsf{add}(I, c, J)$ is equal to the maximal similarity to one of the subexpressions I and J. Case Tall computes the strength of embedding of a composed index expression $\mathsf{add}(I, c, J)$ in another one $\mathsf{add}(K, d, L)$. It computes the maximum similarity of the following three cases: (1) $\mathsf{add}(I, c, J)$ is completely embedded in the leftmost subexpression K, (2) it is completely embedded in the rightmost subexpression L, and (3) subexpression I is embedded in K, subexpression J is embedded in L, and connectors c and d are similar. As before, case Toll gives a penalty for mismatch since composed index expressions can never be embedded in terms.

The embedded content measure is maximal for embedded index expressions, as stated more formally in the next theorem.

Theorem 5.2. $I \ll J \Rightarrow \mathsf{EC}(I, J) = 1$.

Proof: The theorem is proven by induction on the structure of index expressions. Suppose $I \ll J$.

Basis. Suppose I is a term t. Only the first two cases of Definition 4.2 of embedded content, Same and Sub, are to be examined since the third case Split cannot apply (I cannot be composed).

Same: Suppose this case of Definition 4.2 applies. Then, $J = I = t$ and consequently $\mathsf{EC}(I, J) = \mathsf{sim}_T(I, J) = 1$.

Sub: Suppose this case applies which means that J is composed, say $\mathsf{add}(K, c, L)$ and thus that $t \ll K$ or $t \ll L$. Combining this with the induction hypothesis implies that $\mathsf{EC}(t, K) = 1$ or $\mathsf{EC}(t, L) = 1$. This means that, by case Top of the embedded content algorithm, $\mathsf{EC}(I, J) = 1$.

Induction step. Suppose I is a composed index expression, say $\mathsf{add}(K, c, L)$. Furthermore, without loss of generality, let $J = \mathsf{add}(M, d, N)$. Now, only cases Sub and Split of Definition 4.2 need to be examined. Both result in $\mathsf{EC}(I, J) = 1$:

Sub: Here, we have (1) $I \ll M$ implying $\mathsf{EC}(I, M) = 1$ or (2) $I \ll N$ which means that $\mathsf{EC}(I, N) = 1$. Since case Tall of the algorithm is applied, which computes the maximum of these cases, both options cause $\mathsf{EC}(I, J) = 1$.

Split: In this case, we have that $c = d$, $K \ll M$, and $L \ll N$. By the induction hypothesis, this implies that $\mathsf{EC}(K, M) = 1$ and $\mathsf{EC}(L, N) = 1$. Together with $\mathsf{sim}_C(c, d) = 1$ this causes case Tall of the embedded content algorithm to compute $\mathsf{EC}(I, J) = 1$.

We have shown that in all possible cases in which I is embedded in J the embedded content measure computes $\mathsf{EC}(I, J) = 1$. □

The embedded content measure is also maximal for subexpressions. This stems from the observation that every index expression that is a subexpression of another is also embedded in it. This, in turn, follows from the observation that in the subexpression relation, direct ancestorship is preserved, which is stricter than the general ancestorship which is preserved by the embedment relation. In addition, the embedded content measure is maximal for equal arguments, since every index expression is embedded in itself, i.e. embedment is reflexive.

The embedded content measure is not maximal for index expressions that are equal modulo order. Consider, for example, the following pair of index expressions that are equal modulo the order of their subexpressions: $I =$ conference on (biology) in (Holland) and $J =$ conference in (Holland) on (biology). Since in the embedded content measure the order of subexpressions is relevant, index expressions I and J are not maximally similar.

$$\begin{aligned}&\mathsf{EC}_{\mathsf{d}}(\mathsf{add}(I, c, J), \mathsf{add}(K, d, L)) \\ &\quad = \mathsf{EC}_{\mathsf{d}}(I, K) \times \mathsf{sim}_{\mathsf{d+1},C}(c, d) \times \mathsf{EC}_{\mathsf{d+1}}(J, L)\end{aligned} \tag{2}$$

The depth factor d can be incorporated in the inductive representation by the scheme of Eq. (2). Note that the rightmost subexpressions J and L reside one layer deeper than the head.

