ABSTRACT

An increasingly important class of keyword search tasks are those where users are looking for a specific piece of information buried within a few documents in a large collection. Examples include searching for someone’s phone number, the schedule for a meeting, or a package tracking URL, within a personal email collection. We refer to such tasks as “precision-oriented search tasks”. While modern information extraction techniques can be used to extract the concepts involved in these tasks (persons, phone numbers, schedules, etc.), since users only provide keywords as input, the problem of identifying the documents that contain the information of interest remains a challenge.

In this paper, we propose a solution to this problem based on the concept of automatically generating “interpretations” of keyword queries. Interpretations are precise structured queries, over the extracted concepts, that model the real intent behind a keyword query. We formalize the notion of interpretations and address the challenges in identifying the most likely interpretations for a given keyword query. We validate the benefits of our approach by presenting results both from a user study and experimental comparisons against an alternative approach.

1. INTRODUCTION

A popular and important class of keyword search tasks are those where users are looking for a specific piece of information that is present in a small number of documents in a corpus. For instance, several typical search queries over personal email collections fall into this category – queries to locate someone’s phone number, the agenda for a meeting, the URL to initiate a business process, instructions to install a piece of software, and so on.

In this paper, we address the problem of building a retrieval system that is specifically targeted to answer search tasks that fit the above description (hereafter referred to as precision-oriented search tasks). Prior work has addressed specific instances of precision-oriented search. For example, the task of “Home Page Finding” [26] is motivated by Web search queries where users provide the name of an entity (a person name, name of a research project, etc.) and are looking for the home page of that entity as opposed to pages that simply mention the name. Recent work on answering navigational and transactional queries in corporate intranets [17, 19, 15] also targets the retrieval of one or two high quality result pages in response to a keyword query. There has also been work in the IR community on the problem of “Known Item Finding” in email, where the goal is to search for a specific message or thread within a corpus of email messages [31].

However, in all of these cases, the proposed techniques have been specific to a particular type of document collection and to a specific class of search tasks. To be truly applicable for a broad spectrum of enterprise search applications and document collections, a generic precision-oriented retrieval infrastructure is required. The need for such infrastructure is underscored by recent studies that reveal that the average corporate employee spends 25–35% of their productive time searching for information [28] to perform his/her routine tasks.

In the following section, we motivate the problem of precision-oriented search using examples drawn from popular search queries on the Web.

1.1 Precision-oriented Search

Consider the keyword query ‘george w bush age’. The top ranked result for this query on the Google search engine is a snippet that displays the date of birth of President George W. Bush along with the search engine interpreted the query ‘ord sfo’ brings up a form for booking a flight from Chicago (ORD) to San Francisco (SFO) whereas the query ‘jeff gates seattle’ brings up the phone book results for the person Jeff Gates in the city of Seattle.

These queries illustrate a paradigm shift in the way Web search engines process keyword queries. In addition to results from content and link-based ranking algorithms, search engines are beginning to attach specific semantics to user search queries. Indeed, to produce the top-ranked result for the query ‘george w bush age’, the search engine interpreted ‘george w bush’ as a person and concluded that the intent of the user was to retrieve documents mentioning George W Bush’s age or date of birth.

While it is clear that Web search engines are deploying specialized infrastructure to handle precision-oriented search queries, there are several challenges in developing a similar solution for the enterprise. To illustrate these challenges, it is useful to view the problem of precision-oriented retrieval as consisting of three steps:

• Definition of the concept space – deciding the set of concepts that the search system is going to recognize and support. On the Web, the concept space is primarily determined by topics of interest to a vast majority of users (famous persons, movies, restaurants, flights, etc.) and by economic factors that dictate the set of concepts most likely to generate advertising revenue.

On the other hand, both the document collections and the set of concepts of interest vary widely across enterprises and even within different segments of the same enterprise.

• Concept extraction – identifying and extracting occurrences of the chosen concepts in the document collection: Both in the enterprise and on the Web, concepts can be extracted either through automatic information extraction techniques (e.g., extracting the date of birth from web pages) or by exploiting available structured databases (e.g., phone books or yellow pages). The only distinction is that content producers who wish to fea-
Query interpretation – interpreting the user’s keyword query in terms of the concepts: While the precise mechanism used by Google to interpret user queries is not publicly known, a reasonable hypothesis is that specific patterns in the input queries are recognized and mapped to a pre-specified query over the concepts (e.g., the pattern \[\text{person P} \text{ age}\] is mapped to “get me Web pages where P’s date of birth is mentioned”, the pattern \[\text{person P} \text{ city C}\] is mapped to “get me the phone book entry for P living in C”, etc.). Such a strategy is reasonable for a fixed and controlled concept space. However, to be useful in the enterprise, a more generic query interpretation mechanism that can work over an arbitrary concept space is necessary.

