Understanding Queries in a Search Database System

Ronald Fagin Benny Kimelfeld Yunyao Li Sriram Raghavan Shivakumar Vaithyanathan
IBM Research–Almaden
San Jose, CA 95120, USA
fagin@almaden.ibm.com, {kimelfeld, yunyaoli, rsriram, shiv}@us.ibm.com

ABSTRACT

It is well known that a search engine can significantly benefit from an auxiliary database, which can suggest interpretations of the search query by means of the involved concepts and their interrelationship. The difficulty is to translate abstract notions like *concept* and *interpretation* into a concrete search algorithm that operates over the auxiliary database. To surpass existing heuristics, there is a need for a formal basis, which is realized in this paper through the framework of a *search database system*, where an interpretation is identified as a *parse*. It is shown that the parses of a query can be generated in polynomial time in the combined size of the input and the output, even if parses are restricted to those having a nonempty evaluation. Identifying that one parse is more specific than another is important for ranking answers, and this framework captures the precise semantics of being *more specific*; moreover, performing this comparison between parses is tractable. Lastly, the paper studies the problem of finding the most specific parses. Unfortunately, this problem turns out to be intractable in the general case. However, under reasonable assumptions, the parses can be enumerated in an order of decreasing specificity, with polynomial delay and polynomial space.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms

Algorithms, Design, Theory

Keywords

Search database system, auxiliary database

PODS'10, June 6–11, 2010, Indianapolis, Indiana, USA.

1. INTRODUCTION

It has long been recognized that the quality of results in an information-retrieval (IR) system can be significantly improved by exploiting techniques from the area of information extraction (IE) [3, 4, 9, 13, 19, 26, 28, 30]. Intuitively, IE enables the search system to associate, with each document, not just a sequence of constituent words ("tokens"), but also semantically richer information in the form of entities, relationships, sentiments, and opinions. For example, IE applied to a collection of email messages can identify commonly occurring concepts such as persons, addresses, phone numbers, and associated relationships [13]. Similarly, when applied to product reviews, IE techniques can extract mentions of specific product features and associated positive or negative sentiments expressed by the author of the review. Several different approaches have been studied for exploiting such information to improve search quality. One approach is to use IE as a post-processing step to filter and reorder the results from a retrieval engine [3,9,23]. Another approach is to use this information for smarter indexing [19,20] or query expansion [26]. However, all of these approaches apply *indirect* means of using the extracted information to improve search, as they do not directly affect the core of the retrieval engine.

In recent work [13, 30], focused on personal email search and on intranet search, we adopted a more direct approach of using IE for "understanding" search queries. To illustrate the potential of this approach, consider the search query that contains the single word *avi*. On the Web, it is expected to rank highest a page that discusses the AVI multimedia format. In an enterprise intranet search, we can use IE techniques to recognize that Avi is a shorthand for the person name Avigdor, who is mentioned in the underlying corpus as an employee of the company; thus, we may choose to explicitly *interpret* the keyword query *avi* as the query "return personal home pages of people whose name matches *avi* (e.g., Avigdor)."

While our work [13, 30] was originally motivated by auxiliary databases populated through IE, the overall approach applies even if the auxiliary database is obtained from available data underlying the application, or a combination of the two. As an example of available data, consider the query *buzz price* posed to the search engine of an online toy store. A database storing the business logic can indicate that *buzz* matches the product "Buzz Lightyear," and that *price* is a known concept (that can appear as, e.g., "$49.99"). Thus, the engine may decide to bring back, as top search results, pages containing the aforementioned product and a price

(but not necessarily the word "price"). Even better, recognizing that *price* is a property of a product, the aim can be for pages where Buzz Lightyear and the price are associated (i.e., Buzz Lightyear's price is mentioned, as in a page with the commercial "Buzz Lightyear—now on sale: $49.99").

The above discussion illustrates one principle: an auxiliary database of concepts associated with the corpus can help in understanding the user intent behind the search query, and thereby significantly improve search. In contrast to other published solutions, our approach is direct in the sense that we actually apply a preprocessing step that suggests interpretations of the query. Of course, the notion of *interpretation* is vague (so far). Roughly, in the search systems we have implemented at IBM Almaden (which perform quite well in practice [13, 30]), suggesting an interpretation means applying heuristics that detect opportunities for associating keywords with database content. But to make a significant progress in our search solutions, we needed to establish a principled framework that formally captures the task of understanding queries, and in particular that models and relates the notions of *auxiliary databases*, *concepts* (e.g., "product" and "price"), *relationships* between concepts (e.g., "product-price"), and *interpretations* (e.g., *buzz price* refers to Buzz Lightyear's price). A major difficulty in establishing such a framework is that this framework needs to capture the core principles underlying an involved commercial product, and at the same time be simple and elegant enough to allow one to gain insights and develop algorithmic solutions. We take this challenge in this paper. To place this work in context, observe that the task of understanding queries is one component out of many (e.g., rankers, indices, crawlers, IE, etc.) comprising a complete search engine.

Our framework is encapsulated in what we call a *search database system* (*SDBS*). The basic construct in an SDBS is a *concept*, and there are two types of concepts: *atomic* concepts (e.g., product and price) and *compound* concepts (e.g., productPrice, which has the subconcepts product and price). A *schema* is a collection of concepts with an acyclic nesting relationship. A *database instance* over a schema is a collection of records, where each record is associated with a URI (i.e., a unique resource identifier, where a *resource* can be a Web page, a document, an email message, etc.) and with a concept of the schema; when that concept is atomic, the record has textual content, and when the concept is compound, the record comprises references to other records (that correspond to the subconcepts of the compound concept).

In our framework, interpretations are derived from *grammar rules*, which can be composed manually by a developer, possibly with the aid of automatic techniques such as grammar learning [24] and with heuristics that utilize the schema. In these rules, a nonterminal is a concept, and a terminal is either a keyword or an *instantiation* of an atomic concept by means of a keyword match. For illustration, suppose that the compound concept productPrice has the two subconcepts product and price. The rule "productPrice → product *price*" suggests that to refer to a product-price relationship, the user could describe a product (by applying a rule for product) followed by the word *price*. Here, productPrice and product serve as nonterminals, and *price* is a terminal, which is a keyword appearing in the search query. Next, the rule "product → ⌈product⌉" says that in a query, a product is referred to as a string (e.g., *buzz* or *buzz lightyear*) matching a specific product in the database (e.g., the toy Buzz Lightyear), as indicated by the instantiation ⌈product⌉. The two rules allow us to interpret "*buzz price*" as "find pages containing a product matching *buzz* and its price." Formally, we identify an *interpretation* as a hedge (ordered forest) of parse trees (generated by the grammar), which we refer to as a *parse*. An SDBS comprises a schema, a database instance, and a grammar. Since both the instance and the grammar are defined by means of the schema concepts, we can view a parse as a structured query that evaluates to the matching database URIs; thus, the semantics of *interpretation* is captured.

We study the computational problem of generating the *data-nonempty* parses, which are the parses that have at least one answer in the database instance. If one makes assumptions that upper bound by a constant the length of the search query and the depth of the schema, then generating the data-nonempty parses in polynomial time is straightforward. Without these assumptions, the number of data-nonempty parses can be exponential; thus no polynomial-time algorithm can produce all of them. Nevertheless, we show (with no assumptions) that the data-nonempty parses can be generated in time that is polynomial in the combined size of the input and the output. To accommodate the possibility of a large number of data-nonempty parses, later in the paper we revisit this problem, where the goal is to produce the parses incrementally, in a specific order.

A related problem is that of *keyword proximity search* (KPS) in structured (e.g., relational and XML) databases [1, 8, 10, 12, 15, 21, 25].[1] Both in our problem and in KPS, the goal is to answer a search query based on database content. A technical difference is that in KPS, answers are connected database portions containing the keywords, whereas here an answer is a URI. Some of the solutions proposed for KPS are based on generating interpretations (queries) from the schema [10, 21, 25],[2] and that approach could, potentially, be applied here for generating parses. Generation of interpretations in KPS is generally based on *connectivity* of query items, which has the advantage that a grammar is not needed. However, compared to our grammar-based approach, it severely limits the control over the allowed interpretations. Furthermore, it is not clear how that approach can take advantage of highly important grammatical parts of the query (e.g., "of" in *price of buzz*, and "from" in *from avigdor* posed in email search), which our approach handles gracefully. Furthermore, to our knowledge, no known query generator in KPS guarantees both efficiency and data-nonemptiness of the generated parses. Lastly, the aspects we discuss next do not arise in KPS.

From our experience with the needs of search, we learned that a central property of a parse is its *specificity*. As an example, consider again the query *buzz price*. Different parses can interpret the query as follows: (1) occurrence of the toy Buzz Lightyear and some price (not necessarily that of Buzz), and (2) occurrence of Buzz Lightyear along with its price. The second interpretation is more specific, as it ties the two terms of the query in a strictly tighter manner. It is reasonable to decide that interpretation (2) positively affects the rank of the page. As another example, consider the query "*Sara Enron*" posed in the context of enterprise search. Different parses can give interpretations such as (1)

[1]See [5] for a comprehensive survey of research on KPS.
[2]Other solutions [1, 8, 12, 15] operate directly on the database.

occurrence of the person Sara and the company Enron, and (2) specification that Sara is an Enron employee (e.g., as part of a signature). Interpretation (2) is more specific than (1), since an employee is a special case of a person (and (2) has the extra works-for relationship). This illustrates that higher specificity implies a stronger connection to the concepts of interest, and/or a tighter connection between the keywords.