5.2.3. Twigs.

The *twigs* (see Berger (1998)) of an index expression are its subexpressions that consist of exactly two terms and one connector. Twigs are the elementary connections in the concept graph which is formed by an index expression. Twigs have essentially the same structure as the head-modifier pairs described in Strzalkowski (1995), except that they also contain a connector. Twigs enable us to form a global picture about similarity while focusing in on the elementary refinements.

To denote subexpressions of size two, we use the subexpression relation as described in Section 4.2.2.

$$\mathsf{twigs}(I) = \{tct' \mid tct' \preccurlyeq I\}$$

Twigs can be defined constructively in terms of the inductive representation of index expressions. This shows that the twigs of an index expression can be produced by a straightforward syntactic process. Below, twigs are accompanied by their depth factor, modeled by a positive integer. Since the head of an index expression is assigned depth one, the expression $\mathsf{twigs}(I, 1)$, according to the definition below, computes the twigs of I with their correct depths.

$$\begin{aligned}\mathsf{twigs}(t, d) &= \emptyset \\ \mathsf{twigs}(\mathsf{add}(I, c, J), d) &= \{(\mathsf{add}(\mathsf{Head}(I), c, \mathsf{Head}(J)), d)\} \\ &\quad \cup \mathsf{twigs}(I, d) \cup \mathsf{twigs}(J, d+1)\end{aligned}$$

Single terms do not contain subexpressions of size two and thus have no twigs. Composed index expressions $\mathsf{add}(I, c, J)$ lead to at least one twig: $\mathsf{Head}(I)\ c\mathsf{Head}(J)$. Additional twigs may result from the subexpressions, as modeled by the recursive calls.

As observed in Berger (1998), twigs conserve most of the structure of index expressions. In fact, if all terms are different the complete structure can be correctly reconstructed without any additional information. For twigs with depth factor, this can be relieved further to all terms at the same depth should be different.

Twigs of index expressions seem to resemble tri-grams for strings. Considerations similar to tri-grams are therefore expected to hold for twigs. For instance, an advantage of the use of twigs over equality-matching is their robustness for 'spelling variations'. For example, if to index expressions are equal modulo order, then their twigs are the same, as stated in the following lemma.

Lemma 5.1. $\mathtt{EqOrder}(I, J) \Rightarrow \mathsf{twigs}(I, 1) = \mathsf{twigs}(J, 1)$.

Set-based approaches, described for terms in Section 5.1, can also be equipped with twigs. Instead of calling the set based approaches with sets of terms or connectors as arguments, the (sets of) twigs are used. This yields similarity measures using twigs. As an example, the Dice measure for twigs is defined as

$$\mathsf{Dice\text{-}twigs}(I, J) = \mathsf{Dice}(\mathsf{twigs}(I, 1), \mathsf{twigs}(J, 1))$$

The Dice measure for twigs is maximal for index expressions that are equal modulo order and for identical index expressions, since their twigs are equal.

The Dice measure for twigs is not maximal for embedded index expressions nor or subexpressions. This is illustrated by, for instance, index expressions surfing in Holland and Holland.

6. Experimental evaluation

Subexpressions are the basic notion on which the hyperindex is generated. The INN system thus heavily uses subexpressions. We therefore evaluate how the different similarity measures behave in the context of subexpressions. A number of experiments, described in this section, give insight in this matter.

