Graph-Based Concept Identification and Disambiguation for Enterprise Search

Falk Brauer
SAP AG, SAP Research
Dresden, Germany
falk.brauer@sap.com

Michael Huber[*]
ARITHNEA GmbH
Taufkirchen, Germany
michael.huber@arithnea.de

Gregor Hackenbroich
SAP AG, SAP Research
Dresden, Germany
gregor.hackenbroich@sap.com

Ulf Leser
Humboldt-Universitaet
Berlin, Germany
leser@informatik.hu-berlin.de

Felix Naumann
Hasso-Plattner-Institut
Potsdam, Germany
naumann@hpi.uni-potsdam.de

Wojciech M. Barczynski
SAP AG, SAP Research
Dresden, Germany
wojciech.barczynski@sap.com

ABSTRACT

Enterprise Search (ES) is different from traditional IR due to a number of reasons, among which the high level of ambiguity of terms in queries and documents and existence of graph-structured enterprise data (ontologies) that describe the concepts of interest and their relationships to each other, are the most important ones.

Our method identifies concepts from the enterprise ontology in the query and corpus. We propose a ranking scheme for ontology sub-graphs on top of approximately matched token q-grams. The ranking leverages the graph-structure of the ontology to incorporate not explicitly mentioned concepts. It improves previous solutions by using a fine-grained ranking function that is specifically designed to cope with high levels of ambiguity. This method is able to capture much more of the semantics of queries and documents than previous techniques. We prove this claim by an evaluation of our method in three real-life scenarios from two different domains, and found it to consistently be superior both in terms of precision and recall.

Categories and Subject Descriptors

H.3.1 [**Content Analysis and Indexing**]: [Thesauruses, Indexing methods]; D.2 [**Data Structures**]: Graphs and networks

General Terms

Algorithms

1. ENTERPRISE SEARCH

Enterprise Search (ES) is different from searching a digital library or the web in many aspects [8]. One of the major differences is that the corpora to be searched are highly specific and centered around a set of *Enterprise Concepts (EC)*, which describe the company's business, such as products, purchase orders, etc. These ECs are often captured in knowledge bases in the form of enterprise ontologies [17]. Using the knowledge about ECs for enterprise search promises to capture more of the semantics within documents and within queries and to thus improve search results. But this promise is impeded by numerous problems. In particular, it is difficult to identify ECs in text due to syntactic heterogeneity and ambiguity. This paper describes a system for improving the identification and disambiguation of ECs, which in turn helps to improve the quality of search results. Before we describe our approach in more detail, we will describe a number of use-cases in which a technique like the one presented here offers real business value.

[*]Work has been done while affiliated with SAP Research.

1.1 Searching in Developer Networks

Developer networks (DNs) are community platforms for customers and developers of software companies that help find solutions to given problems. Typically, such a network consists of a large unstructured knowledge base of answered questions, technical notes, software documentation, etc., which can be searched by customers. DNs have proven themselves in reducing the cost of support by encouraging customers to self-support, assuming that most customer problems are re-occurring. Consequently, most software companies today provide a DN [44].

Documents within a DN capture information about a specific topic, such as the solution to a known bug in a product or details on configuring a piece of software. However, finding a solution to a specific problem is not as easy as it might seem. In fact, it has been estimated that many of the forum questions have already been answered at the time they are posted, but were not found [41].

Evidently, queries as well as documents deal with products or concepts of the company (ECs) [34]. A natural way to improve response quality is thus to first identify all ECs in a query and match those to ECs in documents. However, usually neither queries nor documents mention concepts in the same way as they are described in a knowledge base. Users often use ad-hoc abbreviations, drop tokens in multi-token concepts, mis-spell tokens, intersperse their queries with error messages, trace snippets or code examples. We highlight these difficulties with a concrete system, which also serves as our running example throughout the paper. Note, that our approach is especially helpful in answering queries that are short text messages (> 5 tokens) rather than only keywords, which comprise 10% to 15% of the distinct queries, i.e. 600,000 per week, in our example scenario.

The *SAP Community Network* (SCN) is a developer network for all users of SAP software. Thus, almost every entry in the SCN must relate to a SAP software product. The concepts used in SAPs products and all their related components are maintained in SAP's inhouse terminology *SAPTerm*[1]. SAPTerm comprises several ten thousands of software components ("Component" in Fig. 1), organized in a disjoint union of hierarchies with a maximum depth of eight. Combinations of components form the blueprints for software products. In addition, SAPTerm contains hundred thousands of technical terms ("Term" in Fig. 1) and corresponding acronyms ("Variant" in Fig. 1), each associated with certain software components. Technical terms also are associated among each other, pointing to synonyms, homonyms, etc. The different classes of ECs are called EC types (bold border in Fig. 1). Each EC has a number of attached textual attributes, such as long- and short-name of components ("long_form" and "long_form" in Fig. 1) or the canonical name for technical terms ("canon_form" in Fig. 1)

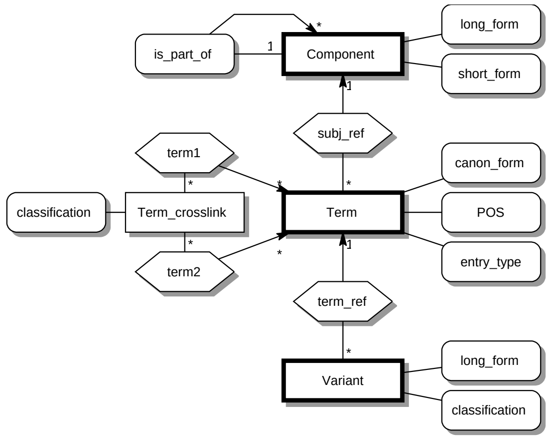


Figure 1: Schema of SAP Terminology Database

The availability of this background knowledge can be leveraged to increase precision and recall when searching the SCN. But on average only one third of all forum entries contain at least one component name exactly as it appears in SAPTerm. This has many reasons. First, EC names are often composed of multiple tokens, some of which are often dropped in texts and queries. Second, EC names and their individual tokens are highly ambiguous, i.e., they appear multiple times in different contexts in SAPTerm. Third, names are often abbreviated (and abbreviations are, again, highly ambiguous). Last, the content of the SCN - as with many other DNs - is usually not edited but consists of ad-hoc queries and answers that are full of typos, ungrammatical sentences, ad-hoc abbreviations, jargon, false names, etc. Additionally, as mentioned before, ECs are organized in hierarchies of concepts that subsume each other. Having matched several ECs, a search engine has to decide which level of the hierarchy is the most suitable to be considered in an answer. Thus, even if customers would use exactly the predefined component names from SAPTerm in their searches, they would still miss important hits.

1.2 Further Usage Scenarios

Proper identification of ECs in text has many applications. Here, we shortly describe two more of them. Note that these, in addition to searching DNs, are also addressed in the evaluation in Sec. 6.

Routing support requests. SAP's customers can subscribe to commercial customer support. Thus, customers send support requests via email and are provided with professional answers within certain time constraints. Every such customer request must be routed to an appropriate technician. The 1st level support routes inquiries to experts of the 2nd level support, each being responsible for a small set of software components. The routing requires manual effort to ensure the correct mapping between request text and addressed software component. If ECs could be identified automatically, this routing could be sped up considerably, if not automated completely.

Searching movie reviews. Searching movie reviews is conceptually similar to searching DNs. Entertainment companies maintain platforms for users to contribute movie reviews. Before buying certain articles such as DVDs, users can browse opinions about a movie. Movie reviews and queries for them reference movies or actors directly and indirectly, e.g., by co-occurrence of a actor's real and cast name. Leveraging a graph-structured knowledge base such as the Internet Movie Database (IMDB)[2] allows for identification of movies and disambiguation of, e.g., actor names in movie reviews when referenced only by their ambiguous surname.

1.3 Concept Identification in Documents

Our approach for solving the problems outlined in the preceding sections is based on the identification of concepts from an enterprise ontology in queries and documents. More specifically, our system first takes a query and a DAG-structured[3] enterprise ontology as input. It identifies the most relevant ECs for each text and for the query. In doing so, it also considers information that is directly or indirectly associated with the EC within the ontology. For each text (and the query), a subgraph of the enterprise ontology is computed, which consists of all identified ECs and their neighborhoods. These graphs are called *document concept graphs (DCG)* and consist of nodes that represent matches between text and ECs' derived from the ontology. Each EC within a DCG is scored respectively. In the last step, all documents are ranked by computing the similarity of the documents' DCG with the DCG of the query.