1.2 Solution Approach

Our approach to query interpretation is based on the idea of generating one or more precise structured queries from a given keyword query. We refer to each generated precise query as an interpretation. To illustrate this approach, consider the scenario of keyword search over a corpus of email messages that includes the two emails shown in Figure 1. Suppose two users \(u_1\) and \(u_2\) submit the same keyword query \`tom phone'\. It is possible that even though their queries are identical, the two users might have very different information needs. For instance, \(u_1\) may be interested in \"emails from tom that mention his phone number\" whereas \(u_2\) may be interested in \"emails sent by tom that mention some phone number\". Therefore, both documents in Figure 1 are relevant for \(u_2\) whereas only Document 1 is relevant for \(u_1\).

In general, there are two factors that determine whether a document is a valid result for a precision-oriented search query: (i) the intended search task of the user submitting the query, and (ii) the semantics of the document content. To capture document semantics, we resort to information extraction techniques for identifying mentions of entities and relationships in text. Each mention is called an annotation and a structured data store containing all of the extracted annotations is called an annotation store. Figure 2 is the scheme for the annotation store that we use as the running example in this paper. Each oval in the figure represents an entity or relationship that has been extracted by analyzing email messages (the attributes for a type are listed in parentheses below the oval). For instance, the text segment 713-223-3426 in Document 1 (Figure 1) has been recognized as an instance of a PhoneNumber, Tom Shelton as an instance of the concept Author, and these two instances are linked via the relationship AuthorPhone.

Given such a store, the following interpretation\(^2\) represents \(u_1\)’s search task:

\[
\text{Query } I_1. \\
\text{select } E \text{ from AuthorPhone AP, Email E where } E = AP.doc \text{ AND match(AP.author.name, 'tom')}
\]

A similar query involving Author and PhoneNumber can be written for \(u_2\)’s search task. Thus, interpretations (structured queries over annotations) provide a powerful mechanism for representing the search task that underlies a precision-oriented search query.

Ambiguity and two-stage retrieval. As the above example demonstrates, there may be multiple potential interpretations for the same keyword query. Table 1 lists other search tasks for which a user may submit the query \`tom phone'\ (restricted to tasks expressible in terms of the concepts in Figure 2). Notice that even with a relatively small concept space, there is considerable ambiguity in how a particular query term is to be interpreted. For example, \`tom' could refer to the name of a person, the author of an email, or the name of an organization (e.g., Tom’s Hardware). Another source of ambiguity is when different users use the same keyword to refer to different concepts (\`phone' could either be a phone number or indicate conference call information).

To deal with this ambiguity, we propose the following two-stage retrieval approach. In the first step, in response to a keyword query, the user is presented with a list of the most likely interpretations of that query – the fact that such a list can be effectively computed will be demonstrated algorithmically in Sections 3 and 4 and experimentally in Section 5. Once the user has chosen a specific interpretation of interest, only documents matching that interpretation are presented to the user.

1.3 Why Interpretations?

To understand the effectiveness of interpretations and the two-stage retrieval model for precision-oriented search, let us consider some alternate approaches.

The simplest alternate approach is to employ an integrated rep-
Table 1: Results for some precision-oriented tasks corresponding to query ‘tom phone’

<table>
<thead>
<tr>
<th>Potential tasks</th>
<th>Number of results</th>
<th>Number of Unique results</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1: Emails containing (person) Tom’s phone number</td>
<td>51</td>
<td>15</td>
</tr>
<tr>
<td>T2: Emails from (person) Tom that contain a phone number</td>
<td>72</td>
<td>42</td>
</tr>
<tr>
<td>T3: Emails containing a (person) Tom and a phone number</td>
<td>439</td>
<td>119</td>
</tr>
<tr>
<td>T4: Emails containing an (organization) Tom and a phone number</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>T5: Emails from (person) Tom that contain conference call information</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T6: Emails containing a (person) Tom and conference call information</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

representation of documents and annotations in a form that can be indexed by a retrieval system. This enables keyword queries to be directly executed over the index without resorting to interpretations. For instance, consider the email message and the set of extracted annotations shown in Figure 1.2. Figures 1.2–1.2 show three possible ways in which this annotated email message can be represented. Figure 1.2 depicts a simple text-based representation in which the original document text has been expanded to include the names of the identified concepts (e.g., the word PhoneNumber, shown underlined in the figure, has been inserted before the actual instance of a phone number). More expressive XML-based representations are shown in Figures 1.2 and 1.2.

Each of these representations can be efficiently indexed to support keyword search queries. For the text-based representation, a standard inverted-index based retrieval engine will suffice whereas the XML representations can leverage work in the area of XML-IR [2, 3, 11, 27, 30].