The experiments exploit a non-trivial index expression (see figure 7(a)) consisting of 11 terms nested in subexpressions of different forms. The similarity between this index expression and all its subexpressions was computed, according to the different similarity measures. Both contiguous and non-contiguous (e.g. workshop during November) subexpressions were generated from the original. In total, the experiment featured 439 such

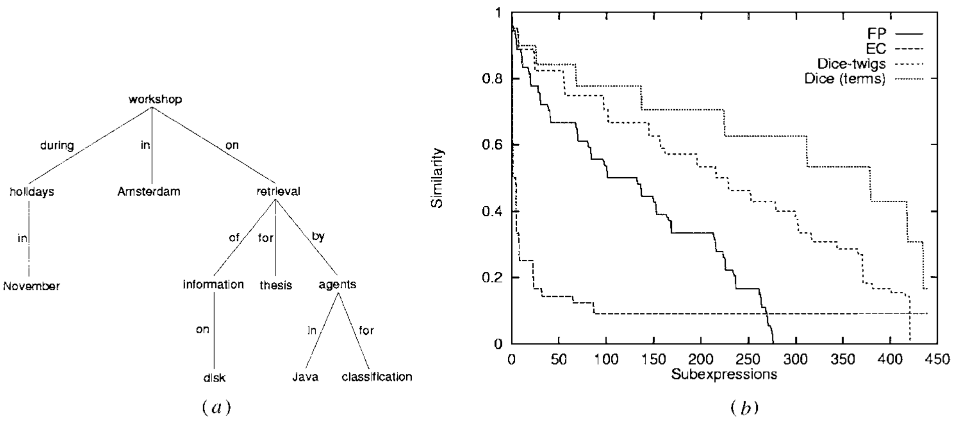

Figure 7. (a) Index expression used for experiments. (b) Overview of similarity measures.

subexpressions. Similar tests may be performed with other notions of subexpressions. For instance, the order of subexpressions may be varied.

Figure 7(b) gives an overview of the different similarity measures. The graphs provided in this figure serve as reference for the next sections. Not all set-based measures are provided; the Dice measure (for terms and twigs) is given as representative. For several reasons, the subexpressions on the x-axis are sorted according to their similarities such that non-increasing graphs are obtained. Sorting similarities non-increasingly and therefore grouping together subexpressions that are equally similar to the original index expression in intervals, clearly shows the results of the similarity measures. For instance, without the sorting, different measures in the same graph would be hard to distinguish. Furthermore, since subexpressions within the same interval have the same similarity value, intervals rather than the positions on the x-axis should serve as indication of the ordering. For instance, the number of intervals gives an indication of the granularity of the measure. We therefore compare similarity measures by examining differences between their intervals. The order of subexpressions on the x-axis may be different for different similarity measures since we sort non-increasingly per measure. The different orders per measure are not explicitly visualised in the graphs. Therefore, our examination used detailed information, as kept in the test results, next to the graphs. The differences between the measures are examined in the rest of this section.

In our experiments, we examine differences between similarity measures by imposing the order of one similarity measure (the so called dominant measure) onto other similarity measures (the subordinate measures). That is, the order of subexpressions on the x-axis is set to conform to the non increasing order of the dominant measure. According to this ordering, the subordinate measures are plotted. In this way, the differences in ordering are visualised.

Figure 8 gives a generic sketch of the graphs that resulted from the experiments. In this figure, the solid line depicts the similarity values of the dominant measure. In later figures, where actual similarity measures are used, the line of the dominant measure equals

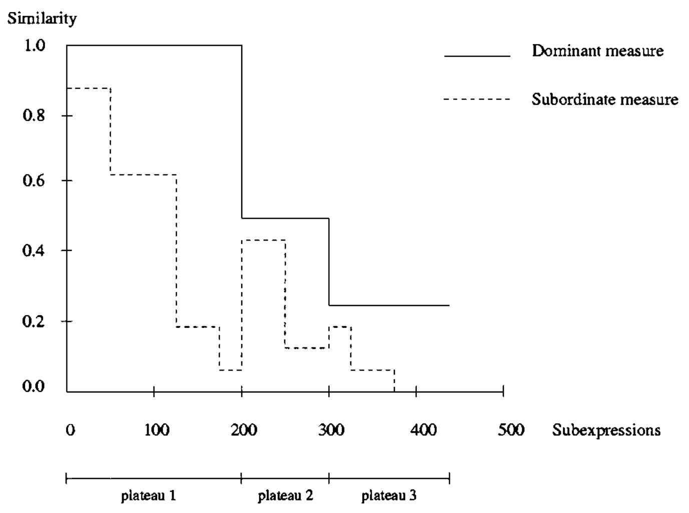

Figure 8. Generic sketch of results.