The document ranking using relevant graphs retrieved via Information Extraction (IE) was implemented successfully in the RankIE System (Ranking on IE Graphs), presented in [7]. In this paper we concentrate on the problem of identification of the most relevant ECs in documents wrt. the given enterprise ontology. We evaluate our solution in all three previously discussed use cases.

In the following section, we present the main challenges in EC ranking and disambiguation by means of an example. Our methods for identifying ECs with high precision and recall despite spelling errors, high level of ambiguity, and incomplete mentions are introduced in Sec. 3. EC ranking is explained in Sec. 4. After considering related work in Sec. 5, we discuss experiments that illustrate the practicability of our approach in Sec. 6.

2. PROBLEMS IN ENTERPRISE SEARCH

As explained above, the most prominent difference between enterprise search and other search scenarios is the availability of a set of interconnected ECs. A typical (naive) approach to enterprise search is to exactly match those ECs in documents and use the intersection of matched ECs in the query and in a document from the corpus as a score. However, this fails to (a) leverage the relationships between ECs as encoded in the enterprise ontology and (b) falls short in adequately considering lexical variations. In this section, we describe the problems using a sample of 120,000 forum entries created in the beginning of 2008. One such forum entry is shown in Fig. 2.i. We start our analysis using only the 16,000 long and short names of components in SAPTerm and later also consider technical terms and acronyms. In the following, we use the word "term" for any of these, i.e., textual values attached to components, technical terms, and acronyms.

[1]`http://help.sap.com/content/additional/terminology`

[2]`http://imdb.com/`

[3]DAG: Directed Acyclic Graph

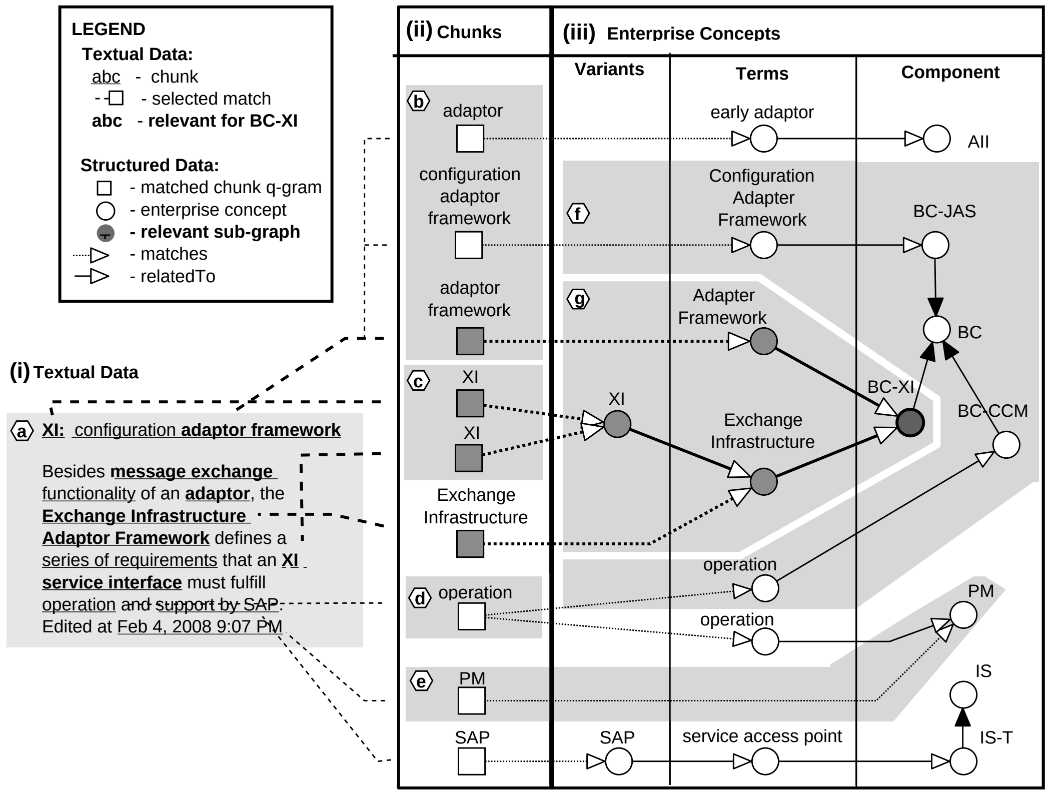


Figure 2: Mapping the text of a SCN forum entry to the enterprise ontology *SAPTerm* : Segments of the text are mapped to concepts of the ontology. Partial, fuzzy, and indirect concept identification techniques are required to achieve a good recall and precision.

2.1 Recall

A dictionary-based approach identifies only 37,000 exact occurrences of component names in the 120,000 forum entries. Assuming that almost every forum entry addresses a problem in the setting of at least one software component, this rate certainly is unsatisfactory. Only 10% of matched names were longer than one token while over 83% of the terms in SAPTerm contain more than one token. Thus, *partial matching strategies* are needed to increase the recall. Partial matches are more ambiguous than the full names are.

Another impediment to high recall is the fact that SCN forum content often is error-prone. Therefore, *fuzzy lookup strategies* are required that are able to find names even in the presence of typos. One such example can be seen in Fig. 2.ii.b where the user typed “adaptor” though “Adapter” would have been the correct term. The drawback is that a significant number of terms are highly similar. Thus, any fuzzy search strategy adds to the problem of ambiguity and hence harms precision. Actually, after incorporating partial and fuzzy matching methods to increase recall, we measured that 98% of all matches were ambiguous.

2.2 Precision

Ambiguity is a major problem in any sizable enterprise ontology. In SAPTerm, only 30% of all terms are distinct and consist of only 23,000 distinct tokens. The most frequent token in component names, “Management”, appears in approximate 3,600 terms. A natural way to cope with such ambiguity is to consider additional, disambiguating information attached to the concepts. This is possible by using the technical terms and acronyms assigned to components. An extreme example for ambiguity is the component name “PM” (long name is “Plant Maintenance”). Only about 10% of “PM” matches refer to the EC in our knowledge base, while 90% of “PM” matches occured in a timestamp (see Fig. 2.ii.e). For instance, one may consider the co-occurrence of “PM” with the match on “operation”, a term assigned to “PM” in the knowledge base (see Fig. 2.ii.d). This additional match would raise the relevance of “Plant Maintainance”. However, the exemplary entry contains many more additional hits that are assigned to the concept “BC-XI” (see Fig. 2.ii). As a consequence, our approach would rank “BC-XI” best. Note that such a technique allows the proper identification of components not only in situations where an ambiguous concept is found, but also in cases where no component name was matched at all. E.g., we may recognize the component “BC-JAS” (abbreviaton for “Java Application Server”) by the context information that “Configuration Adapter Framework” (see Fig. 2.ii.f) is a technical term related to the component “BC-JAS” which would otherwise be neglected.

Another strategy to cope with ambiguity is to simply disregard all possible matches except one. A popular choice is to always consider the longest match as the correct one [51]. However, such a method often fails in our case. For instance, the match “Configuration Adapter Framework” in Fig. 2.ii.b is the longest match between text and ECs. But, in the given scenario the shorter match “Adapter Framework” provides an clue towards “BC-XI”. Indeed, here the user addressed “configuration” of the “Adapter Framework” and not the “Configuration Adapter Framework”. Therefore, all matches should be considered, though longer matches might obtain a higher weight.

2.3 Aggregation and Abstraction of ECs

The dictionary-based approach failed completely for finding abstract components such as “AP” (aka. “Application Platform”) or “BC” (aka. “Basic Components”). Those were never matched directly in any of the 120,000 forum entries. However, in many situations such abstract EC should be ranked top, e.g., when a user addresses a integration problem between two or more “subcomponents”. We can identify abstract components by leveraging the pre-defined relations among ECs and aggregate matches of related more concrete ECs to abstract ECs. However, this is not as simple as it seems. Consider Fig. 2.iii.g. Here, we need to rank "BC-XI" best and not "BC", because the match of "BC-JAS" was caused by ambiguity. Thus, carefully chosen aggregation strategies are required to *balance selections of more concrete or more abstract ECs*.

3. MATCHING ENTERPRISE CONCEPTS

Before we describe our ranking procedure in Sec. 4, we explain how we find potential ECs in a text. The main difficulty in this step is that terms assigned to ECs often consist of many tokens of which only a few may appear in a text, often interspersed with other words and possibly in different orders. Further, we need to obtain meaningful and detailed weights for each occurrence of a term in a text as a basis for ranking later on.