By adopting the traditional notion of *query containment*, our SDBS framework gives a natural way of comparing the specificity of two parses: a parse p is at least as specific as the parse p' if for all instances over the schema, every answer for p is also an answer for p'. We would like our notion of containment to take into account the fact that one concept can specialize another (e.g., employee specializes person). So, we extend our framework by letting the schema specify a preorder (i.e., a reflexive and transitive relationship) over its atomic concepts, and require database instances to respect this preorder (e.g., every employee is indeed a person).

Some delicate issues arise from the fact that a parse is mapped to a database by means of (possibly fuzzy) keyword matching. For example, the string *java beans* does not necessarily match a record with the text *java* in the context of programming (since the user typing *java beans* refers to the specific Java-Bean construct rather than the general Java language), but it makes perfect sense that *java beans* matches *java* as a food item. Thus, keyword matching depends on the concept under which it is made. As another example, consider the query *fisher price* posed in an online toy store. Consider two parses, where in the first *fisher price* is a brand, and in the second, *fisher* is a brand and *price* is the price concept. If we consider there to be a match for *fisher price* whenever there is a match for *fisher* (which is not necessarily true in practice), then the second parse is more specific than the first. The last example is the query *avi from avigdor* in the context of email search. Suppose that we have two parses that interpret the query as an email from Avigdor, but in the first *avi* refers to a person whereas in the second *avi* refers to the format of an attachment. One would think that the two parses are incomparable in terms of specificity. However, the second interpretation is actually more specific than the first, since the fact that Avigdor is the sender already implies that the email contains a person that matches *avi* (namely Avigdor), assuming that sender specializes person. The above examples show some of the subtleties arising from keyword matching, especially when combined with specificity.

We study the problem of deciding, given two parses p_1 and p_2, whether p_1 is contained in (i.e., at least as specific as) p_2. For that, we give a structural characterization of containment, and show how it translates into a polynomial-time decision on containment. Our approach is in the spirit of testing containment among *tree-pattern queries* in XML [2] (namely, by means of homomorphism), but our algorithm is more involved due to the specific properties of our framework. In particular, in tree-pattern containment there is no correspondence to the fact that parses involve keyword matches, or to the fact that concepts are interconnected by the subconcept and the specialization relationships.

Due to the potential existence of many (data-nonempty) parses for a query, one is likely to favor an incremental generation of parses, in some order of desirability, as opposed to an arbitrary order. The exact meaning of *desirability* varies in different contexts. However, based on our experience with deploying search systems in multiple domains, and working with real users, we have observed that users typically intend interpretations that most strongly associate the keywords constituting the query. Indeed, this has been our basic motivation to formalizing and studying the notion of specificity. So, we consider the problem of enumerating the data-nonempty parses by decreasing specificity. Formally, the goal is to enumerate the data-nonempty parses, where for two parses p_1 and p_2, if p_1 is strictly contained in p_2, then p_1 should be generated prior to p_2. The enumeration should be *incremental*, which in particular means that k most specific parses should be generated in polynomial time in the input and in k. Unfortunately, this turns out to be intractable: no such enumeration exists unless P = NP. Nevertheless, we show that under an assumption that, roughly, restricts the ambiguity of the search query, enumeration in the order of decreasing specificity can be done with *polynomial delay* [11] and polynomial space.

2. SEARCH DATABASE SYSTEMS

In this section, we formalize our framework of a *search database system*. We begin with some preliminaries.

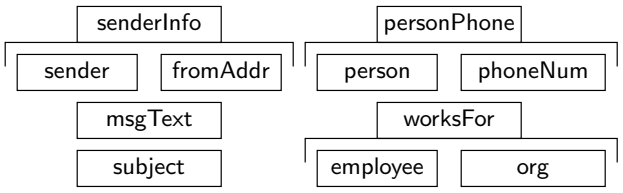
2.1 Preliminaries: Strings, Trees and Hedges

Let Σ be a set, which we refer to as an *alphabet*. We denote by Σ^* the set of all strings over Σ, that is, all the finite sequences $\sigma_1 \cdots \sigma_n$, where $\sigma_i \in \Sigma$ for $1 \leq i \leq n$. The length n of a string $\mathbf{s} = \sigma_1 \cdots \sigma_n$ is denoted by $|\mathbf{s}|$. By Σ^+ we denote the set $\Sigma^* \setminus \{\epsilon\}$, where ϵ is the empty string. Given two strings $\mathbf{s}_1, \mathbf{s}_2 \in \Sigma^*$, the string $\mathbf{s}_1\mathbf{s}_2$ is the concatenation of $\mathbf{s}_1$ and $\mathbf{s}_2$.

The trees we use in this paper are directed and ordered (i.e., the children of each node form a sequence), where each node is assigned a *label*. A *hedge* is a sequence $h = t_1 \cdots t_n$ of (pairwise node-disjoint) trees; in other words, it is a forest where the trees are linearly ordered. Note that a tree is a special case of a hedge. We denote by nodes(h) the set of nodes of the hedge h. If (v, w) is an edge of h, then w is a *child* of v, which in turn is the *parent* of w. The label of the node v is denoted by label(v). A hedge h is *over* an alphabet Σ if the label of every node belongs to Σ. A *leaf* of a hedge is a node without children, and a non-leaf node is an *internal node*. We use leaves(h) and internal(h) to denote the sets of leaf and internal nodes of h, respectively. If a hedge h contains a directed path from node v to node w, then w is a *descendant* of v, which in turn is an *ancestor* of w. For a tree t, the *root* of t, denoted root(t), is the unique node without a parent.

2.2 Concepts, Schemas and Instances

A *schema* is a collection of *concepts* with acyclic nesting relationships. Formally, a schema $\mathcal{S}$ is a finite set of *concepts*, where each concept $\gamma \in \mathcal{S}$ is associated with a (possibly empty) set of *subconcepts* $\delta \in \mathcal{S}$, and we denote by $\mathsf{sc}(\gamma)$ the set of subconcepts of γ. We require $\mathcal{S}$ to be *acyclic*, in the following sense. There is no cycle in the directed graph that has $\mathcal{S}$ as its set of nodes, and an edge (γ, δ) whenever $\delta \in \mathsf{sc}(\gamma)$. A concept $\gamma \in \mathcal{S}$ such that $\mathsf{sc}(\gamma) = \emptyset$ is called an *atomic concept*, abbreviated *a-concept*. If $\gamma \in \mathcal{S}$ is such that $\mathsf{sc}(\gamma) \neq \emptyset$, then γ is a *compound concept*, abbreviated *c-concept*. The subset of $\mathcal{S}$ containing all the a-concepts is



(a) Schema $\mathcal{S}$

rec.	concept	URI	text
r_{sn}	sender	em1	*Sara Shackleton*
r_{ad}	fromAddr	em1	*sara@enron.com*
r_{sb}	subject	em1	*Luisiana-Pacific("LP")*
r_{mt}	msgText	em1	*****
r_{pr1}	person	em1	*John*
r_{pr2}	person	em1	*Sara Shackleton*
r_{ph}	phoneNum	em1	*713-853-5620*
r_{em}	employee	em1	*Sara Shackleton*
r_{org}	org	em1	*Enron North America Corp.*
s_{ad}	fromAddr	em2	*john@enron.com*
s_{sb}	subject	em2	*RE: Profile Error Again*
s_{mt}	msgText	em2	*****
s_{pr}	person	em2	*John Oh*
s_{ph}	phoneNum	em2	*503-464-5066*
s_{em}	employee	em2	*John Oh*
s_{org}	org	em1	*Enron North America*

(b) The a-records of the instance I

rec.	concept	URI	references
r_{si}	senderInfo	em1	$r_{\text{si}}[\text{sender}] = r_{\text{sn}}$, $r_{\text{si}}[\text{fromAddr}] = r_{\text{ad}}$
r_{pp}	personPhone	em1	$r_{\text{pp}}[\text{person}] = r_{\text{pr2}}$, $r_{\text{pp}}[\text{phoneNum}] = r_{\text{ph}}$
r_{wf}	worksFor	em1	$r_{\text{wf}}[\text{employee}] = r_{\text{em}}$, $r_{\text{wf}}[\text{org}] = r_{\text{org}}$
s_{pp}	personPhone	em2	$s_{\text{pp}}[\text{person}] = s_{\text{pr}}$, $s_{\text{pp}}[\text{phoneNum}] = s_{\text{ph}}$
s_{wf}	worksFor	em2	$s_{\text{wf}}[\text{employee}] = s_{\text{em}}$, $s_{\text{wf}}[\text{org}] = s_{\text{org}}$

(c) The c-records of the instance I

Luisiana−Pacific (''LP'')

From: Sara Shackleton ⟨sara@enron.com⟩
To: John Malowney, Jason Williams
Sent: 05/16/2001 at 15:32

John:

LP executed our ISDA on May 7. No need to discuss your email. Thanks.

em1

Sara Shackleton
Enron North America Corp.
1400 Smith Street, EB 3801a
Houston, Texas 77002
713-853-5620
sara@enron.com

RE: Profile Error Again

From: ⟨john@enron.com⟩
To: David Steiner
Sent: 01/31/2002 at 18:59

Realtime group—
This occurs when your H drive is not mapped. To fix the mapping, follow the instructions at:
. . .
Hope this helps!

em2

John Oh
Enron North America
121 SW Salmon Street
Portland, OR 97204
john@enron.com
503.464.5066

(d) Email messages em1 and em2

Figure 1: A schema $\mathcal{S}$, an instance I, and email messages em1 and em2

denoted by $\mathcal{S}^{\text{a}}$. Similarly, the subset of $\mathcal{S}$ containing all the c-concepts is denoted by $\mathcal{S}^{\text{c}}$.

