Matching Index Expressions for Information Retrieval

B.C.M Wondergem , P. van Bommel, Th.P. van der Weide

Computing Science Institute/

**CSI-R9826 December 1998**

Computing Science Institute Nijmegen
Faculty of Mathematics and Informatics
Catholic University of Nijmegen
Toernooiveld 1
6525 ED  Nijmegen
The Netherlands

# Matching Index Expressions for Information Retrieval

B.C.M. Wondergem, P. van Bommel, and Th.P. van der Weide
Computing Science Institute, University of Nijmegen
Toernooiveld 1, NL-6525 ED, Nijmegen
The Netherlands
(bernd@cs.kun.nl)

**Keywords:** Information Retrieval, Similarity, Index Expressions, Matching.

**Abstract**

Index expressions are expressive descriptors that capture some of the linguistic structure of natural language. Index expressions have proven of value for Information Retrieval (IR) in many applications. However, quantitative matching schemes, or similarity measures, have not been provided for index expressions yet. In this article, several quantitative matching strategies for index expressions are devised. This vastly increases the potential of applying index expressions for IR. Both content and structure play a role in these matching strategies. Different views on the semantics of index expressions are analysed: order of subexpressions, embedding, and headedness. These views are formalised in criteria for which corresponding similarity measures are devised. These measures are proven optimal for the corresponding criteria. In designing the similarity measures, both the inductive and the structural representations of index expressions are exploited.

# 1 Introduction

Index expressions (see e.g. [8, 17, 4]) have proven of great value for Information Retrieval (IR). As expressive, yet simple descriptors, they have been applied for query formulation ([3]), document indexing ([7]), and the construction of hyperindices ([6]) and association indices (see e.g. [16, 15]). Index expressions exhibit a nice structure which captures some of the linguistic structure of natural languages. Index expressions also provide a useful simplification of noun-phrases.

However, matching cannot yet be done properly with index expressions. The reason for this is the lack of numerical similarity measures. In this article, several such similarity measures for index expressions are devised, thus enabling index expressions to be used in important IR tasks that involve matching.

Matching is one of the three main concepts in IR next to query formulation and document characterisation. At a conceptual level, matching measures the strength of resemblance between descriptors. Matching is used for query processing, classification, clustering, filtering, routing, etc.

The resemblance between index expressions can be expressed qualitatively and quantitatively. The qualitative approach deals with equality, equivalence, and notions of subexpressions. In [17], a formalisation of index expressions and several qualitative relations describing subexpressions are given. For the mentioned IR tasks that involve matching, qualitative approaches are not sufficiently expressive. In stead of qualitative relations, quantitative (numerical) functions are needed.

The only currently existing way of matching index expressions numerically is with the use of belief networks ([5]). However, one needs expert knowledge about such networks to set up a system with it. Furthermore, the belief network approach seems hard to augment with additional linguistic knowledge. Another major drawback is that it cannot be based on different views on the semantics of index expressions.

Clearly, matching is to establish the resemblance in meaning of index expressions. In other words, matching should deliver the degree of semantical resemblance. We therefore first analyse important issues concerning the semantics of index expressions and, directly based on that, devise several corresponding matching functions.

In this article, we do not consider the use of additional linguistic knowledge. For example, we do not consider stemming and the use of thesauri. However, since these issues are largely orthogonal to our approach, they can readily be incorporated.

Related work is done in the areas of fuzzy matching using WordNet ([12]), where descriptors are flattened before being matched and tree inclusion algorithms ([11]), concentrating on the algorithmical aspects of tree embedding.

In the next section, preliminaries of index expressions are given. In section 3, several issues concerning the semantics of index expressions are analysed. In section 4, matching strategies for index expressions corresponding to the result of section 3 are introduced. Section 5 elaborates on related work. Finally, section 6 provides concluding remarks and directions for further research.

# 2 Index Expressions

Index expressions can be represented in several ways (see e.g. [17]). 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.

## 2.1 Inductive Representation

The inductive representation, as described in definition 2.1, 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. This is illustrated in figure 1. An advantage of such an elementary representation is that it allows several auxiliary functions to be readily designed.