There are two approaches to tackle this problem [49]: (1) sliding windows, or (2) matching text chunks. The first approach slides a window over the text and scores each window by searching all its tokens in the list of all tokens contained in any of the terms. When terms also contain many "normal" words, as in the case of SAPTerm, this method typically suffers from a very large number of token matches, resulting in a non-zero score for virtually every window, thereby decreasing the performance. This problem becomes worse as the window size increases. In the case of SAPTerm, where the longest terms comprise twelve tokens, a window size above 15 would have to be used to safely recognize all terms, taking into account that token occurrences are usually interspersed. Recently, Chandel et al. [10] proposed to prune the list of recognized tokens by using a TF-IDF-based ranking scheme. However, a TF-IDF-based approach assumes statistical independence of token occurrences, which leads to improper estimates when tokens are highly ambiguous and the order of tokens is important (e.g, "Adapter Framework" or "Configuration Adapter Framework"). Another drawback of the sliding window approach is that it disregards the grammatical structure of a sentence. Thus, a fragment such as "Configuration failed because the XI Adapter Framework ..." would potentially (depending on the window size) obtain a high score for the term "Configuration Adapter Framework" compared to the term "Adapter Framework", an error no human reader would ever make. Furthermore, "Adapter Framework" would get a higher TF-IDF than "XI", assuming a similar IDF, because it is built from only one token. Further, "because" would obtain a high support through IDF because it occurs twice in SAPTerm.

The second approach partitions the text into chunks and then matches each chunk against the ontology. Assuming a sensible chunking, this method adequately deals with token sequences (as the whole chunk is used for comparison and not its individual tokens) and adapts the number of tokens that are matched. The conventional approach for achieving a sensible chunking is using a shallow parser [48], which breaks a sentence into grammatical units, among which verbal phrases (*VP*) and noun phrases (*NP*) are the most important ones. Such a linguistically-informed chunking also solves the problem of ungrammatical matches and prunes many less relevant matches, such as "because".

Due to these advantages, we use the 2$^{\text{nd}}$ approach. SAPTerm's textual contents, such as component names or technical terms, are (almost) always sequences of nouns potentially accompanied by some modifiers such as adjectives. Therefore, we can safely assume that their occurrence in a text will be inside a NP, which are further used as chunks for matching. NPs are among those linguistic structures that can be identified quite accurately.

Our entire term matching process breaks up into three steps: (1) document pre-processing, (2) indexing the enterprise ontology to speed up the matching phase, and (3) finding and weighting matches between the NPs of the document and the indexed terms from the enterprise ontology. After matching, all discovered terms are used to rank ECs, as described in Section 4.

3.1 Text Pre-Processing

All texts are subjected to pre-processing. Therein, we first separate grammatical text from ungrammatical insertions, such as exception traces or code snippets. Separation is based on simple heuristics that mostly use specific regular expressions to identify typical insertions. Ungrammatical text parts are not considered further. Grammatical text parts are broken into sentences. We prune all sentences as ungrammatical text that would lead to unusual long sentences (e.g., 1000 characters). Within sentences, NPs are tagged by a state-of-the-art shallow parser.

3.2 Indexing the Ontology

To speed up the matching phase (see Sec. 3.3), we index all terms of the ontology. As we also want to find partial matches, thus we not only index entire terms, but also their token subsequences, i.e., all their token *q-grams*. For instance, the term $t =$*"Configuration Adapter Framework"* $\equiv$ abc results in six q-grams being indexed, i.e., $\{\tilde{t}_1, \ldots, \tilde{t}_6\} \equiv$ {a,b,c,ab,bc, abc}. Because we use all q-grams (from $q = 1$ up to the length of the term), this method also captures reordered appearance of tokens in a text. Thus, the occurrence *"Adapter Configuration"* would still match the above term, but with a lower score than *"Configuration Adapter"*.

We build one index structure per term class (see Fig. 1). Each index is organized as a prefix tree that maps from q-grams to terms. Its size grows at a manageable rate compared to a single token index, because string values can be compressed by incremental prefix-encoding (see [6] for details).

A particular problem occurs for extremely ambiguous q-grams, i.e., those q-grams that occur frequently in the ontology. If those are found in a text, the number of matches with the ontology becomes extremely large (several thousand for SAPTerm and hundreds of thousands for IMDB). Since such extremely ambiguous q-grams carry very little information but incur high processing cost, we disregard all q-grams that are more frequent than a threshold k, but ensure that at least one q-gram of each term remains in the index. We use a value for k ($k = 250$) that is so high that only very ambiguous q-grams are pruned, leaving the final ranking essentially unchanged. Using prefix encoding and pruning of ambiguous q-grams, the resulting index size for SAPTerm is only twice that of a simple token index, but captures much more semantics in respect to the sequence of tokens.

We add two more remarks: First, our pruning strategy inherently prefers longer q-grams over shorter q-grams, as shorter q-grams are in general more frequent and are therefore pruned more often. Second, we deliberately build one index per term class, as the distribution of tokens and token q-grams is quite different within each class. Accordingly, a term in the class "component - long_form" should be valued differently from a match in the class "technical term - canon_form" (see Fig.1). Thus, e.g., the q-gram "Management" in "component - long_form" remains in the index (still highly ambiguous), while it is pruned for "technical term - canon_form".

3.3 Matching NPs and Ontology Concepts

At runtime, each NP is translated to a disjunctive query, containing all its token q-grams, against the ontology index. We retrieve all matches from the index and keep only the longest one for each matched term. For instance, the NP $c \equiv$ abc results in the query for the six q-grams $\{\tilde{c}_1, \ldots, \tilde{c}_6\} \equiv$ {a,b,c,ab,bc,abc}. Given a term $t \equiv$ axbc with ten q-grams $\tilde{t}_1, \ldots, \tilde{t}_{10}$, we would find four matched q-grams, but derive only two of them, i.e. $\tilde{c}_1 \equiv$ a $\equiv \tilde{t}_1$ and $\tilde{c}_5 \equiv$ bc $\equiv \tilde{t}_6$. Each matched pair of term and NP q-gram $\tilde{c} \equiv \tilde{t}$ is weighted with respect to its coverage of the matched term t using the following formula:

$$w_{\mathrm{q}}(\tilde{t}, \tilde{c}) = \frac{|\tilde{t}|}{|t|} \cdot p_{\mathrm{T}}.$$

Here, p_{T} is a factor specific for each term class. Using this factor one can, for instance, give long names of components a higher weight than short acronyms. Thus, the match of $\tilde{t} \equiv$ bc derived from $t \equiv$ axbc would obtain a weight of $w_{\mathrm{q}}(\tilde{t}, t) = 2/4$ assuming that $p_{\mathrm{T}} = 1$. Note that these weights are pre-computed and stored in the term index.

As explained in Sec. 2, fuzzy matching is important to allow for linguistic variations and typos in texts. To this end, we actually consider as matches all q-grams $\tilde{t}$ that are sufficiently similar to a NP q-gram $\tilde{c}$. We use pre-fix filtering [11] to speed up this look-up.

Similarity between term and NP q-grams is computed using Levenshtein [42] distance sim, length-normalized by the number of characters in the NP q-gram $|\tilde{c}|$. In principle, we consider all q-grams as matches for which this value is larger than a similarity threshold s. However, we found that in practice a further scaling of the similarity value with $1/sim$ performs better, since it punishes a high edit distance more effectively. Together, the similarity $w_{\mathrm{sim}} \in [0, 1]$ is computed as

$$w_{\mathrm{sim}}(\tilde{t}, \tilde{c}) = \frac{1}{sim(\tilde{c}, \tilde{t}) + 1} \left(1 - \frac{\min \arg(sim(\tilde{c}, \tilde{t}), |\tilde{c}|)}{|\tilde{c}|} \right).$$