An example of a schema is the set $\mathcal{S} = \{\mathsf{person}, \mathsf{car}, \mathsf{make}, \mathsf{model}, \mathsf{carOwner}\}$, where person, make and model are the a-concepts (thus, the set $\mathsf{sc}(\gamma)$ is empty for each of these three concepts γ), and car and carOwner are the c-concepts with $\mathsf{sc}(\mathsf{car}) = \{\mathsf{make}, \mathsf{model}\}$ and $\mathsf{sc}(\mathsf{carOwner}) = \{\mathsf{person}, \mathsf{car}\}$. Another example of a schema is given next.

Example 2.1. Our running example is an email-search system, and Figure 1(a) depicts a schema $\mathcal{S}$ for this system. The a-concepts are sender, fromAddr, person, phoneNum, msgText, subject, employee and org. The c-concepts are senderInfo, personPhone, and worksFor, where

- $\mathsf{sc}(\mathsf{senderInfo}) = \{\mathsf{sender}, \mathsf{fromAddr}\}$,
- $\mathsf{sc}(\mathsf{personPhone}) = \{\mathsf{person}, \mathsf{phoneNum}\}$, and
- $\mathsf{sc}(\mathsf{worksFor}) = \{\mathsf{employee}, \mathsf{org}\}$.

The schema $\mathcal{S}$ is very simple compared to the general definition of a schema: First, the nesting level in $\mathcal{S}$ is 1 (i.e., the subconcepts of a c-concept are all atomic). Second, no two c-concepts of $\mathcal{S}$ share the same subconcept. □

Next, we define the notion of a *database instance* over a schema. Throughout the paper, we fix two sets. First, we have a set $\mathbf{U}$ of URIs (i.e., Unique Resource Identifiers, which are used as document identifiers). Second, we have a set $\mathbf{T}$ of *terms*; a term can appear in a document, and it can be used as a keyword in a search query. For the sake of complexity analysis (which is done later), we assume that both $\mathbf{U}$ and $\mathbf{T}$ are infinite sets.

Let $\mathcal{S}$ be a schema. A *database instance* (or just *instance*) I over $\mathcal{S}$ is a finite set of *records*, where each record r is associated with a unique concept $\gamma \in \mathcal{S}$ (thus, γ acts as a property of r). We write r^γ (instead of just r) to denote that r is associated with γ (so, the record r can be referred to as either r or r^γ). Every record r has a URI, which is denoted by $\text{uri}(r)$. As for the rest of the content of a record, we distinguish between two types of records r^γ, depending on whether γ is atomic or not.

- If γ is an a-concept (i.e., $\gamma \in \mathcal{S}^{\text{a}}$), then r^γ is called an *a-record*. Then, r has textual data, which is a string of $\mathbf{T}^*$. The textual data of r is denoted by $\mathit{txt}(r)$.
- If γ is a c-concept (i.e., $\gamma \in \mathcal{S}^{\text{c}}$), then r^γ is called a *c-record*. Then, for each subconcept $\delta \in \mathsf{sc}(\gamma)$, the record r^γ references a record $s^\delta \in I$ with the same URI (that is, $\text{uri}(r) = \text{uri}(s)$); the record s is denoted by $r[\delta]$.

Example 2.2. The two tables in Figure 1 show an instance I over the schema $\mathcal{S}$ (discussed in Example 2.1). The top table (Figure 1(b)) contains the a-records and the bottom one (Figure 1(c)) contains the c-records. This instance represents two email messages,[3] which are depicted in Figure 1(d), with the URIs em1 and em2. Each row in the table describes one record. For example, the first row of the top table describes the a-record r_{sn}, and the first row of the bottom table describes the c-record r_{si}. The column "concept" gives the concept of the record. For example, the concept of r_{sn} is sender. The column "URI" gives the URI of the record; for example, $\text{uri}(r_{\text{sn}}) = \texttt{em1}$. In the top table, the column "text" shows the textual data (except for the "***" entries that

[3]These messages are from the Enron corpus [16], with details omitted or slightly edited to meet space limitations.

we discuss later); for instance, $txt(r_{\text{sn}})$ = *Sara Shackleton* (which consists of the two terms *Sara* and *Shackleton*). In the bottom table, the column "references" shows the reference for each subconcept.

Some of the records of I can be produced by a straightforward parsing of the email message (according to the email protocol in use). This is the case for the records of the concepts sender, fromAddr, senderInfo and subject, which are obtained from the top parts of the email messages of Figure 1(d). This is also the case for msgText, which contains the whole body of the email and is replaced in the table with "***" (in order to fit the figure). Other records, however, are not explicitly specified in an email message, and producing those may require more complicated (possibly heuristic) techniques. For example, the person records are obtained by identifying a person name in the message text. Similarly, producing the phoneNum records requires a mechanism for identifying that a substring of the text is a phone number, and producing a personPhone record entails identifying that a person and a phone number recognized in the message text are semantically associated. In Figure 1(d), we underlined the parts of the message that were used to extract the records. For example, in the email `em1`, the name "Sara Shackleton" is extracted as a person (r_{pr2}) as well as employee (r_{em}). These are typical tasks in *information extraction* [17, 27], which we view as a primary tool for the construction of an SDBS. Observe that the example contains only a part of the relevant information that can be extracted from the email. For example, the recipients of the emails are ignored here. □

2.3 Search Queries and Grammars

A *search query* is a nonempty string of terms, that is, a string $\mathbf{q}$ of $\mathbf{T}^+$. Every term of a search query is called a *keyword*. We denote by $kw(\mathbf{q})$ the set of all the keywords of $\mathbf{q}$ (thus, $\mathbf{q}$ is a string, whereas $kw(\mathbf{q})$ is a set). We assign a possible *meaning* to a search query, in the context of a schema, by means of a *parse*. A parse is generated by a *grammar* that comprises *production rules* (or just *rules*).

Let $\mathcal{S}$ be a schema. Conventionally, a grammar has *terminals* and *nonterminals*. In our kind of grammar, the nonterminals are the concepts of $\mathcal{S}$, and there are two types of terminals. A terminal of the first type is a term of $\mathbf{T}$. A terminal of the second type has the form $\lceil\gamma\rceil$, where $\gamma \in \mathcal{S}^{\text{a}}$ is an a-concept. Intuitively, $\lceil\gamma\rceil$ means an *instantiation* of the concept γ. Thus, $\lceil$person$\rceil$ refers to *Sara Shackleton*, *John*, etc. The exact role of the terminal $\lceil\gamma\rceil$ will be clarified later, when we define the *evaluation* of a parse in the database. We use $\lceil\mathcal{S}^{\text{a}}\rceil$ to denote the set of all the terminals $\lceil\gamma\rceil$, such that γ is an a-concept in $\mathcal{S}$. Our grammar rules take the standard form of rules in a *context-free grammar*. There are two types of rules.

a-rules. An *atomic-concept rule* (over $\mathcal{S}$), or *a-rule* for short, defines how an a-concept γ is described by a string of terms, and possibly an instantiation of γ (namely $\lceil\gamma\rceil$). An a-rule has the form $\gamma \rightarrow \sigma_1 \cdots \sigma_m$, where $\gamma \in \mathcal{S}^{\text{a}}$ is an a-concept, m is a positive integer, and each σ_i is either a term of $\mathbf{T}$ or the terminal $\lceil\gamma\rceil$. We allow at most one σ_i to be $\lceil\gamma\rceil$ (while all the rest are terms). As an example, person → *someone* means that a person can be referred to by the keyword *someone*, and person → $\lceil$person$\rceil$ means that a person can be referred to by a string (e.g., *sara*) that matches an actual person (e.g., Sara Shackleton).

c-rules. A *compound-concept rule* (over $\mathcal{S}$), or *c-rule* for short, defines how a c-concept is described by a combination of its subconcepts and of terms. A c-rule has the form $\gamma \rightarrow \sigma_1 \cdots \sigma_m$, where $\gamma \in \mathcal{S}^{\text{c}}$ is a c-concept, m is a positive integer, and each σ_i is in either $\mathsf{sc}(\gamma)$ or $\mathbf{T}$. We allow each $\delta \in \mathsf{sc}(\gamma)$ to appear at most once in $\sigma_1 \cdots \sigma_m$. An example of a c-rule is senderInfo → sender *from* fromAddr, where sender and fromAddr are subconcepts of senderInfo, and *from* is a term.

Note that in the definition of a c-rule, $\sigma_1 \cdots \sigma_m$ does not necessarily contain all the subconcepts of γ. As an example, for the c-concept car with the subconcepts make and model, we may decide that model is enough to refer to a car (in a search query), and have the c-rule car → model.

We refer to an a-rule or a c-rule simply as a *rule*. The *head* of a rule ρ, denoted head(ρ), is the concept on the left-hand side of ρ. The *body* of ρ, denoted body(ρ), is the string on its right-hand side. A *grammar* (over $\mathcal{S}$) is a finite set Γ of rules over $\mathcal{S}$.