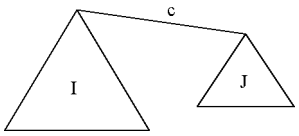


Figure 1: Basic Setup of Inductive Representation

**Definition 2.1**

*Let $T$ be a nonempty set of terms and $C$ be a set of connectors such that $T \cap C = \varnothing$. 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* $\mathsf{add}(I, c, J)$ *is also a non-empty index expression, and*

3. *no other non-empty index expressions exist.*

□

**Example 2.1**

*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* $\mathsf{add}(\mathsf{conference}, \mathsf{on}, \mathsf{biology})$ *represents information on a conference on biology. As a more complex example, consider* $\mathsf{add}(\mathsf{add}(\mathsf{conference}, \mathsf{on}, \mathsf{biology}), \mathsf{in}, \mathsf{Holland})$ *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* $\mathsf{add}(\mathsf{conference}, \mathsf{on}, \mathsf{add}(\mathsf{biology}, \mathsf{in}, \mathsf{Holland}))$. *The last one denotes a conference about biology for as far as it is practiced in Holland.* □

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

Figure 2: Auxiliary functions on index expressions.

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.

### 2.2 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

$$h \otimes_{i=1}^{k} c_i(I_i) \quad \equiv \quad hc_1(I_1) \ldots c_k(I_k)$$

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.

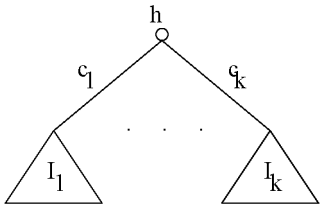


Figure 3: Structural Representation

Figure 3 gives a schematic view on the structural representation.

**Example 2.2**

*The two last mentioned index expressions of example 2.1 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.* □

## 3 Issues concerning Semantics

The index expression

$$hc_1(I_1) \ldots c_k(I_k)$$

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

## 3.1 Contents

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

### 3.1.1 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. Furthermore, additional lexico-semantical knowledge can be used taking, for instance, hypernyms and synonyms into account. In IR, stemming is often performed to identify equivalent terms. We abstract from the particular techniques used in computing the similarity and concentrate on using this in computing similarity between index expressions.

### 3.1.2 Connectors

Similar to terms, we also assume a similarity function $\mathsf{sim}_C : C \times C \rightarrow [0..1]$ 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 [9] and [10]). 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.

## 3.2 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*.

### 3.2.1 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 3.1**
*We call two index expressions* $I = h \otimes_{i=1}^{k} c_i(I_i)$ *and* $J = h' \otimes_{j=1}^{k} d_i(J_j)$ equivalent modulo order, *denoted by* $\texttt{EqOrder}(I, J)$, *iff*

1. $h = h'$, *and*
2. *there exists a permutation* $\pi$ *of* $[1..k]$ *such that for all* $1 \leq j \leq k$ *it holds that* $c_i = d_{\pi(j)}$ *and* $\texttt{EqOrder}(I_i, J_{\pi(j)})$.

□

Two index expressions are equal modulo the order of their subexpressions iff their heads are equal and (recursively) their subexpressions can be grouped in pairs (by the permutation) that are also equal modulo order.

**Lemma 3.1** For every non-empty index expression $I$, we have $\texttt{EqOrder}(I, I)$.

**Proof:**
This follows from the observation that identity is to be taken as permutation.

Later in this article, we describe a matching strategy that is optimal for index expressions that are equal modulo order.

### 3.2.2 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.

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. In stead 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 3.2**

*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 3.2. 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 $J$ and $L$.

**Lemma 3.2** For every non-empty index expression $I$, we have $I \ll I$.

The subexpression relation for index expressions (see [17]), denoted by $\preccurlyeq$, defines the (connected) subexpressions of index expressions. The expression $I \preccurlyeq J$ means that index expression $I$ is a subexpression of $J$.

**Lemma 3.3** For every two non-empty index expressions $I$ and $J$, we have $I \preccurlyeq J \Rightarrow I \ll J$.

**Proof:** The lemma follows from the observation that in the subexpression relation $\preccurlyeq$, direct ancestorship is preserved, which is stricter than the general ancestorship which is preserved by the embedment relation $\ll$.