Ambiguity is the most crucial problem when matching terms. Not only q-gram matches derived from one NP q-gram $\tilde{c}$ are afflicted with ambiguity. Sequences of matches overlap and intersect each other. For instance assume that we found with $c \equiv$ abcxd the four matching term q-grams: $\tilde{c_1} \equiv$ ab $\equiv \{\tilde{t_1}, \tilde{t_2}\}$, $\tilde{c_2} \equiv$ bc $\equiv \{\tilde{t_3}\}$ and $\tilde{c_3} \equiv$ d $\equiv \{\tilde{t_4}\}$. Therein, the semantic of the NP sub-sequence $\tilde{c_1} \cup \tilde{c_2} \equiv$ abc remains unclear. To alleviate this problem, previous approaches prune (1) shorter matches and (2) more ambiguous ones. Such a method often fails when identifying ECs as discussed in Sec. 2. Thus, we keep all matches and weight each $\tilde{t} \equiv \tilde{c}$ with respect to the length of the match sequence it is contained in. Such a match sequence c' of a matching q-gram $\tilde{t} \equiv \tilde{c}$ is the subsequence of tokens from c that is derived by extending $\tilde{c}$ in c recursively with overlapping and also matching NP q-grams. Finally, the ambiguity weight is computed as

$$w_{\mathrm{amb}}(\tilde{t}, \tilde{c}) = \frac{(1 - p_{\mathrm{out}})}{N(\tilde{t})} \cdot \frac{|\tilde{t}|}{|c'|}, \tilde{c} \in c'. \qquad (1)$$

Here, $N(\tilde{t})$ is the number of term q-grams matched by the same $\tilde{c}$ and p_{out} is a tunable parameter to adjust the individual weight of a match by a general tendency that $\tilde{c}$ does not refer to any term in the ontology, but rather to a term outside the known domain. Note that p_{out} also ensures that interspersed matches get a lower weight than consecutive ones. Further, we use the raw $N(\tilde{t})$ without logarithmic smoothing because terms which occur frequently in the ontology, occur similarly frequently in a text. Thus, the discriminative nature of $N(\tilde{t})$ is balanced through the term frequency in a text as discussed later on.

Figure 3 summarizes the computation of w_{amb} and w_{q} described in this section. The NP $c \equiv$ xabcye is broken into its q-grams $\tilde{c}_1, \ldots, \tilde{c}_{\mathrm{n}}$. Five term q-grams $\tilde{t}_1, \ldots, \tilde{t}_5$ are matched by a $\tilde{c}_i$. The value w_{amb} is computed for each $\tilde{t}_{\mathrm{i}}$ (assuming $p_{\mathrm{out}} = 1/5$). Therein, the semantically vague match sequence of $\tilde{t}_1, \ldots, \tilde{t}_4$ is $c' =$ abc leading to $|c'| = 3$. The weight of q-gram $\tilde{t}_5$ is not lowered through other intersecting matches. Finally, w_{q} estimates the coverage for each term q-gram $\tilde{t}$ with respect to its originating term t. An equal weight $p_{\mathrm{T}} = 1$ for all term classes was used to calculate w_{q} for simplification.

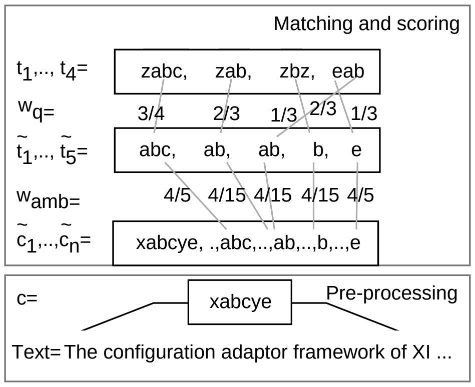


Figure 3: Matching and confidence computation

3.4 Scoring Ontology Terms

Individual matches in a NP c are aggregated into a confidence score $w_{\mathrm{conf}}(t, c)$ for a term t by summing the aggregated weights of all matched q-grams $\tilde{t} \equiv \tilde{c}, \tilde{t} \in t, \tilde{c} \in c$. We normalize the individual scores by the number of non intersecting match sequences in a NP, denoted as $S(c)$, kept from calculations of w_{amb} (see Eq. 1) to ensure $w_{\mathrm{conf}}(t, c) \in [0, 1]$. This leads to

$$w_{\mathrm{conf}}(t, c) = \sum_{\tilde{t} \in t} \frac{w_{\mathrm{q}}(\tilde{t}, \tilde{c}) \cdot w_{\mathrm{sim}}(\tilde{t}, \tilde{c}) \cdot w_{\mathrm{amb}}(\tilde{t}, \tilde{c})}{S(c)}. \qquad (2)$$

The calculated confidence score is used for ranking ECs with respect to the whole document as explained in next section. We store for each term match the ID of the EC that the term is assigned to, the confidence score, and the span of the NP, defined as begin and end index in the text.

4. GRAPHICAL MODEL AND RANKING

In the following, we introduce a graph-based model for ranking ECs. Further, we discuss the application of ranked ECs for document retrieval.

4.1 Document Concept Graphs

To rank ECs, we represent the text and the ontology in a unified graph-based model, the so-called document concept graph. To rank ECs, the following points have to be addressed: First, a term might occur in several noun phrases (NPs) of a document. Next, several terms might be assigned to one EC, such as the long and short name of a software component, as illustrated in Fig. 1. Further, the relations of ECs as defined in the ontology have to be taken into account to (1) disambiguate matches and (2) rank ECs that were not directly matched through their assigned terms.

The *document concept graph (DCG)* represents term matches and ECs in their ontological context. Formally, a DCG is a directed acyclic graph $G = (N, E, L_E, L_R)$. N is a set of nodes, $E \subseteq N \times N$ a set of edges, and L_{N} and L_{E} a set of node and edge labels respectively. Nodes are either of type `Term` or of type `EC`. Each term occurrence in an NP of the text is represented as a node (e.g., two distinct matches for "XI" in Fig. 2.ii.c). In the following, we denote a term match in a NP by t. Further, we denote the confidence score between a term match t and an EC ε (see Eq. 2), as $s_{\mathrm{conf}}(\varepsilon, t)$. Several terms assigned to an EC lead to a higher certainty for this EC, e.g., occurrences of long and short names. Several occurrences of identical terms in several NPs increase certainty as well.

The graph-based model supports two basic types of relationships: "match"-edges between matched terms and ECs and directed "related to"-edges, between ECs as derived from the ontology. A configuration file determines the direction of relations between EC-types, that is used to propagate scores in a DCG, e.g., from *Variant* to *Component* for SAPTerm (see Fig.1). Such configuration is simple for ontologies with less EC-types, such as SAPTerm or IMDB (see Sec. 1). Approaches for weighting the importance of EC types by statistics (e.g., [53]) may automate such configuration in future.

An exemplary DCG that illustrates typical match situations is shown in Fig. 4. Here, two NPs, namely c_1, c_2, match six terms $t_1, \ldots, t_6$. Eight ECs, i.e. $\mathrm{EC}_1, \ldots, \mathrm{EC}_8$, are either directly or indirectly connected to the matched terms.

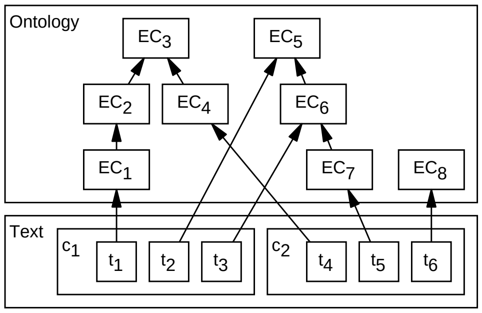


Figure 4: Illustration of a Document Concept Graph.

4.2 Ranking Function for ECs

A DCG is used to score ECs for a given text. Based on the decomposition of a text into NPs with assigned term matches, we can express the score of a directly matched ECs simply as the sum over confidence scores of all terms that are assigned to an EC and were matched by an NP. However, many relevant ECs are not directly matched. Therefore, we introduce matching paths that spread confidence scores through the whole DCG. Let $\mathrm{MP}(\varepsilon, t)$ denote such a matching path, i.e. a connected sequence of edges $e_1, \ldots, e_{\mathrm{m}} \in G$ that link a matched term t with an EC ε. The weight of the "match"-edge $e_1 \in \mathrm{MP}$ is determined by the confidence score (see Sec. 3.3). The weights of edges $e_{\mathrm{k}} \in \mathrm{MP}, k \neq 1$, that were derived from the ontology, are adjusted by a decay as $w(e_{\mathrm{k}}) = (1 - p_{\mathrm{self}}), k \neq 1$. The parameter p_{self} specifies the extend to which the start node ε of e dominates the scoring.