Example 2.3. We now construct a grammar Γ over the schema $\mathcal{S}$ of Figure 1(a). Following are the rules of Γ.

person → $\lceil$person$\rceil$
sender → $\lceil$sender$\rceil$ | *from* $\lceil$sender$\rceil$
fromAddr → $\lceil$fromAddr$\rceil$ | *from* $\lceil$fromAddr$\rceil$
phoneNum → *phone* | *number* | *phone number*
msgText → $\lceil$msgText$\rceil$
employee → $\lceil$employee$\rceil$
org → $\lceil$org$\rceil$ | $\lceil$org$\rceil$ *organization*
personPhone → person phoneNum | phoneNum *of* person
senderInfo → sender fromAddr | sender *at* fromAddr
worksFor → employee org | employee *from* org

Note that $x \rightarrow y_1 \mid y_2 \mid \cdots \mid y_k$ denotes the k rules $x \rightarrow y_i$ for $i = 1, \ldots, k$. Words in italic font (e.g., *from* and *phone*) are terms. Note that the top seven lines show a-rules, and the bottom three lines show c-rules.

Observe that all the a-rules use symbols of $\lceil\mathcal{S}^{\text{a}}\rceil$ in their body (possibly, in addition to terms), except for those with the head phoneNum. This illustrates the idea that we may decide not to view a phoneNum part of a search query as one that refers to a specific number (e.g., 408-123-4567), because it is probably unlikely to capture the intent of the user phrasing the query; rather, phoneNum always comes as a description of the phone-number concept (e.g., by writing "*phone*," "*number*," or "*phone number*").

Some rules (e.g., the rule person → $\lceil$person$\rceil$ and the rule personPhone → person phoneNum) can be obtained from the schema automatically and straightforwardly. Other rules (such as the rules phoneNum → *number* and personPhone → phoneNum *of* person) are introduced either manually or by adopting relevant learning techniques [24]. □

2.4 Search Database Systems

Now, we are ready to define a *search database system*, which is our basic infrastructure for answering search queries.

Definition 2.4. A *search database system*, or *SDBS* for short, is a triple $(\mathcal{S}, I, \Gamma)$, where $\mathcal{S}$ is a schema, I is an instance over $\mathcal{S}$, and Γ is a grammar over $\mathcal{S}$. □

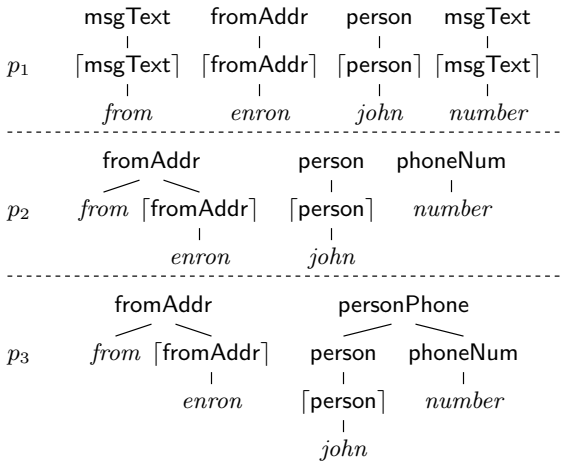


Figure 2: Parses for *"from enron john number"*

Example 2.5. In our running example, we have the SDBS $(\mathcal{S}, I, \Gamma)$ of the email-search domain, where $\mathcal{S}$ (Figure 1(a)) is described in Example 2.1, I (Figures 1(b) and 1(c)) is described in Example 2.2, and Γ is described in Example 2.3.

2.5 Parses

Next, we define a *parse hedge* (or just *parse* for short) for a search query. Intuitively, a parse for a search query $\mathbf{q}$ is a hedge that is produced by the grammar, such that every tree represents the meaning of a fragment of $\mathbf{q}$ by means of concepts and concept instantiations. The formal definition follows.

Definition 2.6. Let $(\mathcal{S}, I, \Gamma)$ be an SDBS, and let $\mathbf{q} \in \mathbf{T}^+$ be a search query. A *parse* for $\mathbf{q}$ is a hedge $p = t_1 \cdots t_n$ over $\mathcal{S} \cup \lceil \mathcal{S}^{\mathrm{a}} \rceil \cup kw(\mathbf{q})$, such that p satisfies the following.

1. The leaves of p are exactly the nodes v having $\mathrm{label}(v) \in kw(\mathbf{q})$. Moreover, $\mathbf{q}$ is the concatenation of all the keywords that appear in p from left to right. We note that $\mathbf{q}$ is often referred to as the *yield* of p.

2. For each node v of p with children $u_1, \ldots, u_m$ (from left to right), if $\mathrm{label}(v) \in \mathcal{S}$, then Γ contains the rule
$$\mathrm{label}(v) \to \mathrm{label}(u_1) \cdots \mathrm{label}(u_m)\,.$$

3. For each node v of p, if $\mathrm{label}(v) \in \lceil \mathcal{S}^{\mathrm{a}} \rceil$, then all the children of $\mathrm{label}(v)$ are leaves (equivalently, the label of each child of v is a keyword).

4. For all $i = 1, \ldots, n$, the label of $\mathrm{root}(t_i)$ is a concept of $\mathcal{S}$. □

Example 2.7. Let $(\mathcal{S}, I, \Gamma)$ be the SDBS of Example 2.5, and let $\mathbf{q}$ be the search query *from enron john number*. Figure 2 shows three parses for $\mathbf{q}$. The parse p_1 comprises four trees, p_2 comprises three trees, and p_3 comprises two trees. Note that Item 1 of Definition 2.6 holds in each hedge, since the leaves constitute the search query when read from left to right. As an example for Item 2, observe that p_2 has a fromAddr node with children *from* and ⌈fromAddr⌉, and indeed, the grammar Γ contains the rule fromAddr → *from* ⌈fromAddr⌉. For Item 3, observe that the child of each node with a label of the form $\lceil\gamma\rceil$ (e.g., ⌈person⌉) is a leaf. Finally, Item 4 holds since every root label (e.g., fromAddr) is a concept.

Intuitively, a tree in a parse represents an independent part of the search query, and the user intent for that part is captured by the concept that labels the root of the tree. For example, the parse p_1 describes four parts: the first and fourth keywords (i.e., *from* and *number*) are terms that should appear in (or more precisely match) the message text, the second keyword says that the sender address matches *enron*, and the third one says that a person that matches *john* should be in the message. The parse p_2 has three parts: the first one, which is described by the first two keywords, says that the sender address matches *enron*, the second says that there is a person that matches *john*, and the third says that *some* phone number is specified in the email. Note that unlike the parse p_1, the parse p_2 does not require any actual match, in the email, for the keywords *from* and *number*. Finally, p_3 has two parts: first, the sender address matches *enron*, and second, there is a person that matches *john*, who has a phone number specified in the email. □

Finally, we define the semantics of *evaluating* a parse in an SDBS, namely, finding all the URIs with a content that match the parse. We first consider the question of when a string $\mathbf{k}$ of keywords *matches* an a-record r^γ for an a-concept γ. We do not actually study this aspect here; rather, we assume a binary relationship over strings that is already resolved and represented by the relation $\propto_\gamma$. Thus, $\mathbf{k}$ matches r if $\mathbf{k} \propto_\gamma txt(r)$. For simplicity, in our examples we interpret $\propto_\gamma$ as the pure substring relationship where case is ignored (e.g., *sara* $\propto_{\mathsf{person}}$ *Sara Shackleton*). Clearly, much more intricate interpretations are required in practice (including, e.g., dictionaries of synonyms). For example, in practice we will probably have the following, even though there is not a substring relationship: *Sarah* $\propto_{\mathsf{person}}$ *Sara Shackleton*, *PGE Training* $\propto_{\mathsf{org}}$ *PGE Energy Training*, and *ENA* $\propto_{\mathsf{org}}$ *Enron North America Inc.* Note that using a specific $\propto_\gamma$ for each a-concept γ is highly important in practice, since one is likely to employ different matching techniques in different contexts. For example, consider the term *chris* and the term *Christopher*, where the latter is a popular person name and also the name of a city of Illinois. The term *chris* is likely to match *Christopher* as a person name, but not as a city name (thus, e.g., *chris* $\propto_{\mathsf{person}}$ *Christopher* holds whereas *chris* $\propto_{\mathsf{city}}$ *Christopher* does not). Finally, observe that in practice this binary relationship often means that the *level* of relevancy of the record to the string (estimated by some scoring function) is above some threshold.

Let $(\mathcal{S}, I, \Gamma)$ be an SDBS, let $\mathbf{q}$ be a search query, and let p be a parse for $\mathbf{q}$. For a node v of p, we denote by $\mathbf{T}^+(v)$ the string that comprises the terms (labels) of the leaves that are descendants of v from left to right. A *match* of p in I is a mapping $\mu : \mathrm{internal}(p) \to I$ (i.e., μ maps every internal node of p to a record of I) that satisfies the following.

1. The image of μ has a single URI; that is, $\mathrm{uri}(\mu(v)) = \mathrm{uri}(\mu(u))$ for all nodes $v, u \in \mathrm{internal}(p)$. We denote this URI by $\mathrm{uri}(\mu)$.

2. Concepts are preserved; that is, for all $v \in \mathrm{internal}(p)$ where $\mathrm{label}(v)$ is either γ or $\lceil\gamma\rceil$, the record $\mu(v)$ has the concept γ.