### 3.2.3 Headedness

The head of an index expression is considered to be its most important part. This concept is called *headedness* in [13]. 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 with the depth gives the desired result. We will indicate how the designed similarity functions can be equipped with such a depth factor.

# 4 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 to more comprehensive measures that consider both content and structure.

## 4.1 Contents Only

Measuring the similarity between index expressions by sets of terms and connectors considers only their contents. That is, these similarity measures are based solely on the terms and connectors of index expressions. This means that the structure of index expressions is not taken into account.

The similarity between sets of elements can be expressed by several standard measures. Examples are the Dice and Jaccard coefficients and the cosine and overlap measures (see e.g. [14]). These measures do not use the primitive similarity functions for terms and connectors as described in sections 3.1.1 and 3.1.2, respectively. Instead, they use set primitives as union,

disjunction, and cardinality which use equality of elements (terms and connectors).

As an example, we apply the Dice coefficient for matching index expressions. The Dice coefficient, which normalizes the intersection $A \cap B$ with the sum of constituents, is defined as

$$\mathsf{Dice}(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

in case $A$ or $B$ is nonempty. Furthermore, $\mathsf{Dice}(\varnothing, \varnothing) = 0$. As an example, consider the following similarity measure for index expressions based on the Dice coefficient:

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

Figure 4: Dice's Similarity Measure for Index Expressions

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.

## 4.2 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. Finally, the *Twigs* measure is based on decomposing index expressions into elementary connections called twigs.

### 4.2.1 Full Product

The *Full Product* similarity measure computes the degree to which an index expression is equivalent 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 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

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

Figure 5: Full Product Algorithm

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 composed index expressions. Correspondingly to definition 3.1, 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 fully embedded in 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. We say that a similarity measure is *optimal* for certain combinations of index expressions if it returns similarity value 1.

**Theorem 4.1** The Full-Product similarity measure is optimal for index expressions that are equal modulo the order of their subexpressions.

**Proof:**
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 $\mathtt{max}\{\textsf{sim}(hc_i(I_i), h'd_j(J_j)) | 1 \leq j \leq l\}$ is 1 since

1. $h = h'$, meaning $\mathsf{sim}_T(h, h') = 1$, and
2. there exists a $1 \leq j \leq l$ such that $d_j = c_i$ and $\texttt{EqOrder}(I_i, J_j)$ meaning that $\mathsf{sim}_C(c_i, d_j) = 1$ and, as can be shown by induction, $\mathsf{sim}(I_i, J_j) = 1$.

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

**Corollary 4.1** The Full Product similarity measure computes $\mathsf{sim}(I, I) = 1$.

The depth factor $\mathtt{d}$ can be incorporated in the Full Product measure by altering case Tall of the Full Product algorithm of figure 5:

$$\begin{array}{ll} & \mathsf{sim}_{\mathtt{d}}(h \otimes_{i=1}^{k} c_i(I_i), h' \otimes_{j=1}^{l} d_j(J_j)) \\ = & \mathsf{sim}_{\mathtt{d},T}(h, h') \times \frac{1}{k}\Sigma_{i=1}^{k}\{\mathsf{sim}_{\mathtt{d+1},C}(c_i, d_j) \times \mathsf{sim}_{\mathtt{d+1}}(I_i, J_j) | 1 \leq j \leq l\} \end{array}$$

### 4.2.2 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 3.2) in another one. This means that the order of subexpressions is considered relevant.

$$\begin{array}{lll} (\mathsf{Terms}) & & \mathsf{sim}(t, t') \\ & = & \mathsf{sim}_T(t, t') \\ \\ (\mathsf{Top}) & & \mathsf{sim}(t, \mathsf{add}(I, c, J)) \\ & = & \mathtt{max}\{\mathsf{sim}(t, I), \mathsf{sim}(t, J)\} \\ \\ (\mathsf{Tall}) & & \mathsf{sim}(\mathsf{add}(I, c, J), \mathsf{add}(K, d, L)) \\ & = & \mathtt{max}\{\mathsf{sim}(\mathsf{add}(I, c, J), K),\ \mathsf{sim}(\mathsf{add}(I, c, J), L), \\ & & \quad \mathsf{sim}(I, K) \times \mathsf{sim}_C(c, d) \times \mathsf{sim}(J, L)\} \\ \\ (\mathsf{Toll}) & & \mathsf{sim}(\mathsf{add}(I, c, J), t) \\ & = & \frac{\mathsf{sim}_T(\mathsf{Head}(I), t)}{|\mathsf{Terms}(\mathsf{add}(I, c, J))|} \end{array}$$