We also assign weights to nodes in the DCG. In the following, let ε denote any EC, D a text, t any matched term in a NP of D, and $\mathrm{MP}(\varepsilon, t)$ a matching path. The initial weight of any t in a DCG is $w(t) = 1/|D|$, where $|D|$ is the number of all term matches, thus ensuring an upper bound of 1 for the final score. The contribution of matching paths to the score of an EC depends on its constituent edge sequence $e_1, \ldots, e_{\mathrm{n}} \in \mathrm{MP}$ that connects matches and ECs. To summarize this section, we compute the score for an EC $s(\varepsilon, D)$ as

$$s(\varepsilon, D) = \sum_{t \in D} \sum_{\mathrm{MP}(\varepsilon,t)} w(t) \prod_{e \in \mathrm{MP}} w(e). \qquad (3)$$

The derived scores for ECs lead to a novel, fine grained ranking of ECs using matching and aggregation strategies of various granularity. For the example in Fig. 2, we obtain a ranked list of several hundred ECs in the order "BC-XI", "BC", "AII", etc. Indeed "BC-XI" is the most relevant EC with respect to the given text. It gets a significant higher score than the directly matched component "PM", because of the support through many matched terms in its ontological context. Second and third ranked ECs are meaningful as well in the context of a integration problem.

Furthermore, we realize that "Adapter Framework" in the constituent sub-graph of "BC-XI" is a more likely interpretation for the corresponding text segment than "Configuration Adapter Framework" in the sense of "BC-JAS", and that "operation" is more likely to be interpreted in the sense of "BC" than in that of "PM".

4.3 Document Retrieval using ECs

Our approach aims to increase the performance of document retrieval in enterprise search. In the following, we explain briefly the application of the computed scores to rank documents.

After document processing as described in previous sections, the highest scored ECs and their subsumed sub-graphs are stored. The sub-graphs capture the most likely disambiguated sense of matches. The EC scores capture the relevance for each EC in the actual context. User queries are analyzed in the same way. A user may re-rank ECs and discard false positives (see [7] for details). For computing the similarity between a query and previously processed documents, we represent them as vectors of EC scores, denoted as $\langle T \rangle$, that encode their respective graphs' structure. To rank documents, we compute the cosine similarity between query T_{query} and a document T_{doc} as

$$sim(T_{\mathrm{query}}, T_{\mathrm{doc}}) = cos(\theta) = \frac{\langle T_{\mathrm{query}} \rangle \cdot \langle T_{\mathrm{doc}} \rangle}{||T_{\mathrm{query}}||\,||T_{\mathrm{doc}}||}. \qquad (4)$$

Notice that the results of such an approach, although it captures much more semantics than traditional Information Retrieval (IR) techniques, are narrowed by the coverage of the ontology with respect to the corpus. In order to address a broader range of queries, our document ranking method is combined with previous IR systems by using rank aggregation techniques, such as [20].

5. RELATED WORK

This work touches upon several research areas. We shall discuss its relationship with coreference resolution, semantically enhanced document retrieval, search within (semi-)structured data sources and ontologies, named entity recognition, as well as concept disambiguation and identification.

Coreference Resolution. The problem discussed in this paper can be viewed as a graph-based deduplication problem (see [21] for a recent survey), where one of the objects under study is described in a structured form (the enterprise ontology), whereas the other is described in an unstructured fashion (forum entries, query). Recent work has started to address less rigidly structured instances, such as XML objects (e.g., [45]). We are not aware of deduplication approaches encompassing unstructured and structured data.

Semantically enhanced Document Retrieval. Information retrieval using structured knowledge bases is not an entirely new idea. Many approaches leverage WordNet as an ontology (e.g., [46, 46, 36, 43]) that captures word senses and not ECs. Multi token concepts are not addressed, which are of intrinsic importance in enterprise search. Approaches in this a area augment documents and queries by synonym sets acquired during a rather simple, graph intersection based disambiguation step. The ambiguity in WordNet is much lower, e.g., [43] reports on an average number of 3.1 senses per term. Further, our approach provides a list of EC subgraphs with scored concepts. [37] proposes to assign documents to an ontology for enterprise search, but does not suggest a method for matching and scoring ECs. EntityRank [13] ranks documents that are linked by recognized concepts. Pre-defined relationships are not addressed.

Keyword search on (semi-)structured data. Some elements of our ranking schemes resemble ideas proposed in the area of keyword search over (semi-)structured data. For a given keyword query a list of ranked joint tuple trees (JJTs) of a database or subtrees of a XML-document are generated that match all keywords. Most methods leverage IR-style ranking. PageRank is favored by [3, 30]. In our case, the direction of relations is fixed by the type of documents that are under investigation, e.g., for movie reviews only the relations towards movies are relevant. A TF-IDF-style ranking was discussed in [33, 28, 35, 54] by interpreting a query and JJTs as "bag of words", which proves inadequate when queries are documents as in our case. Further, less compact JJTs are pruned intensively [35, 32, 29, 26, 23], which leads to a decrease in accuracy for long queries (longer than eight tokens), as indicated in the experiments of [54]. To the best of our knowledge, extremely long, ambiguous queries (such as small documents) have not been studied in this area.

Keyword search on ontologies. Many approaches tackle search within ontologies, such as [47, 19, 31]. NAGA [31] ranks answers for structured queries. [47] provides answers for keyword queries on ontologies. It uses a spread activation method in the ontology graph comparable to PageRank. SWOOGLE [19] additionally leverages character q-gram similarity for matching queries and concepts. However, character q-gram similarity performs poorly for short terms such as acronyms (e.g., discussed in [52]). Therefore, we do not use this feature, but rely on edit distance.

Geographic Information Retrieval. A closely related field is geographic information retrieval (GIR) [39]. Given a query, GIR tries to match the geographic references from the query with geographic information extracted from documents using a spatial vicinity (see for instance [12]), which is not possible in our case. Web-a-Where [2] uses geographic taxonomies to determine the focus area of a web page. Recognized location names are assigned a confidence value in certain match situations, e.g., 0.5 if the location is ambiguous. Finally, the most relevant abstraction level in the taxonomy, an instance of city, state, or country is chosen, given a list of matched locations with different abstraction levels. There are important differences to our approach: Geospatial names usually comprise much fewer tokens (usually one). Our approach addresses a much higher level of ambiguity and computes confidence scores based on the constituent match situation between text and ECs.

Named Entity Recognition. Rule based and supervised learning methods for named entity recognition aim to extract unknown concepts, which is beyond the scope of this paper. Many efficient algorithms for dictionary matching have been proposed [42]. So far, only [10] addressed the recognition of *top-k* multi-token concepts using dictionaries. Matches in a sliding text window are weighted by token-based TF-IDF. However, such weighting tends to provide improper estimates for ECs that contain highly ambiguous tokens as discussed in Sec. 3. Furthermore, our approach ranks the identified top-k ECs for a complete text and considers implicit occurrences.

Concept Disambiguation. The problem of concept disambiguation in text has been studied intensively. There are two general approaches in this area. The first class of approaches employs data acquired during bootstrapping or supervised learning that is stored in an entity profile for each concept and is used for disambiguation at run time [18, 22, 4, 18]. Among the use cases presented in Sec. 1 only one offers a large annotated training corpus. A comparison to a machine learning approach leveraging this data is discussed in Sec. 6. Other approaches for disambiguation use structured reference data, similar to our approach. For instance [27] presented an application specific approach to disambiguate person names. [16] employed Wikipedia as reference data. Flat lists of context concepts are used for disambiguation. Approaches such as [46, 25, 36] use WordNet for disambiguation. They essentially target the disambiguation of words, not multi token concepts. Further, all discussed approaches disambiguate explicitly mentioned concepts in texts and do not consider implicit mentions.

Concept Identification. Some previous work approached the problem of linking an unstructured data source to a structured one. [24] proposed relevance ranking of directly matched life-science concepts using IDF-style word weights and weights that prefer "correctly" ordered tokens with respect to EC names. Our approach additionally considers the structure of the ontology, uses token q-grams to match EC names, and weights occurrences by the constituent match situation. The work [40] investigated the identification of record-like ECs in text – no relations among ECs were considered. EROCS [9], the most similar system, allows the identification of ECs organized as trees. The ECs (also indirectly matched ones) having the highest aggregated TF-IDF scores are identified within sentence boundaries, which are finally used to determine an optimal segmentation of a text with regard to ECs. Thus, EROCS assumes that each sentence relates to one concept. A fine grained scoring was not targeted. Further, our approach outputs a ranked list of entities that are mentioned across sentence boundaries, not a segmentation.

6. EXPERIMENTS

We conducted an experimental study to compare our system's performance in ranking ECs with respect to documents. Assuming that users are provided with an interface that allows to refine the information needs wrt. ECs, EC identification in documents is the most crucial processing step of the system.

We are not aware of any other method that solves exactly the same problem we tackle in this work. However, to implement a comparative study, we use three data sets to compare to related approaches [33, 3, 30, 5, 40, 9, 22]. Furthermore, we study the influence of the most important parameters of our ranking model.