3. For all nodes $v \in \text{internal}(p)$, if $\text{label}(v)$ is $\lceil\gamma\rceil$ (and, in particular, γ is an a-concept), then $\mathbf{T}^+(v) \propto_\gamma txt(\mu(v))$.

4. For all edges (v, w) of p, if $\text{label}(v)$ is γ and $\text{label}(w)$ is $\lceil\gamma\rceil$ (in particular, γ is an a-concept), then $\mu(v) = \mu(w)$.

5. For all edges (v, w) of p, if $\text{label}(w)$ is a concept $\delta \in \mathcal{S}$ (hence, $\mathsf{sc}(\text{label}(v))$ contains δ), then $\mu(v)$ references $\mu(w)$; that is, $r[\delta] = \mu(w)$, where $r = \mu(v)$.

We denote by $p(I)$ the set of all the URIs u such that there exists a match of p in I with $\text{uri}(p) = u$.

Example 2.8. Consider again the SDBS $(\mathcal{S}, I, \Gamma)$ of Example 2.5, and the three parses p_1, p_2 and p_3 of Figure 2 (discussed in Example 2.7). Recall that $\mathcal{S}$ and I are depicted in Figure 1. We will illustrate the notion of a match by considering the parse p_3. Since every node of p_3 has a unique label, we will describe a match μ by means of mappings $\text{label}(v) \mapsto r$, where v is a node and r is a record. The following is a match μ of p_3 in I, such that $\text{uri}(\mu) = \texttt{em2}$.

$$\mathsf{fromAddr} \mapsto s_{\text{ad}} \quad \lceil\mathsf{fromAddr}\rceil \mapsto s_{\text{ad}} \quad \mathsf{person} \mapsto s_{\text{pr}}$$
$$\lceil\mathsf{person}\rceil \mapsto s_{\text{pr}} \quad \mathsf{phoneNum} \mapsto s_{\text{ph}} \quad \mathsf{personPhone} \mapsto s_{\text{pp}}$$

Consider Items 1–5 in the above definition of a match. Item 1 holds for μ since every record r in the image of μ is such that $\text{uri}(r) = \texttt{em2}$ (hence, $\text{uri}(\mu) = \texttt{em2}$). For Item 2, observe that each $\gamma \mapsto r$ and each $\lceil\gamma\rceil \mapsto r$ are such that γ is the concept of r. For Item 3, consider the mapping $\lceil\mathsf{fromAddr}\rceil \mapsto s_{\text{ad}}$. Note that the sequence of keywords under $\lceil\mathsf{fromAddr}\rceil$ is *enron*, the string $txt(s_{\text{ad}})$ is *john@enron.com*, and, as required, it holds that *enron* $\propto_{\mathsf{fromAddr}}$ *john@enron.com* (under our specific interpretation of $\propto_\gamma$ in the examples). For Item 4, note that the nodes of p_3 labeled $\mathsf{fromAddr}$ and $\lceil\mathsf{fromAddr}\rceil$ (which are a parent and a child, respectively) are mapped to the same record, namely s_{ad}. Finally, for Item 5, consider the mapping $\mathsf{personPhone} \mapsto s_{\text{pp}}$. As required, the children of the node labeled $\mathsf{personPhone}$ are mapped to the records s_{pr} and s_{ph}, which are referenced by s_{pp}.

Since the match μ exists, we conclude that $\texttt{em2} \in p_3(I)$. Note that there is no match μ' of p_3 in I with $\text{uri}(\mu') = \texttt{em1}$, since John is not associated with a phone number in $\texttt{em1}$. Thus, it follows that $p_3(I) = \{\texttt{em2}\}$.

One can similarly show that $p_2(I) = \{\texttt{em1}, \texttt{em2}\}$. In particular, note that a match μ of p_2 in I, with $\text{uri}(\mu_2) = \texttt{em1}$, maps $\lceil\mathsf{person}\rceil$ to r_{pr1} and $\mathsf{phoneNum}$ to r_{ph} (which are unrelated person and phone number). Finally, note that $p_1(I) = \emptyset$; for example, this holds since there is a match of the word *number* in neither of the two email bodies in I. □

3. GENERATING PARSES

In this section, we consider the problem of producing the parses for a search query. Formally, we are given as input an SDBS $(\mathcal{S}, I, \Gamma)$ and a search query $\mathbf{q} \in \mathbf{T}^+$, and the goal is to generate all the data-nonempty parses, where a parse p is *data-nonempty* if $p(I)$ is nonempty.

If one assumes that the length of the query $\mathbf{q}$ and the depth of the schema $\mathcal{S}$ are upper bounded by some constant value, then all the parses can be generated by a straightforward polynomial-time algorithm that considers all the hedges over $\mathcal{S} \cup \lceil\mathcal{S}^{\text{a}}\rceil \cup kw(\mathbf{q})$ with bounded height and width. Here, we show that tractability holds even without any bound assumption. The problem is that, now, the number of data-nonempty parses can be exponential in the size of $\mathcal{S}$ and $\mathbf{q}$. Thus, *polynomial running time* is not a suitable yardstick of efficiency, since just writing the output may require exponential time. We view an algorithm for generating the parses as an *enumeration* algorithm that produces a (possibly large) set of *output items*, where in our case an output item is a parse for $\mathbf{q}$. An enumeration algorithm is not allowed to print the same output item more than once. The conventional yardstick of efficiency for such an algorithm is *polynomial input-output complexity* [11], which means that the running time is polynomial in the combined size of the input and the output. Our formal result is the following.

THEOREM 3.1. *Producing all the data-nonempty parses, given an SDBS and a search query, has a polynomial input-output complexity.*

Note that the task that is considered in Theorem 3.1 differs from standard generation of parses (under context-free grammars) in two technical aspects. First, our parses are hedges (not necessarily trees). Second, data-nonemptiness is required. In the remainder of this section, we prove this theorem. We fix an SDBS $(\mathcal{S}, I, \Gamma)$ and a query $\mathbf{q} = q_1 \cdots q_k$ for the problem.

The algorithm is based on the notion of *constraints* over parses, which we define as follows. Consider a parse p and a node v of p. Suppose that among the keywords of $\mathbf{q}$ that are descendants of v, the ith keyword, q_i, is the leftmost one, and the jth keyword, q_j, is the rightmost one. We denote by $\text{span}(v)$ the pair (i, j). We use two types of constraints over parses. A *positive* constraint has the form $\frac{X}{(i,j)}$, where $X \in \mathcal{S} \cup \lceil\mathcal{S}^{\text{a}}\rceil$ and i and j are indices such that $1 \leq i \leq j \leq k$. A parse p *satisfies* $\frac{X}{(i,j)}$ if p contains a node v, such that $\text{label}(v) = X$ and $\text{span}(v) = (i, j)$. A *negative* constraint is the negation of a positive constraint, and it has the form $\neg\frac{X}{(i,j)}$.

Let $\mathcal{C}$ be a set of constraints. We say that a parse p *satisfies* $\mathcal{C}$ if p satisfies each of the constraints of $\mathcal{C}$. The set $\mathcal{C}$ is *data satisfiable* if it is satisfied by at least one data-nonempty parse. We say that $\mathcal{C}$ is *saturated* if there is no positive constraint $\frac{X}{(i,j)}$, such that $\frac{X}{(i,j)} \notin \mathcal{C}$ and $\mathcal{C} \cup \{\frac{X}{(i,j)}\}$ is data satisfiable. Our algorithm is based on the following lemmas.

LEMMA 3.2. *If $\mathcal{C}$ is a set of constraints that is data satisfiable and saturated, then there exists exactly one data-nonempty parse that satisfies $\mathcal{C}$.*

For a saturated, data satisfiable set $\mathcal{C}$ of constraints, we denote by $p_{\mathcal{C}}$ the unique data-nonempty parse that satisfies $\mathcal{C}$.

LEMMA 3.3. *Given a set $\mathcal{C}$ of constraints, whether $\mathcal{C}$ is data satisfiable and whether $\mathcal{C}$ is saturated can be decided in polynomial time. Moreover, if $\mathcal{C}$ is data satisfiable and saturated, then $p_{\mathcal{C}}$ can be computed in polynomial time.*

Building on Lemma 3.3, we get a simple algorithm for enumerating all the data-nonempty parses in polynomial time in the combined size of the input and the output. This algorithm is depicted in Figure 3, and it follows the lines of the algorithm of [15] for enumerating "reduced subtrees." The algorithm gets as input the SDBS $(\mathcal{S}, I, \Gamma)$ and the search query $\mathbf{q}$. In addition, the algorithm gets as input a set $\mathcal{C}$ of

Algorithm EnumDataNonempty($\mathcal{C}$)

1: **if** $\mathcal{C}$ is not data satisfiable **then**
2: **return**
3: **if** $\mathcal{C}$ is saturated **then**
4: **print** $p_{\mathcal{C}}$
5: **return**
6: **else**
7: let $\frac{X}{(i,j)}$ be s.t. $\frac{X}{(i,j)} \notin \mathcal{C}$ and $\mathcal{C} \cup \{\frac{X}{(i,j)}\}$ is data satisfiable
8: EnumDataNonempty($\mathcal{C} \cup \{\frac{X}{(i,j)}\}$)
9: EnumDataNonempty($\mathcal{C} \cup \{\neg\frac{X}{(i,j)}\}$)

Figure 3: Enumerating all the data-nonempty parses

constraints, which is empty when the algorithm is called for the first time (i.e., not within a recursive call). For simplicity of presentation, we omit $(\mathcal{S}, I, \Gamma)$ and $\mathbf{q}$ from the arguments, and specify only $\mathcal{C}$. EnumDataNonempty($\mathcal{C}$) enumerates all the data-nonempty parses that satisfy $\mathcal{C}$.