its similarity graph of figure 7(b): the subexpressions on the x-axis are sorted such that a non-increasing line for the dominant measure is obtained. According to this dominant ordering, the subordinate measures are plotted. Note that the example dominant measure has three horizontal plateaus, i.e. intervals of subexpressions that all receive the same similarity value, or, which are not discriminated by the dominant measure. For subexpressions out of the same plateau, the differences in positions on the x-axis are thus irrelevant to the dominant measure. However, a subordinate measure may assign different similarity values to subexpressions out of the same dominant plateau. Therefore, we re-sort subexpressions per dominant plateau according to subordinate measures. In this way, we obtain non-increasing graphs for the subordinate measures per dominant plateau. Note that the number of subordinate plateaus may differ per dominant plateau. However, this number at most equals the number of plateaus in the original graph of the subordinate measure of figure 7(b). As a result of sorting non-increasingly, all peaks in subordinate graphs occur at the beginning of dominant plateaus. The form of actual graphs provided later conforms to the generic sketch of figure 8 but has much finer granularity because of more and smaller plateaus.

We first compare the measures that use both content and structure: the full product, embedded content, and set based measure for twigs. In Sections 6.1 and 6.2, respectively, the full product and embedded content are used as primary measure. Set-based measures for twigs are compared in Section 6.3.

6.1. *Full product*

The results of imposing the full product ordering onto the embedded content and Dice's measure for twigs are shown in figure 9. The three measures define different orderings, as can be concluded from the many peaks (embedded content) and drops (Dice-twigs) with respect to their original lines in figure 7(b).

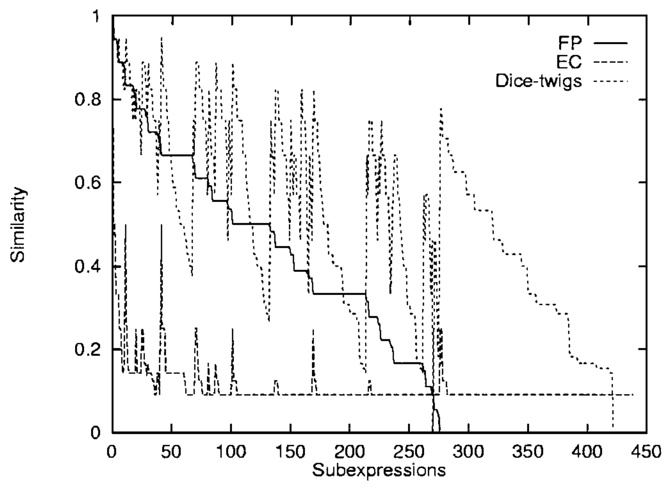

Figure 9. Full product order imposed.

The ordering imposed by full product is related to the removal of leaves starting at maximal depth, or, a layer by layer (bottom up) defoliation. Thus, the full product measure scores high if the heads are (recursively) equal. In other words, a subexpression that equals a topmost (layer by layer) part of the original gets a high score. Full product thus demands the main concepts to be equal and is less sensitive for modification of the most specific (maximal depth) modifiers. The gradually decreasing graph of the full product measure combined with its high granularity implies that this measure delicately distinguishes most subexpressions.

The rightmost interval of full product is a large zero-plateau. In this large interval, the subordinate measures resemble their original graphs of figure 7(b). Inspection of the test results showed that the subexpressions in this plateau, compared to the original, either have a different head or contain a single subexpression that has a different head. For comparing index expressions that represent different main concepts, the full product measure is thus inadequate.

In general, each full product interval shows a peak for embedded content and a drop for the Dice measure for twigs. By examining which subexpressions cause these drops and peaks, the differences between the measures are studied.