The experiments were conducted on a desktop machine with a 2.0GHz double core processor and 3GB RAM running Windows Vista. All DCGs were stored in a database. The system itself was implemented in Java using the RapidUIM platform [7]. For token based TF-IDF scoring, ECs were indexed with Lucene[4] using its white space tokenizer together with a stop word filter. We used the LingPipe NLP API[5] to implement a machine learning approach with a language model classifier.

Before discussing the achieved results in detail, we introduce the data sets and the evaluation metrics, that have been used for our experiments.

6.1 Data Sets and Metrics

SCN. The *gold standard* for the SCN-scenario as described in the course of this paper was created by SAP experts. The experts created a ranked list of ECs, denoted as $\mathcal{U}_{\mathrm{n}}$, with $n_{\min} = 2$, $n_{\max} = 10$ and $\bar{n} \approx 3.6$ relevant software components of SAPTerm for 100 randomly chosen forum entries. Note, that the experts could discard forum entries that were not in their expertise.

CSS. We acquired a corpus of 7,700 real customer inquiries to address the routing of support messages within the *SAP Customer Support Service (CSS)* as described in Sec. 1. Support employees assigned a SAPTerm component for routing the inquiry to an expert. We took these assignments as a gold standard and considered only the problem description for EC identification. Because of the huge amount of tagged data, we compare to a machine learning approach [22] in this scenario. 10% of the documents were used as test corpus and the remaining 7,000 documents to train the Language Model based classifier. The training corpus consists of 5,000,000 words tagged with 1,500 components of SAPTerm. We removed documents from the test corpus that were assigned to components, which could not trained by at least 50 documents.

[4]`http://lucene.apache.org/`
[5]`http://alias-i.com/lingpipe`

REVIEW. For the movie-review search introduced in Sec. 1, we crawled 214 reviews of the most popular movies between 1920 and 1990[6]. These reviews were split into segments of 8 sentences, which allows to treat them as key word query for a web search engine. Finally, more than 1,000 distinct documents for evaluation were available. We removed the movie names at the beginning of the articles, but kept them for evaluation. We left other occurrences of movie names in the documents, e.g., that point to previous movies of actors. Finally, the *Internet Movie Database (IMDB)* was applied as knowledge base in this scenario. The publicly available web sites of IMDB, which provide exactly the same information as the raw IMDB data, allowed a performance comparison with web search engines. Web search engines interpret each EC as document, thus, compares to previous approaches, such as [33, 29, 1, 28, 3, 30] (further explanations follow).

Knowledge Bases. We evaluate rankings of SAPTerm components for *SCN* and *CSS*, and IMDB's movies for *REVIEW*.

We considered only the English version of SAPTerm, that contained 16,117 *components* with the term classes: *short name* and *long name*; 112,381 *technical terms* and 20,579 *variants*. Apart from the relations between variants, technical terms and components, SAPTerm contains 15,789 relations among components. The relations between *technical terms* were not further considered.

The IMDB was acquired via publicly available plain text exports and parsed into a RDBMS. The RDBMS contains 1,224,983 movie names and 263,542 alternative movie names (in the sense of "'also known as"'); 2,296,365 person names and 513,112 aka. person names; 2,075,439 character names and 194,057 company names. Apart from the relations between alternative and correct names for movies and persons we counted 16,421,791 n-to-m relations between movies, characters, and persons, 1,473,773 relations among companies and movies, and 785,818 relations between movies (e.g., 'remake of','spin off from', etc.). All these relationships were used for ranking ECs.

Evaluation Metrics. Our approach can be seen as a reverse IR task: Documents are queries and concepts of a structured vocabulary (usually the query) are results. Thus, evaluation can proceed along similar lines by comparing a user provided object list $\mathcal{U}_\mathrm{n}$ of length n to a ranked list $\mathcal{A}$ provided by the system. To quantify the intersection of both lists, we use slightly modified metrics of *Precision at* k (P_k) and *Recall at* k (R_k) [38] to evaluate *SCN*. Therefore, we compare the cropped user list $\mathcal{U}_{\arg\min(\mathrm{n},\mathrm{k})}$, simplyfied denoted as $\mathcal{U}_\mathrm{k}$, and system list $\mathcal{A}_\mathrm{k}$ at rank k. The metrics P_k and R_k are defined as $P_\mathrm{k} = |\mathcal{U}_\mathrm{k} \cap \mathcal{A}_\mathrm{k}|/k$ and $R_\mathrm{k} = |\mathcal{U}_\mathrm{k} \cap \mathcal{A}_\mathrm{k}|/n$. Thus, P_k captures the number of correctly identified ECs above k and R_k the ratio of expected ECs that were identified above k wrt. the complete user list $\mathcal{U}_\mathrm{n}$.

Three properties of these metrics have to be discussed. First, even a perfect system can not achieve $R_\mathrm{k} = 1$ if $n > k$. We will observe a constantly growing R_k with increasing k. Next, a system cannot achieve $P_\mathrm{k} = 1$ if $n < k$, leading to constantly decreasing P_k for $k > n$. Finally, the cropping of U_n leads to an increasing P_k within $k < n$, if the system provides a different ranking compared to the user's one. The described properties can be observed in *recall precision graphs (RCG)*, e.g. shown in Fig. 5(c). It depicts the average R_k and P_k for several documents. The left-most point of a series shows R_1 and P_1 and the right-most point R_{10} and P_{10}.

For *CSS* and *REVIEW* only the most relevant EC is available as gold standard. Here, we apply the *Success at k* measurement [15], denoted as S_k. It indicates whether the expected object ε was identified above rank k of $\mathcal{A}_\mathrm{k}$, leading to $S_\mathrm{k} = R_\mathrm{k}$ with $n = 1$.

[6]http://www.filmsite.org/allfilms2.html

6.2 Performance in Ranking ECs

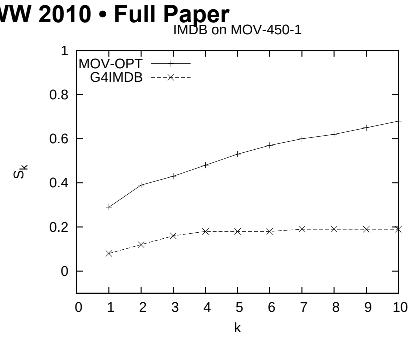
We use a standard parameter setting $p_\mathrm{out} = 0.2$, and $w_\mathrm{T} = 1$, $s = 0.7$ and $k = 250$ (the top-k limit for the term matcher), if not stated differently. First, we compare to a web search engine using the *REVIEW*-corpus, which compares to approaches like [33, 29, 1, 28, 3, 30]. Next, a machine learning approach [22] was implemented, that would be the first choice to identify ECs for the *CSS*-corpus, because of the large amount of available training data. Last, we discuss a comparison to [9, 40, 5] using *CSS*.

REVIEW. Our problem can be implemented by standard IR-approaches in treating each DCG as a mini document as proposed by [33]. A simple way to implement such approach, is to leverage a standard we search engine. We used Google, depicted as *G4IMDB* in Fig. 5(a), to implement EC-ranking for this scenario because [33] showed that Google outperforms [29, 1, 28] in ranking ECs for key word queries. Further, assuming that PageRank is one of the most important ranking functions of Google, we compare indirectly to approaches alike [3, 30]. Documents of *REVIEW* were sent as keyword queries to Google's webservice API with a restriction to IMDB's film homepages that correspond to ECs. We retrieved the list of ranked movie homepages including their titles, which were manually compared to the gold standard. Results of our approach are illustrated as *MOV-OPT* in Fig. 5(a).

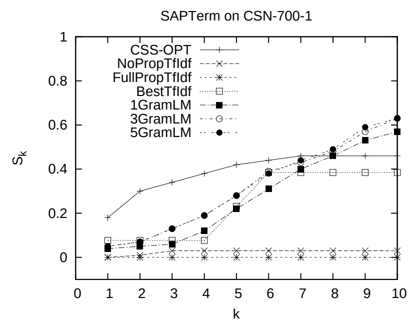
The results of this experiment, shown in Fig. 5(a), indicate that our ranking function outperforms *G4IMDB* for document processing. We achieve $\Delta S_1 \approx 0.21$ and $\Delta S_{10} \approx 0.58$. We assume that popularity ranking, e.g., encoded in PageRank, is an important performance factor for keyword search, but is inadequate when analyzing documents, that refer in an equally distributed manner to ECs.