In Line 1, the algorithm tests whether $\mathcal{C}$ is data satisfiable. If not, then the algorithm terminates in Line 2. In Line 3, the algorithm tests whether $\mathcal{C}$ is saturated. If so, then $p_{\mathcal{C}}$ is printed in Line 4 and the algorithm terminates in Line 5. The ability to efficiently execute Lines 1, 3 and 4 is due to Lemma 3.3. If $\mathcal{C}$ is data satisfiable but not saturated, then in Line 7 a constraint $\frac{X}{(i,j)}$ is chosen, such that $\mathcal{C} \cup \{\frac{X}{(i,j)}\}$ is data satisfiable. Note that such a constraint can be found efficiently by testing the applicability of each possible $\frac{X}{(i,j)}$ (again, Lemma 3.3 is needed). Observe that there is only a polynomial number of possible constraints. Then, the space of data-nonempty parses that satisfy $\mathcal{C}$ is split into two partitions: The recursive call of Line 8 prints all the data-nonempty parses that satisfy $\mathcal{C} \cup \{\frac{X}{(i,j)}\}$, and that of Line 9 prints the rest, namely, the data-nonempty parses that satisfy $\mathcal{C} \cup \{\neg\frac{X}{(i,j)}\}$.

Correctness and efficiency of the algorithm are shown in the next lemma. Note that theorem 3.1 follows immediately from this lemma (specifically, when taking $\mathcal{C} = \emptyset$).

LEMMA 3.4. EnumDataNonempty($\mathcal{C}$) *enumerates all the data-nonempty parses that satisfy $\mathcal{C}$. The running time is polynomial in the combined size of the input and the output.*

EnumDataNonempty does not have any guarantee on the order of printed answers. Later on, we study the problem of enumerating by decreasing specificity, where the notion of *specificity* is considered in the next section.

4. CONTAINMENT OF PARSES

In this section, we consider the notion of *specificity* among parses of a search query. In our implemented engines, this notion plays a central role in ranking a set of candidate parses. We identify specificity with the standard concept of *query containment*. More formally, let $\mathcal{S}$ be a schema, let Γ be a grammar over $\mathcal{S}$, and let $\mathbf{q}$ be a search query. A parse p_1 is *at least as specific as* the parse p_2 if $p_1(I) \subseteq p_2(I)$ for all instances I over $\mathcal{S}$; in that case, we also say that p_1 is *contained in* p_2, and denote it by $p_1 \sqsubseteq p_2$.

Example 4.1. As a simple example, consider the schema $\mathcal{S}$ and the grammar Γ of Example 2.5. In Figure 2, the parse p_3

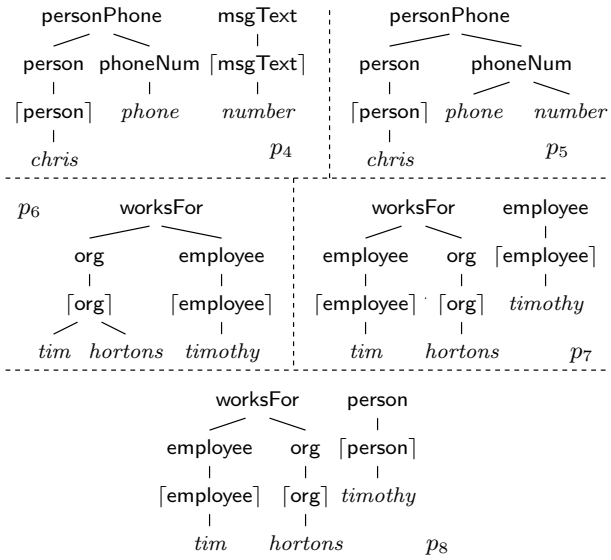


Figure 4: Examples of containment relationships

is contained in the parse p_2. This is a simple example where containment is obvious, since p_2 can be obtained from p_3 by removing the node with the label personPhone. More interesting examples are depicted in Figure 4. The parses p_4 and p_5 for the query *chris phone number* are such that $p_4 \sqsubseteq p_5$, since both require the specification of Chris with his phone number, whereas p_4 has the extra requirement that "number" appears in the text (hence, $p_5 \not\sqsubseteq p_4$).

A more intricate example is about p_6 and p_7, which are parses for *tim hortons timothy*. Recall that, in our examples, we use the substring relationship as the specific (naive) interpretation of $\propto_\gamma$. With this interpretation, we claim that p_6 is contained in p_7. To see that, consider an instance I over $\mathcal{S}$ and suppose that $u \in p_6(I)$. Then in u, Timothy is an employee of Tim Hortons. This implies that u satisfies the right tree of p_7. Moreover, if *timothy* $\propto_{\mathsf{employee}}$ $txt(r)$, then *tim* $\propto_{\mathsf{employee}}$ $txt(r)$ (due to the special interpretation of $\propto_\gamma$). Similarly, if *tim hortons* $\propto_{\mathsf{org}}$ $txt(r)$ for a record $r \in I$, then *hortons* $\propto_{\mathsf{org}}$ $txt(r)$. Therefore, a match of p_6 can be thought of as a match of the left tree of p_7. It follows then that $u \in p_7$. Hence, $p_6 \sqsubseteq p_7$, as claimed. □

The discussion about p_6 and p_7 in Example 4.1 implies that in order to reason about specificity of parses, one needs to know when a match of one substring of the query guarantees a match of another substring of the query. In the above example, we used the transitivity and reflexivity of the substring interpretation of $\propto_\gamma$. In the remainder of this paper, we take these two properties as an assumption. Formally, we assume that, for each a-concept γ, the binary relation $\propto_\gamma$ is a *preorder*, that is, it is transitive (i.e., for all strings $\mathbf{k}$, $\mathbf{r}$ and $\mathbf{s}$, if $\mathbf{k} \propto_\gamma \mathbf{r}$ and $\mathbf{r} \propto_\gamma \mathbf{s}$, then $\mathbf{k} \propto_\gamma \mathbf{s}$) and reflexive (i.e., $\mathbf{k} \propto_\gamma \mathbf{k}$ for all strings $\mathbf{k}$).[4] We furthermore require that for all a-concepts γ and nonempty strings $\mathbf{k}$, it is never the case that $\mathbf{k} \propto_\gamma \epsilon$ (where ϵ is the empty string).

[4]We note that the results in the remainder of this paper hold under a restriction that is much weaker than $\propto_\gamma$ being a preorder. However, this restriction requires a nontrivial discussion that we avoid for simplicity of presentation.

Consider now the parses p_7 and p_8 of Figure 4. These parses are similar, except that in p_7 Timothy is an employee whereas in p_8 he is a person. So, in principle, these parses are incomparable. However, person is effectively a generalization of employee. Thus, it makes practical sense to assume that if Timothy appears in an employee record, he will also appear in a person record. By taking that into account, we would like it to be the case that p_7 is contained in p_8 (and also, p_6 is contained in p_8 since p_6 is contained in p_7).

Related to the above is the following issue. The concept msgText has a special role, since it contains the entire body of the email. Since an organization is extracted from the body of the email, it makes sense that whenever a string (e.g., *tim hortons*) matches an organization, that string also matches a msgText record. To accommodate that, we extend our model with a preorder among the a-concepts. This is formalized next.

From now on, we require a schema $\mathcal{S}$ to have a preorder $\preceq$ over $\mathcal{S}^{\mathrm{a}}$.[5] We interpret $\gamma_1 \preceq \gamma_2$ as "γ_1 specializes γ_2." In the above examples, for instance, $\preceq$ can be such that employee $\preceq$ person and org $\preceq$ msgText. If $\gamma_1 \preceq \gamma_2$, then we require the relation $\propto_{\gamma_2}$ to extend $\propto_{\gamma_1}$ in the following sense: for all strings $\mathbf{k}_1$ and $\mathbf{k}_2$ in $\mathbf{T}^*$, if $\mathbf{k}_1 \propto_{\gamma_1} \mathbf{k}_2$, then $\mathbf{k}_1 \propto_{\gamma_2} \mathbf{k}_2$. For example, if *tim* matches *timothy* as an employee, then *tim* also matches *timothy* as a person.

The introduction of $\preceq$ necessitates adapting the definition of an *instance over a schema* $\mathcal{S}$. An instance I over $\mathcal{S}$ is now required to respect $\preceq$, in the following sense. If I says that Tim is an employee in the URI u, then Tim should also appear as a person in u. If the URI u has the organization "IBM Research," then the message text of u *contains* (or includes a match for) "IBM Research." This adaptation is formalized in the following requirement.

An instance I over $\mathcal{S}$ should satisfy that for all a-concepts $\gamma \in \mathcal{S}^{\mathrm{a}}$, a-records $r^\gamma \in I$, and a-concepts $\delta \in \mathcal{S}^{\mathrm{a}}$ such that $\gamma \preceq \delta$, there exists an a-record $s^\delta \in I$, such that $\mathrm{uri}(r^\gamma) = \mathrm{uri}(s^\delta)$, and $txt(r^\gamma) \propto_\delta txt(s^\delta)$.