Basically, two types of subexpressions cause the peaks for the embedded content measure. Of both types, a stereotypical example is investigated. Consider, for instance, the peak in figure 9 that occurs at position 43, having similarity value 0.5. The subexpression causing this peak consists of 10 terms, missing only the depth two leaf Amsterdam. However, full product favours subexpressions which are obtained by removing leaves at larger depth. The many possibilities to construct subexpressions in this way makes that full product has many higher scoring subexpressions. As concluded from analysing the test results, embedded content would have placed it in the interval at positions 2–5 of figure 7(b). The removal of the highest leaf (i.e. Amsterdam) is, to embedded content, similar to removing a leaf at largest depth, such as disk. Similar subexpressions cause the peaks at positions 17, 80, 101, and 171. Thus, when the influence of concept modification is desired to gradually decrease with depth, full product is a better measure than embedded content.

Another type of subexpression causes the peak at position 277, which is visually somewhat obscured by a simultaneous drop in the graph of the Dice-twig measure. This peak of similarity value 0.25 is surrounded by values of 0.09. It is caused by an index expression that does not have workshop as head, but has a large embedded subexpression consisting of 7 terms. Since the main concepts are different, full product scores it low. However, embedded content scores it higher and resorts it to the beginning of the value zero interval. Embedded content, by its own order, would have placed it in the interval at positions 2–5. Thus, full product, more strongly than embedded content, targets at equal main concepts.

The drop in Dice's graph at position 68 to similarity value 0.375 is caused by a non-contiguous subexpression, five of whose six twigs do occur in the original. Full product, by its layer by layer computation, does not favour non-contiguous (layer skipping) subexpressions. A similar argument applies to the drops at positions 133, 214 and 263. Twig based measures thus are more robust for slightly differing non-contiguous subexpressions than full product.

Dice's drop to the x-axis at position 271 is caused by the single term workshop which consists of the head of the original index expression only. As stated before, single terms have no twigs and Dice's twig measure thus returns value zero.

6.2. *Embedded content*

The second experiment imposes the embedded content order onto the full product and Dice measure for twigs. The results are given in figure 10. Consider the line of the embedded content measure. A striking feature is the large rightmost plateau, stemming from the following.

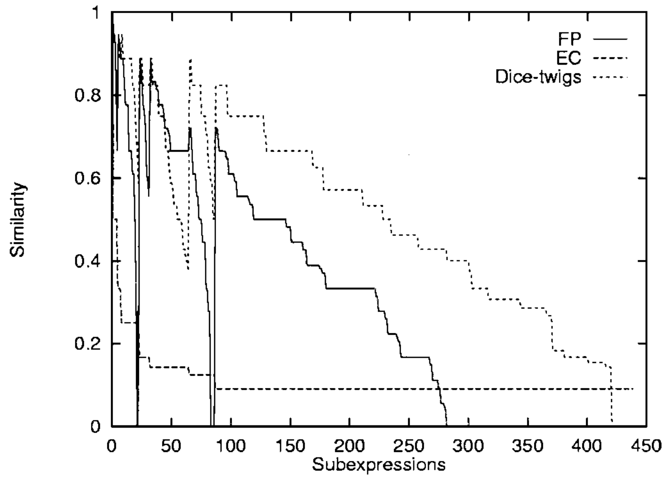

Figure 10. Embedded content order imposed.

We computed the embedment of the original index expression in its subexpressions, not the other way around since this would result in maximal embedment for all subexpressions. Thus, the embedding of the original is computed in (the components of) the subexpressions. For many subexpressions, this eventually results in the embedding of the original in the single terms of the subexpression. This, in turn, results in a constant value which is inversely proportional to the size of the original. As a consequence, embedded content is in this context best used for discriminating between highly similar subexpressions.

In general, the order imposed by embedded content is relative to the removal of terms, irrespective of their depth. For instance, removing Amsterdam or disk results in equally similar subexpressions. Thus, embedded content makes no distinction to the specificity (depth) of the removed term. As a consequence, it is not so sensitive for the removal or addition of a complete modification (subexpression). This makes the embedded content measure suitable for matching queries with rich document characterisations.