CSS. We implemented an approach based on a generative Language Model [4] that would be the first choice to detect ECs in text if a large training corpus is available. In particular, we trained the token q-gram based language model by the 7,000 training documents as described above. We applied experiments for uni-, three- and 5-gram tokenizers (*1GramLM*, *2GramLM* and *5GramLM* in Fig. 5(b)). We realized that in *CSS* almost no component names were matched, thus a lower decay ($p_\mathrm{self} = 0.1$) was used, to gap longer distances between matches and ECs. Further, the matcher had to support a higher bandwidth of linguistic variations, thus we used $p_\mathrm{self} = 0.1$ allowing on average three edits for Levensthein. The results of those settings are depicted as *CSS-OPT*. Additionally, we show *FullPropTfIdf* and *NoPropTfIdf* that used TF-IDF for scoring individual matches. They use for aggregation $p_\mathrm{self} = 1$ and $p_\mathrm{self} = 0$ for aggregation. *FullPropTfIdf* is an adaption of EROCS [9], which uses the summed TF-IDF scores for ranking and treats the whole document as one segment (see Sec. 5). *NoPropTfIdf* interprets ECs as records and leverages only directly attached terms. Thus, it extends [40] by incorporating the frequency of matched terms. Further, we extended the EROCS like approach by a decay over distance and tested all possible parameterizations. A value of $p_\mathrm{self} = 0.3$ lead to the best results, illustrated as *BestTfIdf*.

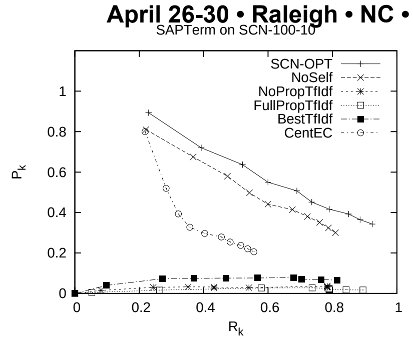
Fig. 5(b) summarizes evaluation results for *CSS*. First of all, we realize that *FullPropTfIdf* as well as *NoPropTfIdf* are not applicable to this data set, because neither the direct matched ECs, nor the most abstract ones lead to the correct results. Furthermore, we realize a performance difference of $\Delta S_1 \approx 0.14$ between *CSS-OPT* and *5GramLM*. The difference increases to $\Delta S_2 \approx 0.23$ and decreases for $k > 2$. For $k > 8$ the language model outperforms our approach. We assume that the reason behind this behavior is that the language model is only aware of 10% of the components in SAPTerm. *BestTfIdf* provided comparable results as the language model up to S_7. Considering the application, we show that our ap-



(a) S_{k} using REVIEW

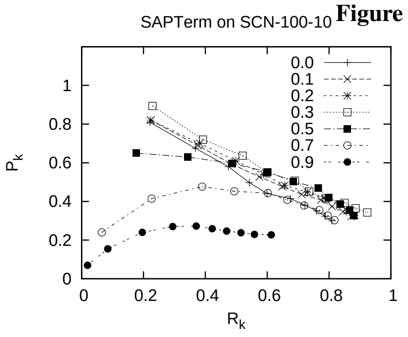


(b) S_{k} using CSS

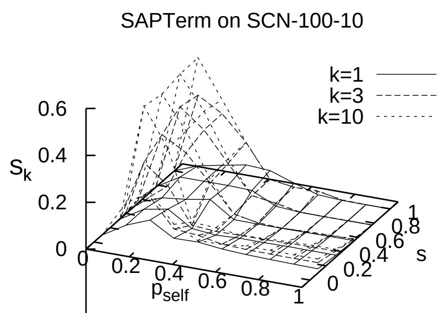


(c) RPG for P_{k} and R_{k} using SCN

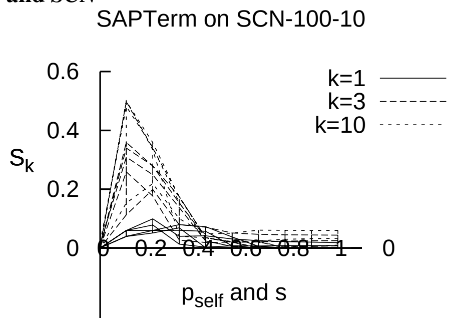
Figure 5: Performance evaluation for *REVIEW*, *CSS* and *SCN*



(a) SCN, varying p_{self}



(b) CSS, varying p_{self} and s



(c) Profile of Fig. 6(b)

Figure 6: Influence of parameter settings for *SCN* and *CSS*

proach provides a list of ECs for routing customer inquiries that contains the correct EC in one of two cases.

SCN. We used *FullPropTfIdf*, *NoPropTfIdf* and *BestTfIdf* to compare to TF-IDF based approaches. We found $p_{\mathrm{self}} = 0.7$ as best setting for *BestTfIdf*. We implemented additionally *CentEC* as a variant of our approach using a score aggregation function inspired by BANKS [5], which considers structural compact graphs. It interprets each EC as the central node of a possible result and sums scores of neighboring ECs in the DCG (1-Star Tree). Match and edge weights were computed with the standard settings as described above and $p_{\mathrm{self}} = 0.3$, which turned out to produce the best results for this approach. The experiment *NoSelf* uses the weighting described in the course of this paper, but runs with $p_{\mathrm{self}} = 0$. The results of our approach using $p_{\mathrm{self}} = 0.3$ are depicted as *SCN-OPT*.

Fig. 5(c) summarizes evaluation results for *SCN*. First of all, we found that *NoPropTfIdf* performed badly with a maximum precision at $P_5 = 0.032$, but achieves a good recall $R_{10} \approx 0.8$. The EROCS like approach *NoPropTfIdf* achieves same recall level as *NoPropTfIdf* at $k = 6$, which ends up in $P_{10} \approx 0.9$. *BestTfIdf* produces a better precision between $2 < k < 8$ of approximately 0.08. The low precision arises because the top scored term matches are often not relevant. Thus, the relevant ECs are mostly ranked at an upper k. *SCN-OPT* outperforms *CentEC* and *NoSelf* by $\Delta P_1 \approx 0.10$. The performance of *CentEC* drops significantly for $P_{k>1}$ to $\Delta P_5 \approx 0.20$ compared to *SCN-OPT*. Thus, objects which are ranked lower in $\mathcal{U}_{\mathrm{n}}$ are often dropped. Further, *SCN-OPT* outperforms *CentEC* in recall by $\Delta R_5 \approx 0.29$ and $\Delta R_{10} \approx 0.37$, and *NoSelf* by $\Delta R_5 \approx 0.10$ and $\Delta R_{10} \approx 0.15$. The good P_1 and R_1, but rather low R_{10} and P_{10} of *CentEC* indicate that structural compact graphs perform well in identifying the most relevant result, e.g. important for keyword searches, but not for scenarios, where several ECs have to be identified. The comparison of *SCN-OPT* and *NoSelf* shows that a proper setting for p_{self} constantly helps improving recall and precision, here, by 10 to 15 percentage points.

6.3 Parameter Settings

The trade-off between qualitative performance and processing time for *top k* retrieval has been studied intensively (see, e.g., [10, 14, 50]), thus, we do not investigate the parameter k. Furthermore, we found $p_{\mathrm{out}} = 0.2$ to perform well in all experiments and used $w_{\mathrm{T}} = 1$ for simplification. We discuss the influence of the parameter p_{self} for *SCN* and *CSS*. Next, we evaluate the influence of the similarity threshold s for *CSS*.

Impact of decay factor. The parameter for p_{self} has a major influence on recall and precision of our approach because different scenarios address different abstraction levels. A rough heuristic to determine a value for p_{self} is the usage of the reciprocal average depth of the ontology, that is $p_{\mathrm{self}} = 1/4.9 \approx 0.2$ for SAPTerm. Fig. 6(a) shows the results for *SCN* for varying p_{self} with respect to R_k and P_k. We identified a maximum at $p_{self} = 0.3$ with $P_1 = 0.9$ that slightly outperforms $p_{self} = 0.2$ with $P_1 = 0.83$. We show the results for varying p_{self} for *CSS* in Fig. 6(b) and Fig. 6(c). Here, we realize a better performance for $p_{self} = 0.1$ that outperforms the estimated value $p_{self} = 0.2$ for upper k. For $k = 1$ indeed $p_{self} = 0.2$ would be the best setting in combination with $s = 0.1$.