Figure 6: Embedded Content Algorithm

The cases considered by the embedded content measure are the same as for the Full Product measure. The case Terms is exactly the same and calls the similarity function for terms $\mathsf{sim}_T$. The second case Top of figure 6 states that the strength at which a term $t$ is embedded in a composed index expression $\mathsf{add}(I, c, J)$ is equal to the maximal similarity to one of $\mathsf{add}(I, c, J)$. 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. Again, case Toll gives a penalty for mismatch since terms can never be embedded in composed index expressions.

Note that the Embedded Content measure never results in a higher value than the basic skeleton measure since extra constraints are implied by the content.

**Theorem 4.2** The Embedded Content measure is optimal for embedded index expressions.

**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 3.2 of embedded content, Same and Sub, are to be examined since the third case Split cannot apply ($I$ is a term).

**Same:** Suppose this case of definition 3.2 applies. Then, $J = I = t$ and consequently $\mathsf{sim}(I, J) = \mathsf{sim}_T(I, J) = 1$.

**Sub:** Suppose this case of applies which means that $J = \mathsf{add}(K, c, L)$ and $t \ll K$ or $t \ll L$. Similar induction to the structure of $J$ shows that either $\mathsf{sim}(t, K) = 1$ or $\mathsf{sim}(t, L) = 1$. This means that, by case Top of the embedded content algorithm, $\mathsf{sim}(I, J) = 1$.

**Induction step.** Suppose $I$ is a composed index expression $\mathsf{add}(K, c, L)$. Furthermore, without loss of generality, let $J = \mathsf{add}(M, d, N)$. Now, only cases Sub and Split of definition 3.2 need to be examined. Both result in $\mathsf{sim}(I, J) = 1$:

**Sub:** Here, we have (1) $I \ll M$ implying $\mathsf{sim}(I, M) = 1$ or (2) $I \ll N$ which means that $\mathsf{sim}(I, N) = 1$. Since case Tall of

the algorithm is applied, computing the maximum of these cases, both options cause $\mathsf{sim}(I, J) = 1$.

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

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

**Corollary 4.2**

1. The embedded content measure is optimal for subexpressions, i.e., $I \preccurlyeq J \Rightarrow \mathsf{sim}(I, J) = 1$
2. $\mathsf{sim}(I, I) = 1$ for all index expressions $I$

The depth factor d can be incorporated in the inductive representation by the following scheme:

$$\mathsf{sim_d}(\mathsf{add}(I, c, J), \mathsf{add}(K, d, L)) \;=\; \mathsf{sim_d}(I, K) \times \mathsf{sim}_C(c, d) \times \mathsf{sim_{d+1}}(J, L)$$

Figure 7: Depth-factor

### 4.2.3 Twigs

The *twigs* (see [18] or [1]) 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 enable us to form a global picture about similarity while focusing in on the elementary refinements.

Our contributions to the twigs measure are (1) the inclusion of headedness, (2) an functional algorithm to compute twigs for inductively defined index expressions, and (3) an implementation of the measure in the functional language Clean (see e.g. [2]).

To denote subexpressions of size two, we use the subexpression relation as given in the previous section.