After position 88, the graphs of full product and Dice-twigs seem to equal their original graphs of figure 7(b). This is caused by the large dominant plateau, starting at that position, which, by resorting the subordinate measures, ensures that all the subexpressions are in the same order as in their original graph. Differences with their original graphs are caused by subexpressions that are placed in the last interval by the dominant measure but not by the original order of the subordinate measure, or vice versa.

A number of stepwise drops occur in the line of full product in figure 10. For instance, full product drops to zero at positions 22, 23, and 84–87. This is caused by subexpressions with different heads. As was also observed for the first experiment, embedded content is more robust to changes in the main concept. The downward steps towards value zero are caused by subexpressions that miss several terms, but preserve other parts of subexpressions.

Dice's twig measure shows a clear drop at position 65, which is caused by a (rather small) non-contiguous subexpression which has only a few twigs in common with the original. However, embedded content, being more robust to non-contiguous subexpressions, ranks it relatively high. A similar argument holds for the drops at positions 22 and 87. Thus, highly non-contiguous subexpressions are still recognised as relevant by embedded content.

6.3. Set-based twig similarity

In the final experiment, we compared set-based approaches for terms with those for twigs. In particular, the Dice measure for terms is used as baseline for comparison with the Dice, Jaccard, and Cosine measures for twigs. Figure 11 provides an overview of the used set-based measures, each measure being non increasingly ordered as was done for the other measures in figure 7(b).

In our experiments, twig based measures have finer granularity than term based measures. This is concluded from the number of intervals imposed by these measures. The set based measures divide the subexpressions in classes (intervals) corresponding to the number of constituents in common with the original. As can be seen from figure 11, the Dice measure for terms has 10 intervals for $\alpha = 1.0$ (i.e. no connectors considered) and 20 for $\alpha = 0.5$. The measures for twigs each have 26 intervals. This number directly depends on the number of different twigs that can be generated from the subexpressions. Since we included

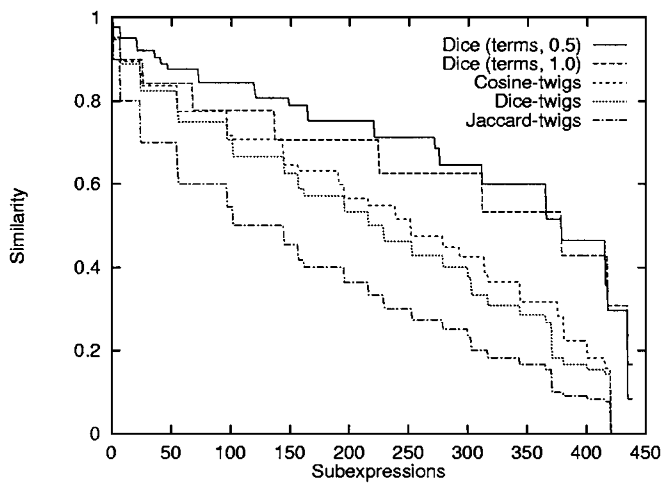

Figure 11. Set-based approaches for terms and twigs.

non-contiguous subexpressions in our experiment, new twigs are introduced that are not in the original. The original index expression has 10 twigs thus leaving room for 16 newly introduced twigs. This suggests that twig based measures should be favoured over term based measures if delicate matching is required.

The lines for the twig-based measures eventually drop to zero whereas the term-based measure does not. The reason for this is that single terms do not result in twigs although every (non-empty) subexpression has a minimal overlap of at least one term with the original. Clearly, twig based measures should thus not be used for matching single terms. The minimal similarity value according to Dice for terms (for $\alpha = 1.0$) is $\frac{2\times 1}{11+1} = 0.167$.

In order to further compare the set-based approaches, the Jaccard ordering for twigs was imposed on the others. Figure 12 shows the results. Dice and Jaccard for twigs impose the same ordering: they have exactly the same intervals. As can thus be concluded from the definitions of these measures (see Section 5.1), in comparing two subexpressions, no distinction (in terms of the ordering) is made between the sum of twigs and the number of different twigs. Their similarity values, however, differ in size. This basically means that, practically, the differences between both measures can be overcome by using rightly set thresholds. The Cosine measure for twigs has closely resembling intervals: of its 26 intervals, only six differ slightly (maximally nine subexpressions).