Impact of similarity threshold. We use *CSS* to show the influence of s. We realized that in general a low s produces better results for S_1, but does not impact the results for S_{10}. In particular, a value of $s = 0.1$ leads to $S_1 \approx 18\%$ and $s = 0.7$ leads to $S_1 \approx 6\%$ in this experiment. However, $s = 0.7$ lowers the number of nodes in a DCG from 5,000 to 1,000 nodes compared to $s = 0.1$. Thus, high values of s lead to significantly better document processing times. In practice, we apply $s = 0.1$ because support employees prefer a better ranking over processing time.

7. CONCLUSION

In this paper, we addressed the problem of identifying complex enterprise concepts (EC) in textual documents. Since the ECs are predefined and available in an enterprise ontology, the essence of this problem is to first recognize potential ECs and to map them to the ontology. Both steps are difficult due to the nature of the documents we consider (non grammatical, typos, frequent use of abbreviations and jargon, etc.) and due to the high ambiguity of ECs and the words they are formed of. We presented solutions to all these problems based on a scoring scheme that considers the *confidence* and the *relevance* of each potential match. In particular, we showed that a careful combination of the evidence of multiple potential matches within a text, together with an exploitation of the logical structure defined by the ontology helps to boost the performance of EC recognition considerably.

There are multiple ways of further improving our approach. First, to be applicable in heavily loaded online forums (like SCN), we need to improve execution speed, we expect to be not too difficult because many of the calculations are repetitive. Furthermore, we plan to combine our approach with other document processing techniques, such as document classification and topic detection to estimate edge weights of the knowledge base on an individual basis.

Acknowledgments

This work is supported by the FP7 EU Large-scale Integrating Project "‘OKKAM - Enabling a Web of Entities"’[7] (contract no. ICT-215032). We like to thank members of the Data Management and Analytics team at SAP Research, especially the former colleagues Alexander Löser and Hong-Hai Do, for valuable discussions.

8. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proc. ICDE 2002*.

[2] E. Amitay, N. Har’El, R. Sivan, and A. Soffer. Web-a-where: geotagging web content. In *Proc. SIGIR 2004*.

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *Proc. VLDB 2004*.

[4] K. Balog, L. Azzopardi, and M. de Rijke. Formal models for expert finding in enterprise corpora. In *Proc. SIGIR 2006*.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *Proc. ICDE 2002*.

[6] C. Binnig, S. Hildenbrand, and F. Färber. Dictionary-based order-preserving string compression for main memory column stores. In *Proc. SIGMOD 2009*.

[7] F. Brauer, W. Barczynski, G. Hackenbroich, M. Schramm, and A. Mocan. RankIE: Document Retrieval on Ranked Entity Graphs. In *Proc. VLDB 2009 (Demo Track)*.

[8] A. Z. Broder and A. C. Ciccolo. Towards the next generation of enterprise search technology. *IBM Syst. J.*, 43(3):451–454, 2004.

[9] V. T. Chakaravarthy, H. Gupta, P. Roy, and M. Mohania. Efficiently linking text documents with relevant structured information. In *Proc. VLDB 2006*.

[10] A. Chandel, P. C. Nagesh, and S. Sarawagi. Efficient Batch Top-k Search for Dictionary-based Entity Recognition. In *Proc. ICDE 2006*.

[11] S. Chaudhuri, V. Ganti, and R. Kaushik. A Primitive Operator for Similarity Joins in Data Cleaning. In *Proc. ICDE 2006*.

[12] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *Proc. SIGMOD 2006*.

[13] T. Cheng, X. Yan, and K. C.-C. Chang. EntityRank: searching entities directly and holistically. In *Proc. VLDB 2007*.

[14] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A Comparison of String Metrics for Matching Names and Records. In *KDD Workshop on Data Cleaning and Object Consolidation*, 2003.

[15] N. Craswell and D. Hawking. Overview of the TREC 2004 Web Track. In E. M. Voorhees and L. P. Buckland, editors, *TREC*, volume Special Publication 500-261. National Institute of Standards and Technology (NIST), 2004.

[16] S. Cucerzan. Large-scale named entity disambiguation based on Wikipedia data. In *Proc. of EMNLP-CoNLL*, 2007.

[17] J. L. G. Dietz. *Enterprise Ontology: Theory and Methodology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[18] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. Tomlin, et al. SemTag and Seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proc. WWW 2003*.

[19] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proc. CIKM 2004*.

[20] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *Proc. WWW 2001*.

[21] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate Record Detection: A Survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[22] H. Fang and C. Zhai. Probabilistic Models for Expert Finding. In *Proc. ECIR 2007*.

[23] F. Farfán, V. Hristidis, A. Ranganathan, and M. Weiner. XOntoRank: Ontology-Aware Search of Electronic Medical Records. In *Proc. ICDE 2009*.

[24] S. Gaudan, A. J. Yepes, V. Lee, and D. Rebholz-Schuhmann. Combining evidence, specificity, and proximity towards the normalization of gene ontology terms in text. *EURASIP J. Bioinformatics Syst. Biol.*, pages 1–9, 2008.

[25] J. Gonzalo, F. Verdejo, I. Chugur, and J. Cigarran. Indexing with WordNet synsets can improve text retrieval. *Arxiv preprint cmp-lg/9808002*, 1998.

[26] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: ranked keyword search over XML documents. In *Proc. SIGMOD ’03*.

[27] J. Hassell, B. Aleman-Meza, and I. B. Arpinar. Ontology-Driven Automatic Entity Disambiguation in Unstructured Text. In *Proc. ISWC 2006*.

[28] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In *Proc. VLDB 2003*.

[29] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *Proc. VLDB 2002*.

[30] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proc. VLDB 2005*.

[31] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and Ranking Knowledge. In *Proc. ICDE 2008*.

[32] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proc. SIGMOD 2008*.

[33] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *Proc. SIGMOD 2006*.

[34] A. Löser, W. M. Barczynski, and F. Brauer. What’s the Intention Behind Your Query? A few Observations From a Large Developer Community. In *Proc. IRSW 2008*.

[35] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *Proc. SIGMOD 2007*.

[36] R. Mandala, T. Takenobu, and T. Hozumi. The use of WordNet in information retrieval. In *Use of WordNet in Natural Language Processing Systems: Proceedings of the Conference*, 1998.

[37] C. Mangold, H. Schwarz, and B. Mitschang. u38: A framework for database-supported enterprise document-retrieval. In *Proc. IDEAS 2006*, 2006.

[38] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[39] K. S. McCurley. Geospatial mapping and navigation of the web. In *Proc WWW 2001*.

[40] M. Michelson and C. A. Knoblock. Unsupervised information extraction from unstructured, ungrammatical data sources on the World Wide Web. *Int. J. Doc. Anal. Recognit.*, 10(3):211–226, 2007.

[41] K. Muthmann, A. Loeser, W. Barczynski, and F. Brauer. Near-Duplicate Detection for Web-Forums. In *Proc. IDEAS 2009*.

[42] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1), 2001.

[43] R. Navigli and P. Velardi. An analysis of ontology-based query expansion strategies. In *Workshop on Adaptive Text Extraction and Mining*, 2003.

[44] J. K. Owyang, S. VanBoskirk, S. Glass, C. S. Overby, G. O. Young, and A. Polanco. The Forrester Wave: Community Platforms, Q1 2009. Forrester Wave (white paper), 2009.

[45] S. Puhlmann, M. Weis, and F. Naumann. XML Duplicate Detection Using Sorted Neighborhoods. In *Proc. EDBT 2006*.

[46] R. Richardson and A. Smeaton. Using WordNet in a knowledge-based approach to information retrieval. In *Proceedings of the BCS-IRSG Colloquium, Crewe*, 1995.

[47] C. Rocha, D. Schwabe, and M. P. Aragao. A hybrid approach for searching in the semantic web. In *Proc. WWW 2004*.

[48] E. F. T. K. Sang. Memory-based shallow parsing. *J. Mach. Learn. Res.*, 2:559–594, 2002.

[49] S. Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[50] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proc. VLDB 2004*.

[51] Y. Tsuruoka and J. ichi Tsujii. Improving the performance of dictionary-based approaches in protein name recognition. *Journal of Biomedical Informatics*, 37(6), 2004.

[52] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient approximate entity extraction with edit distance constraints. In *Proc. SIGMOD 2009*.

[53] X. Yang, C. M. Procopiuc, and D. Srivastava. Summarizing Relational Databases. *Proc. VLDB 2009*.

[54] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. SPARK: Adapting Keyword Query to Semantic Search. In *Proc. ISWC/ASWC 2007*.

[7] `http://www.okkam.org`