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

In terms of the inductive representation of index expressions, twigs can be defined constructively as follows. This shows that the twigs of an index expression can be produced by a straightforward syntactic process. Note that twigs are accompanied by their depth factor, modeled by a positive integer. The expression $\mathsf{twigs}(I, 1)$ computes the twigs of $I$ and their depth.

$$\begin{array}{lcl} \mathsf{twigs}(\epsilon, d) & = & \varnothing \\ \mathsf{twigs}(t, d) & = & \varnothing \\ \mathsf{twigs}(\mathsf{add}(I, c, J), d) & = & \{(\mathsf{add}(\mathsf{Head}(I), c, \mathsf{Head}(J)), d)\} \\ & & \cup\, \mathsf{twigs}(I, d) \cup \mathsf{twigs}(J, d+1) \end{array}$$

**Lemma 4.1** If to index expressions are equivalent modulo order, then their twigs are the same, i.e., if $\mathtt{EqOrder}(I, J)$ then $\mathsf{twigs}(I, 1) = \mathsf{twigs}(J, 1)$.

The similarity between two twigs is the product of the similarity between the two connectors and the average similarity between the left- and rightmost terms. In addition, the depth factors are taken into account by function $f : \mathcal{N} \times \mathcal{N} \to [0..1]$.

$$\mathsf{twigsim}((tct', k), (udu', k')) = f(k, k') \times \mathsf{sim}_C(c, d) \times \frac{\mathsf{sim}_T(t, u) + \mathsf{sim}_T(t', u')}{2}$$

The *twig measure* is defined as the normalized sum of similarity between the twigs of both index expressions.

$$\mathsf{sim}_{\mathsf{twigs}}(I, J) \;=\; \mathsf{avg} \sum_{p \in \mathsf{twigs}(I)} \sum_{q \in \mathsf{twigs}(J)} \mathsf{twigsim}(p, q)$$

Figure 8: Twigs similarity

**Corollary 4.3** If index expressions $I$ and $J$ are equivalent modulo order, then for every index expression $K$ we have $\mathsf{sim}_{\mathsf{twigs}}(I, K) = \mathsf{sim}_{\mathsf{twigs}}(J, K)$.

As observed in [1], 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 should be different at the same depth.

Twigs of index expressions seem to resemble trigrams for strings. Considerations similar to trigrams 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' such as variations in order of subexpressions.

### 4.3 Overview of Properties

Figure 9 gives an overview of the optimality of the different similarity measures with respect to several criteria. The columns correspond to Dice's measure, the full product, embedded content, and the twigs similarity measure, respectively. The rows correspond to the notion of equality modulo the order of subexpressions (`EqOrder`), embedding ($\ll$), identical arguments ($\mathsf{sim}(I, I)$), and subexpressions ($\preccurlyeq$). A + denotes that the measure is optimal for the criterion.

| Optimal | Dice | FullProd | EmbCont | Twigs |
|---|---|---|---|---|
| `EqOrder` | + | + | - | - |
| $\ll$ | - | - | + | - |
| $\mathsf{sim}(I, I)$ | + | + | + | - |
| $\preccurlyeq$ | - | - | + | - |

Figure 9: Overview of optimality.

The optimal cases were proven earlier in this section. Below, illustrative counterexamples are given for the non optimal cases. The contents of figure 9 is discussed row by row.

First, consider 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 their corresponding sets of terms and connectors are equal, the Dice measure is optimal.

Since in the embedded content measure the order of subexpressions is relevant, index expressions $I$ and $J$ are not optimally similar.

Although the sets of twigs for $I$ and $J$ are equal, the twigs measure is not optimal. This is caused by the fact that the twigs measure computes the average similarity over all pairs of twigs. This includes pairs that are not equal, such as conference on biology and conference in Holland.

Second, we focus on embedding of index expressions, as denoted by $\ll$. Consider the example index expressions $I =$ surfing in Holland and $J =$ surfing in sunny$\circ$Holland from section 3.2.2. The Dice measure, setting $\alpha = 1$

for reasons of clarity, computes the similarity based on the sets of terms $\mathsf{Terms}(I) = \{\mathsf{surfing}, \mathsf{Holland}\}$ and $\mathsf{Terms}(J) = \{\mathsf{surfing}, \mathsf{Holland}, \mathsf{sunny}\}$. Since these sets are not equal, the Dice measure does not return value 1. In stead, it results in $\mathsf{Dice}(I, J) = \frac{2\times 2}{5} = \frac{4}{5}$.