There is a big difference between set-based approaches for terms and those for twigs, since terms do not capture structure. The peaks in the line of Dice's term measure (figure 12) are caused by non-contiguous subexpressions, not having many twigs in common. Irrespective of their structure, however, they share terms with the original. Clearly, when the structure of descriptors is to be taken into account, twig based measures are more suitable than term based measures.

The experiments illustrated characteristics of the different similarity measures. A number of conclusions can be drawn. First, approaches only using terms should be avoided if

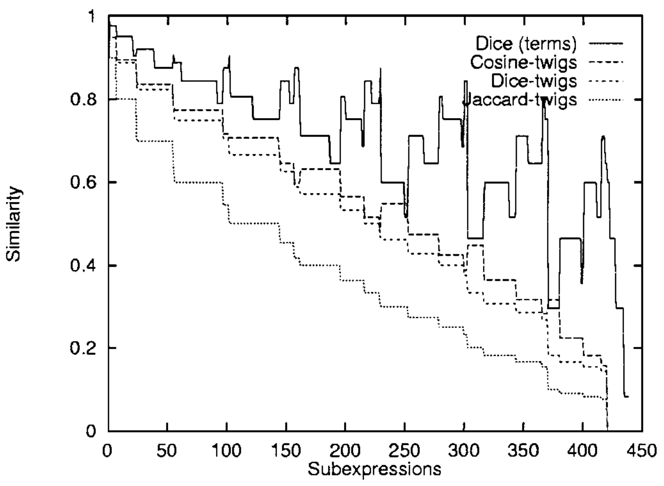

Figure 12. Jaccard (twigs) order imposed.

the structure of descriptors is relevant. This is what one would expect, as well as that set based measures using terms provide less granularity than those using twigs. Second, the different criteria about subexpressions resulted in similarity measures with distinct properties. The layer by layer computation of the full product measure, derived from the idea of headedness, favours descriptors that only differ in their most specific subexpressions. However inadequate for comparing index expressions that represent different main concepts, it naturally supports delicate discrimination of most subexpressions and a gradual notion of concept modification. The embedded content measure is not sensitive to the depth of the deleted subexpression. Furthermore, since it is based on embedding, it is more robust to non-contiguous subexpressions. Embedded content is, in the context of our experiments, best used for discriminating highly similar subexpressions. Twig based measures fall somewhere in between. They are somewhat sensitive to non-contiguous subexpressions: this introduces new twigs, but may preserve other twigs. There is little difference between twig based measures themselves. It appears mainly a matter of thresholds, viz size of the similarity values, to distinguish between them.

7. Conclusions

In this article, we devised several similarity measures for index expressions aiming at use in a dynamic hypertext system, the INN system. This envisaged application implied our focus on different notions of subexpressions for index expressions and several constraints concerning high efficiency, domain independency, and restricted memory. These constraints prohibit the use of sophisticated natural language processing techniques. However, several issues on the semantics of index expressions and corresponding notions of subexpressions were formulated in criteria. These issues were the order of subexpressions, non contiguous embedding, and headedness. The devised similarity measures were both theoretically and experimentally evaluated showing their suitability for the mentioned criteria. Although the experiments described in this article are rather small, they illustrate some of the essential characteristics of the similarity values.

Future research can be directed towards augmenting index expressions with wildcards and the design of corresponding matching functions. This could aim at using the structure of index expressions for fact finding and the generation of partly specified overviews. In the augmented language, descriptors like flying from Tasmania to * can be used to find all documents that deal with air transport from the island of Tasmania to any destination.

Furthermore, it would be interesting to see if our approach is also viable for other retrieval related tasks such as clustering, routing, and filtering. This would enable an augmentation of the functionality of the INN system. For instance, the INN system could then present grouped results or it could work off-line by using stable information needs in the case of filtering.

It is also interesting to investigate different instantiations of the abstract term and connector comparison functions. For instance, stemming could be included by computing the similarity of stems. In addition, connectors can be grouped by their function type. In matching, connectors with the same function could be given higher similarity.