4.1 Deciding on Containment

Next, we consider the computational problem of deciding containment among parses. We will show that this problem is tractable by presenting an efficient decision procedure. The definition of containment is not constructive, as it requires one to consider every instance I over $\mathcal{S}$. So, to be able to decide on containment, we first give a constructive characterization. For that, some definitions are required.

Let δ and γ be concepts of a schema $\mathcal{S}$. We say that δ is *derived* from γ if the existence of a γ record implies the existence of a δ record. For example, person is implied by employee since employee $\preceq$ person. The concept person is also implied by worksFor, since person is implied by employee and employee is a subconcept of worksFor. Formally, δ is derived from γ if there is a directed path from γ to δ in the directed graph that has $\mathcal{S}$ as its set of nodes, and an edge (γ_1, γ_2) whenever $\gamma_2 \in \mathsf{sc}(\gamma_1)$ or $\gamma_1 \preceq \gamma_2$.

Let $\mathcal{S}$ be a schema, let Γ be a grammar, let $\mathbf{q} \in \mathbf{T}^+$ be a search query, and let p be a parse for $\mathbf{q}$. A node v of p is *dummy* if none of the ancestors or descendants of v (including v itself) has a label in $\lceil \mathcal{S}^{\mathrm{a}} \rceil$. For example, in Figure 4, the dummy nodes of p_5 are those with the labels phoneNum, *phone* and *number*. We denote by $p_\downarrow$ the hedge that is obtained from p by removing all the dummy nodes. Following is the definition of an *embedding* between parses. Recall that $\mathbf{T}^+(v)$ is the concatenation of the keywords that are descendants of the node v.

[5]Note that $\preceq$ is defined only over a-concepts. Extending $\preceq$ to c-concepts is left for future work.

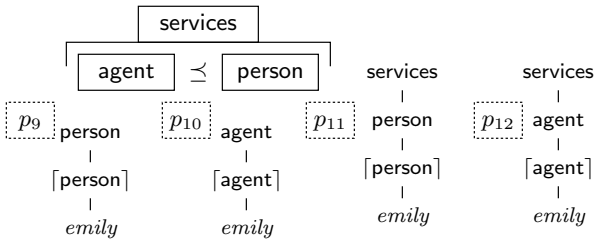


Figure 5: Four parses for *emily*

Definition 4.2. Let $\mathcal{S}$ be a schema, let Γ be a grammar, and let $\mathbf{q} \in \mathbf{T}^+$ be a search query. Let p and p' be two parses for $\mathbf{q}$. An *embedding* of p in p' is a mapping $\varphi : \mathrm{internal}(p_\downarrow) \to \mathrm{internal}(p'_\downarrow)$ that satisfies the following.

1. For all nodes $v \in \mathrm{internal}(p_\downarrow)$, if $\mathrm{label}(v) \in \mathcal{S}^{\mathrm{c}}$, then $\mathrm{label}(\varphi(v)) = \mathrm{label}(v)$.
2. For all nodes $v \in \mathrm{internal}(p_\downarrow)$, if $\mathrm{label}(v) \in \mathcal{S}^{\mathrm{a}}$ then $\mathrm{label}(\varphi(v)) \in \mathcal{S}^{\mathrm{a}}$ and $\mathrm{label}(\varphi(v)) \preceq \mathrm{label}(v)$. Moreover, if v is not a root of p, then $\mathrm{label}(\varphi(v)) = \mathrm{label}(v)$.
3. For all nodes $v \in \mathrm{internal}(p_\downarrow)$, if $\mathrm{label}(v) = \lceil \gamma \rceil$, then $\mathrm{label}(\varphi(v)) \in \lceil \mathcal{S}^{\mathrm{a}} \rceil$ and $\mathbf{T}^+(v) \propto_\gamma \mathbf{T}^+(\varphi(v))$.
4. For all nodes $v, w \in \mathrm{internal}(p_\downarrow)$, if (v, w) is an edge of p, then $(\varphi(v), \varphi(w))$ is an edge of p'. □

The following theorem gives a structural characterization of the containment of one parse in another.

THEOREM 4.3. *Let $\mathcal{S}$ be a schema, let Γ be a grammar, and let $\mathbf{q} \in \mathbf{T}^+$ be a search query. Let p_1 and p_2 be two parses for q. Then $p_1 \sqsubseteq p_2$ if and only if both of the following conditions hold.*

1. *Every concept that labels a dummy node v of p_2 is derived from some label of p_1.*
2. *There is an embedding of p_2 in p_1.*

Note that Condition 2 of Definition 4.2 has different requirements for nodes v with atomic $\mathrm{label}(v)$, depending on whether or not v is a root of p. We discuss this difference by an example. Let $\mathcal{S}$ be the schema depicted in the top-left corner of Figure 5. In this schema, services is a c-concept that has as subconcepts the a-concepts agent and person, and it represents the relationship "the agent services the person." In addition, agent $\preceq$ person. Consider the four parses for the query *emily* in this figure. There is an embedding of p_9 in p_{10} and, indeed, if Emily is an agent then she is also a person. Now, consider the parses p_{11} and p_{12}. If we removed the "Moreover" part of Condition 2, then there would be an embedding of p_{11} in p_{12}. However, p_{12} is not contained in p_{11}, since the fact that Emily is an agent that gives service does not imply that she gets service as well. Thus, the "Moreover" part of Condition 2 is necessary for the correctness of Theorem 4.3.

The characterization of Theorem 4.3 implies an efficient, bottom-up algorithm for testing containment among parses. (We do not give the algorithm here, since it is fairly straightforward, and requires additional notation.) Hence, we get the following result.

THEOREM 4.4. *Given a schema $\mathcal{S}$, a grammar Γ, a search query $\mathbf{q}$ and two parses p_1 and p_2 for $\mathbf{q}$, whether $p_1 \sqsubseteq p_2$ can be decided in polynomial time.*

5. ENUMERATION BY SPECIFICITY

Theorem 3.1 says that the data-nonempty parses for a search query can be produced in polynomial time in the combined size of the input and the output. Since many data-nonempty parses may exist, we may require more—generate the parses *incrementally* in a *ranked* order. Ranking may vary in different contexts. In our domains of interest, we desire parses of a high specificity. So, here, we consider the case where ranked order means decreasing specificity. Note that an immediate corollary of Theorems 3.1 and 4.4 is that we can generate all the data-nonempty parses with polynomial input-output complexity, and then efficiently sort them in the order of specificity. However, that requires the user to wait until all the data-nonempty parses are generated and sorted, before even one parse is printed. To avoid that, we set the goal of an incremental enumeration that guarantees only *polynomial delay* [11], that is, the time before printing the first parse, and then between every two consecutive parses, is polynomial only in the size of the input.

Formally, the problem at hand is the following. We are given as input an SDBS $(\mathcal{S}, I, \Gamma)$ and a search query $\mathbf{q} \in \mathbf{T}^+$. For two parses p_1 and p_2, we denote by $p_1 \sqsubset p_2$ the fact that $p_1 \sqsubseteq p_2$ holds, but $p_2 \sqsubseteq p_1$ does not. The goal is to enumerate the data-nonempty parses for $\mathbf{q}$, so that for two parses p_1 and p_2, if $p_1 \sqsubset p_2$ than p_1 is printed before p_2. We then say that the enumeration is *by decreasing specificity*. Our goal is to enumerate the data-nonempty parses by decreasing specificity with polynomial delay. Unfortunately, we next show that this goal is intractable, even under strong restrictions.

Consider an SDBS $(\mathcal{S}, I, \Gamma)$. We say that $\mathcal{S}$ is *flat* if all the concepts of $\mathcal{S}$ are atomic (i.e., $\mathcal{S} = \mathcal{S}^{\mathrm{a}}$). We say that $\mathcal{S}$ is *unordered* if the concepts of $\mathcal{S}$ are incomparable by $\preceq$; that is, for all concepts $\gamma_1, \gamma_2 \in \mathcal{S}$, if $\gamma_1 \neq \gamma_2$ then neither $\gamma_1 \preceq \gamma_2$ nor $\gamma_2 \preceq \gamma_1$ holds. Finally, we say that Γ is *text oblivious* if none of the rules of Γ contains a terminal of $\lceil \mathcal{S}^{\mathrm{a}} \rceil$. Thus, if Γ is text oblivious, then a parse p is simply a requirement for existence concepts (namely those specified in p), and it disregards the textual content of records. We use the following lemma.

LEMMA 5.1. *Given a schema $\mathcal{S}$, a grammar Γ and a search query $\mathbf{q}$, deciding whether there is a parse p containing all the concepts of $\mathcal{S}$ is NP-complete, even under the restriction that $\mathcal{S}$ is flat and unordered, and Γ is text oblivious.*

The proof of Lemma 5.1 is by a reduction from the problem of *scheduling on a single machine*, which is known to be NP-complete[6] [6].

[6]Our proof builds on the fact that this problem is NP-complete *in the strong sense* [7], which means that the problem is NP-complete even if it is assumed that the lengths, release times and deadlines of tasks are given in unary representation.