Since the full-product measure computes the similarity between index expressions layer by layer, the expression $\mathsf{sim}(\mathsf{Holland}, \mathsf{sunny} \circ \mathsf{Holland})$ has to be evaluated at depth one. By case two of the full-product measure, this expression results in $\mathsf{sim}(\mathsf{Holland}, \mathsf{sunny}) = 0$.

The resulting sets of twigs, not considering their depth, are $\mathsf{twigs}(I) = \{\mathsf{surfing\ in\ Holland}\}$ and $\mathsf{twigs}(J) = \{\mathsf{surfing\ in\ sunny}, \mathsf{sunny} \circ \mathsf{Holland}\}$. The similarities between the individual twigs are less than one. For instance, $\mathsf{twigsim}(\mathsf{surfing\ in\ Holland}, \mathsf{sunny}\circ\mathsf{Holland})$ involves computing $\mathsf{sim}_C(\mathsf{in}, \circ)$ and $\mathsf{sim}_T(\mathsf{surfing}, \mathsf{sunny})$, which both are less than one.

Third, consider $I = \mathsf{conference\ on\ (biology)\ in\ (Holland)}$. This single index expression contains two different twigs. The twigs measure also compares these different twigs and therefore is not optimal for this case.

Finally, the explanation of the line for subexpressions, denoted by $\preccurlyeq$, follows a similar line of argument as for embedding. Consider, for example $\mathsf{Holland}$ and $\mathsf{surfing\ in\ Holland}$. Clearly, the sets of terms and connectors are unequal, preventing Dice to be optimal. Since the heads of both index expressions are unequal, the layer by layer computation of the full product measure is not optimal either. Finally, note that the sets of twigs are not equal.

# 5 Related Work

In this section, related approaches to our work are described.

## 5.1 Frames

In the DORO (see [13] and [12]) project, noun phrases and verb phrases are normalized before being matched. The first normalisation phase, syntactic normalisation, consists of three steps: (1) elimination of redundant elements, (2) morphological normalisation by lemmatization, and (3) syntactic normalisation by mapping syntactically different but semantically equivalent forms onto the same form.

The third step has strong resemblance to computing twigs. In the DORO project, noun phrases are represented by by so called *frames* which consist

of a head and a number of modifiers. Both head and modifiers can contain nested frames. In the syntactic normalisation, *unnesting* is applied to flatten the frames, resulting in a multiset of unnested frames. Unnesting frames is done in order to raise recall.

As an example, taken from [13], the somewhat curious sentence

`man visited conference on software engineering`

might be transduced to the nested frame

$$[\texttt{visit}, [\texttt{conference}, \texttt{on}[\texttt{engineering}, \texttt{software}]]]$$

which is unnested to

$$\{[\texttt{visit}, \texttt{conference}], [\texttt{conference}, \texttt{on engineering}], [\texttt{engineering}, \texttt{software}]\}$$

Fuzzy matching is used in the DORO project. This means that (1) frames are matched partially after unnesting, and (2) that semantical knowledge in the form of hypernym relations is exploited.

The similarity of unnested frames is expressed as the product of similarities between head and modifiers. This is similar to our approach.

## 5.2 Tree Inclusion

In [11], Kilpelaïnen and Mannila describe a language for querying structured text based on tree inclusion. 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 hinge on equality of labels but uses approximate matching of strings by exploiting similarity functions for terms and connectors. In the introduction of the mentioned article the authors indicate that such "standard IR techniques should be added to the language". Although the authors claim that such techniques are largely orthogonal to preserving binary properties, we claim that computing similarity between index expressions involves more.

Our skeleton-content approach thus 'preserves' labels by taking into account their similarity. In a way, 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 the order of subexpressions is irrelevant, the corresponding inclusion pattern $O$ is not satisfied.

# 6 Conclusions

In this article, we devised several similarity measures for index expressions. Since many Information Retrieval tasks require numerical matching, these measures enable numerous new possibilities to exploit index expressions.

We investigated different views on the semantics of index expressions (order of subexpressions, embedding, and headedness) and formalised these into criteria. This enabled the design of corresponding similarity measures. Our similarity measures exploit the inductive as well as the structural representations for index expressions using profitable aspects of both.