Including term weights potentially increases the effectiveness of matching. We did not focus on this issue here since it is not primarily concerned with the structure of index expressions. However, it seems a valuable extension worth further research.

Acknowledgment

The authors kindly acknowledge the valuable remarks of the referees.

References

Arampatzis AT, Tsoris T, Koster CHA and van der Weide ThP (1998) Phrase-based information retrieval. Information Processing & Management, 34(6):693–707.

Berger FC (1998) Navigational query construction in a hypertext environment. PhD Thesis, Department of Computer Science, University of Nijmegen.

Brill E (1994) Some advances in rule-based part of speech tagging. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), Seattle, Wa.

Bruza PD (1993) Stratified information disclosure: A synthesis between information retrieval and hypermedia. PhD Thesis, University of Nijmegen, Nijmegen, The Netherlands.

Bruza PD and van der Weide ThP (1992) Stratified hypermedia structures for information disclosure. The Computer Journal, 35(3):208–220.

Evans DA, Ginther-Webster K, Hart M, Lefferts RG and Monarch I (1991) Automatic indexing using selective NLP and first-order thesauri. In: Lichnerowicz A, Ed., Proceedings of RIAO'91, Barcelona, Spain, pp. 624–643.

Evans D, Lefferts R, Grefenstette G, Handerson S, Hersch W and Archbold S (1992) Clarit trec design, experiments, and results. In: Harman DK, Ed., Proceedings of TREC-1, Gaithersburg, MD, US, pp. 251–286.

Farradane J (1980a) Relational indexing part I. Journal of Information Science, 1(5):267–276.

Farradane J (1980b) Relational indexing part II. Journal of Information Science, 1(6):313–324.

Iannella R, Ward N, Wood A, Sue H and Bruza P (1995) The open information locator project. Technical Report, Resource Discovery Unit, Cooperative Research Centre, University of Queensland, Brisbane, Australia.

Kilpelaïnen P and Mannila H (1993) Retrieval from hierarchical texts by partial patterns. In: Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Pittsburgh, PA, USA, pp. 214–222.

Lucarella D and Zanzi Z (1993) Information retrieval from hypertext: An approach using plausible inference. Information Processing & Management, 29(3):299–312.

Mauldin ML (1989) Retrieval performance in FERRET. In: Proceedings of the ACM SIGIR Conference, pp. 347–355.

Metzler DP and Haas SW (1989) The constituent object parser: Syntactic structure for information retrieval. ACM Transactions of Information Systems, 7(3):292–316.

Salton G and Smith M (1989) On the application of syntactic methodologies in automatic text indexing. In: Proceedings of the ACM SIGIR Conference, pp. 137–150.

Smeaton AF and Sheridan P (1991) Using morpho-syntactic language analysis in phrase matching. In: Lichnerowicz A, Ed., Proceedings of RIAO'91, Barcelona, Spain, pp. 414–430.

Sparck Jones K and Tait JI (1984) Automatic search term variant generation. Journal of Documentation, 40(1):50–66.

Strzalkowski T (1995) Natural language information retrieval. Information Processing & Management, 31(3):397–417.

van Rijsbergen CJ (1975) Information Retrieval. Butterworths, London, United Kingdom.

van der Vet P and Mars NJI (1998) Bottom-up construction of ontologies. IEEE Transactions on Knowledge and Data Engineering, 10(4):513–526.

Wilkinson R and Fuller M (1996) Integrated information access via structure. In: Agosti M and Smeaton A, Eds., Hypertext and Information Retrieval, Kluwer, Boston, U.S.A., pp. 257–271.

Wondergem BCM, van Bommel P and van der Weide ThP (2000) Nesting and defoliation of index expressions for information retrieval. Knowledge and Information Systems, 2(1).

Wondergem BCM, van Uden M, van Bommel P and van der Wei de ThP (1999) INdex navigator for searching and exploring the WWW. Technical Report CSI-R9917, University of Nijmegen, Nijmegen, The Netherlands.