Consider an SDBS $(\mathcal{S}, I, \Gamma)$ and a search query $\mathbf{q} \in \mathbf{T}^+$. Suppose that $\mathcal{S}$ is flat and unordered, and that Γ is text oblivious. Suppose also that I contains a single URI u and a record r^γ, with $\mathrm{uri}(r) = u$, for each $\gamma \in \mathcal{S}$. In this specific case, for two parses p_1 and p_2, the relation $p_1 \sqsubseteq p_2$ is equivalent to $\mathcal{S}(p_1) \subseteq \mathcal{S}(p_2)$, where $\mathcal{S}(p_i)$ is the set of concepts that appear in p_i. In particular, the first parse that is generated by an algorithm that enumerates by decreasing specificity includes all the concepts of $\mathcal{S}$, unless no such parse exists. Lemma 5.1 implies that such a parse cannot be obtained in polynomial time if P $\neq$ NP. Thus, we conclude the following theorem, showing that the task of enumerating the parses by decreasing specificity is intractable.

THEOREM 5.2. *If* P $\neq$ NP*, then no algorithm enumerates the data-nonempty parses by decreasing specificity with polynomial delay.*

In the remainder of this section, we formulate a restriction of the general problem, which we believe holds commonly in practice, and show that this restriction allows for ranked enumeration with polynomial delay. The restriction is that of requiring the query to be *simple*, which we define as follows. Let i and j be integers, such that $i \leq j$. We denote by $[i, j]$ the set $\{i, i+1, \ldots, j\}$. Let us say that a parse tree is *dummy* if it does not contain a label of $\lceil \mathcal{S}^{\mathrm{a}} \rceil$ (i.e., all the nodes are dummy).

Definition 5.3. Let $(\mathcal{S}, I, \Gamma)$ be an SDBS. A query $\mathbf{q} \in \mathbf{T}^+$ is *simple* if the following conditions hold for all substrings $\mathbf{k} = q_i, \ldots, q_j$ and $\mathbf{k}' = q_{i'}, \ldots, q_{j'}$ of $\mathbf{q}$.

1. If there is an a-concept $\gamma \in \mathcal{S}^{\mathrm{a}}$ and a record $r^\gamma \in I$ where $\mathbf{k} \propto_\gamma \mathbf{k}' \propto_\gamma \mathit{txt}(r)$, then $[i, j] \subseteq [i', j']$.
2. If there are dummy parse trees t and t' for $\mathbf{k}$ and $\mathbf{k}'$, respectively, where $\mathrm{label}(\mathrm{root}(t)) = \mathrm{label}(\mathrm{root}(t'))$, then $[i, j] \cap [i', j'] \neq \emptyset$.

As an example, recall the search query *tim hortons timothy* of Figure 4. This query is not simple if *tim* $\propto_{\mathsf{person}}$ *timothy* and I contains a person record that matches *timothy*, because then Condition 1 is violated, as $[1, 1]$ (which is the span containing *tim*) is not contained in $[3, 3]$ (which is the span containing *timothy*). Note that $\mathbf{k}$ = *tim* and $\mathbf{k}'$ = *tim hortons* do not violate Condition 1, since $[1, 1] \subseteq [1, 2]$. As another example, consider the query $\mathbf{q}$ = *phone number*, and suppose that Γ contains only two rules: phoneNum $\rightarrow$ *phone* and phoneNum $\rightarrow$ *phone number*. With these two rules, Condition 2 is not violated, since $[1, 1] \cap [1, 2] \neq \emptyset$. However, if we add the rule phoneNum $\rightarrow$ *number*, then Condition 2 is violated since $[1, 1]$ and $[2, 2]$ are disjoint. The following theorem shows that simplicity of the query facilitates ranked enumeration.

THEOREM 5.4. *Let $(\mathcal{S}, I, \Gamma)$ be an SDBS, and let $\mathbf{q} \in \mathbf{T}^+$ be a search query, such that $\mathbf{q}$ is simple. The data-nonempty parses for $\mathbf{q}$ can be enumerated by decreasing specificity with polynomial delay.*

The proof of Theorem 5.4 is the most involved of any theorem in this paper. This proof presents an algorithm that builds on the algorithm EnumDataNonempty of Figure 3, with two main differences. First, we use constraints that are more general than those defined in Section 3 (roughly

speaking, the new constraints involve *containment*, rather than *equality*, of spans within the search query). Second, the constraint that corresponds to that of Line 7 is carefully chosen, and the crux of the proof is in showing that this choice guarantees the desired order. An important note about the algorithm is that it enumerates all the parses even if the query is not simple; of course, the order is not guaranteed then, but we highly believe that the resulting order is a good heuristic. Finally, we note that ranked enumeration by means of constraints stands behind the method of Murty-Lawler-Yen [18, 22, 29] (which has been used in the context of keyword search [8, 14]). However, that method heavily builds on the fact that the order is complete (i.e., every two elements are comparable), and is not suitable to handle a preorder (as required here).

6. CONCLUDING REMARKS

In an SDBS, a keyword-search query is assigned parses (interpretations), within a contextual schema, where a parse is evaluated over the database instance as a structured database query. We gave a special consideration to the notion of specificity of parses, which is gracefully captured by the SDBS framework using the standard notion of query containment. We showed that this model is tractable, in the sense that the parses can be efficiently generated, and containment can be efficiently decided. Furthermore, parses can be enumerated incrementally in decreasing specificity in the case of simple search queries (but not in the general case, assuming P $\neq$ NP). In future work, we plan to implement the SDBS framework and evaluate its benefits within a full-scale search engine. We also plan to investigate principled ways of *quantifying* the semantic value of parses. Such a quantification may be realized by weighting the concepts, the rules (in the spirit of the known notion of a *weighted* or *stochastic* context-free grammar), and the substrings of the search query (e.g., by incorporating their *domain frequency*).

7. REFERENCES

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, and S. Sudarshan. BANKS: Browsing and keyword searching in relational databases. In *VLDB*, pages 1083–1086. Morgan Kaufmann, 2002.

[2] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *VLDB J.*, 11(4):315–331, 2002.

[3] J. Bear, D. J. Israel, J. Petit, and D. L. Martin. Using information extraction to improve document retrieval. In *TREC*, pages 367–377, 1997.

[4] A. Z. Broder. A taxonomy of Web search. *SIGIR Forum*, 36(2):3–10, 2002.

[5] Y. Chen, W. Wang, Z. Liu, and X. Lin. Keyword search on structured and semi-structured data. In *SIGMOD Conference*, pages 1005–1010. ACM, 2009.

[6] M. R. Garey and D. S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM J. Comput.*, 6(3):416–426, 1977.

[7] M. R. Garey and D. S. Johnson. "Strong" NP-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, 1978.

[8] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD Conference*, pages 927–940. ACM, 2008.

[9] M. A. Hearst. Direction-based text interpretation as an information access refinement. In P. S. Jacobs, editor, *Text-Based Intelligent Systems: Current Research and Practice in Information Extraction and Retrieval*, pages 257–274. Erlbaum, Hillsdale, 1992.

[10] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, pages 670–681. Morgan Kaufmann, 2002.

[11] D. Johnson, M. Yannakakis, and C. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.

[12] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516. ACM, 2005.

[13] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar semantic search: a database approach to information retrieval. In *SIGMOD Conference*, pages 790–792. ACM, 2006.

[14] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, pages 173–182. ACM, 2006.

[15] B. Kimelfeld and Y. Sagiv. Efficiently enumerating results of keyword search over data graphs. *Inf. Syst.*, 33(4-5):335–359, 2008.

[16] B. Klimt and Y. Yang. Introducing the Enron corpus. In *CEAS*, 2004.

[17] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. SystemT: a system for declarative information extraction. *SIGMOD Record*, 37(4):7–13, 2008.

[18] E. L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1972.

[19] D. D. Lewis. Text representation for intelligent text retrieval: A classification-oriented view. In P. S. Jacobs, editor, *Text-Based Intelligent Systems: Current Research and Practice in Information Extraction and Retrieval*, pages 179–197. Erlbaum, Hillsdale, 1992.

[20] Y. Li, R. Krishnamurthy, S.Vaithyanathan, and H. V. Jagadish. Getting work done on the Web: supporting transactional queries. In *SIGIR*, pages 557–564. ACM, 2006.

[21] Y. Luo, W. Wang, and X. Lin. SPARK: A keyword search engine on relational databases. In *ICDE*, pages 1552–1555. IEEE, 2008.

[22] K. G. Murty. An algorithm for ranking all the assignments in order of increasing costs. *Operations Research*, 16:682–687, 1968.

[23] B. Pang and L. Lee. Using very simple statistics for review search: An exploration. In *Proceedings of COLING: Companion volume: Posters*, pages 73–76, 2008.

[24] G. Petasis, V. Karkaletsis, G. Paliouras, and C. D. Spyropoulos. Learning context-free grammars to extract relations from text. In *ECAI*, volume 178 of

Frontiers in Artificial Intelligence and Applications, pages 303–307. IOS Press, 2008.

[25] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: the power of RDBMS. In *SIGMOD Conference*, pages 681–694. ACM, 2009.

[26] Y. Qiu and H.-P. Frei. Concept based query expansion. In *SIGIR*, pages 160–169. ACM, 1993.

[27] F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, and S. Vaithyanathan. An algebraic approach to rule-based information extraction. In *ICDE*, pages 933–942. IEEE, 2008.

[28] K. Sparck Jones. Assumptions and issues in text-based retrieval. In P. S. Jacobs, editor, *Text-Based Intelligent Systems: Current Research and Practice in Information Extraction and Retrieval*, pages 157–177. Erlbaum, Hillsdale, 1992.

[29] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.

[30] H. Zhu, S. Raghavan, S. Vaithyanathan, and A. Löser. Navigating the intranet with high precision. In *WWW*, pages 491–500. ACM, 2007.