We introduced measures that are only based on the contents of index expressions and similarity functions that exploit both structure and content. The similarity measures were proven optimal with respect to the corresponding criteria. In addition, other criteria, such as subexpressions, were also checked. Finally, we compared our approach with related work.

Future research can be directed towards including wildcards and variables in matching index expressions. In addition, other criteria for the semantics of index expressions may exist, leading to different optimal similarity measures. Several of the similarity measures provided in this article are not symmetric since they compute the degree of embedment. Symmetric variants of these measures can be defined readily.

We have implemented the mentioned similarity measures in a functional language. Large-scale experiments researching the effectiveness of index expressions in comparison to other descriptor languages are also considered an issue for further research.

# References

[1] F.C. Berger. *Navigational Query Construction in a Hypertext Environment.* PhD thesis, Department of Computer Science, University of Nijmegen, September 1998.

[2] T. Brus, M.C.J.D. van Eekelen, M. van Leer, and M.J. Plasmeijer. Clean - A Language for Functional Graph Rewriting. In *Proceedings of*

*the Third International Conference for Functional Programming Languages and Computer Architectures (FPCA '87)*, volume 274 of *Lecture Notes in Computer Science*, pages 364–384, Portland, Oregon, USA, 1987. Springer-Verlag.

[3] P.D. Bruza. Hyperindices: A Novel Aid for Searching in Hypermedia. In A. Rizk, N. Streitz, and J. Andre, editors, *Proceedings of the European Conference on Hypertext - ECHT 90*, pages 109–122, Cambridge, United Kingdom, 1990. Cambridge University Press.

[4] P.D. Bruza. *Stratified Information Disclosure: A Synthesis between Information Retrieval and Hypermedia.* PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[5] P.D. Bruza and L.C. van der Gaag. Efficient Context-Sensitive Plausible Inference for Information Disclosure. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 12–21, 1993.

[6] P.D. Bruza and Th.P. van der Weide. Two Level Hypermedia - An Improved Architecture for Hypertext. In A.M. Tjoa and R. Wagner, editors, *Proceedings of the Data Base and Expert System Applications Conference (DEXA 90)*, pages 76–83, Vienna, Austria, 1990. Springer-Verlag.

[7] P.D. Bruza and Th.P. van der Weide. The Modelling and Retrieval of Documents using Index Expressions. *ACM SIGIR FORUM (Refereed Section)*, 25(2), 1991.

[8] P.D. Bruza and Th.P. van der Weide. Stratified Hypermedia Structures for Information Disclosure. *The Computer Journal*, 35(3):208–220, 1992.

[9] J. Farradane. Relational Indexing Part I. *Journal of Information Science*, 1(5):267–276, 1980.

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

[11] P. Kilpelaïnen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 214–222, Pittsburgh, PA, USA, 1993.

[12] C.H.A. Koster. Fuzzy Matching using WordNet. Addendum to [13], Department of Computer Science, University of Nijmegen, Nijmegen, The Netherlands, 1998.

[13] C.H.A. Koster. Normalization and matching in the DORO system. Deliverable T3.1 of DORO project, Department of Computer Science, University of Nijmegen, Nijmegen, The Netherlands, 1998.

[14] C.J. van Rijsbergen. *Information Retrieval.* Butterworths, London, United Kingdom, 1990.

[15] B.C.M. Wondergem, P. van Bommel, and Th. P. van der Weide. Association Index Architecture for Information Brokers. Technical Report CSI-R9820, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, July 1998.

[16] B.C.M. Wondergem, P. van Bommel, and Th. P. van der Weide. Construction and Applications of the Association Index Architecture. In *Proceedings of CIW'98, the Conferentie Informatiewetenschappen 1998*, Antwerp, Belgium, December 1998.

[17] B.C.M. Wondergem, P. van Bommel, and Th.P. van der Weide. Nesting and Defoliation of Index Expressions for Information Retrieval. *Knowledge and Information Systems.* To appear.

[18] P. Wouda. Similarity between Index Expressions. Master's thesis, University of Nijmegen, Nijmegen, The Netherladnds, February 1997.