Let us now revisit our running example query ‘tom phone’ to understand how these approaches will perform. Imagine a user who submits this query with the intention of finding out the phone number of a person named Tom. As shown in Table 1, there are a total of 51 emails in the Enron corpus that mention Tom’s phone number but there are only 15 distinct Tom’s in that set (assuming persons are distinguished by their full names). The phone number of a specific person named Tom is therefore mentioned, on average, in about 3 documents. On the other hand, in the total corpus, there are 598 emails that satisfy the following conditions: (i) the word tom is mentioned (either as a keyword, a person, an organization, or an author), and (ii) the word phone, a PhoneNumber annotation, or a ConferenceCall annotation are present. Since all of the approaches based on direct execution of keyword queries over a text/XML index use documents as the unit of ranking and retrieval, the burden on the ranking algorithms of these systems is one of accurately identifying 3 documents from the 598 documents that are potentially matches to the query.

However, using our two-stage approach, the two ranking problems are: (i) ranking interpretations, and (ii) ranking documents that match a specific interpretation – both of which are smaller than the overall ranking problem described earlier. In summary, the key insight of our approach is as follows: by using interpretations to explicitly represent search tasks and allowing users to pick a specific interpretation, the search engine receives additional help from the user in disambiguating the keyword query.

1.4 Contributions

The primary contributions of this paper are as follows:

- We propose the use of interpretations and an associated two-stage retrieval model for addressing precision-oriented search tasks.
- We define the concept of justification to formally characterize the relationship between a keyword query and each of its valid interpretations. (Section 3)
- We introduce the semantic optimizer – the runtime component of a precision-oriented retrieval system that is responsible for generating a ranked list of the most likely interpretations of a keyword query. We present detailed algorithms for each stage of semantic optimization. (Section 4)
- Finally, we validate the effectiveness of our approach through a user study and extensive experiments using the Enron email corpus [8]. (Section 5)
2. ARCHITECTURE

We have implemented a generic precision-oriented search engine conforming to the architecture shown in Figures 4 and 5. While the specific examples and experimental results in this paper are based on the deployment of our engine over the Enron corpus, the same architecture has been successfully deployed over diverse document collections ranging from blog pages to Intranet Web collections.

As shown in Figure 4, offline processing begins with the annotation process in which documents are analyzed by several information extraction engines to produce annotations (see Appendix A). Next, an index build process scans the resulting annotation store to produce two indexes: (i) a translation index to assist in the generation of interpretations, and (ii) a data index over which individual interpretations are executed to retrieve documents. The runtime architecture consists of two components—the semantic optimizer and the user interaction engine. The semantic optimizer is responsible for generating a ranked list of the most likely interpretations of a given keyword query. The user interaction engine is a suite of tools for displaying interpretations and result documents to the user. In this paper, our focus will be on the semantic optimizer.

3. JUSTIFIED INTERPRETATIONS

Before we describe how the semantic optimizer accomplishes the task of producing ranked list of interpretations for a keyword query, we must first characterize precisely the space of interpretations that are under consideration. To this end, we establish the following: (i) the class of structured queries that we use to represent interpretations, and (ii) the concept of justification to formally characterize when a structured query from the above class is a valid interpretation of a given keyword query.

To help with the formalism in this section, it is convenient to view the annotation store in the following terms:

**Definition 1 (Annotation Store).** An annotation store \( S = (T, O, D) \) consists of a set of types \( T \), a set of objects \( O \), and a unique distinguished type \( D \in T \), such that for every \( x \in O \), \( \text{type}(x) \in T \). Further, for every object \( x \in O \), either \( \text{type}(x) = D \) or there exists an attribute \( d \) with \( \text{type}(x.d) = D \).

In the above, \( \text{type}(x) \) denotes the type of object \( x \), \( D \) is the document type, and every other type in \( S \) is an annotation type. Notice that every annotation object contains a doc attribute that refers to the document from which that object was extracted. A path in this annotation store is any expression of the \( T.a_1 \ldots a_m \) where \( a_i \) is a valid attribute of type \( T.a_i \) and \( a_2 \) is a valid attribute of type \( \text{type}(T,a_1) \), and so on.

3.1 Class of Interpretation Queries

The class of queries used to represent interpretations over an annotation store \( S = (T, O, D) \) is shown in Figure 6. In the figure, \( D \) is the document type, each \( T_i \in T \) is an annotation type, each \( p_i \) is a valid path beginning with \( D \) or with one of the \( T_i \)’s, each \( c_i \) is a string constant, and the “match” predicate denotes matches over text values.

For example, given the annotation store in Figure 2, the following two queries are within the class defined in Figure 6:

**Query \( I_2 \).**

\[
\text{select } E \\
\text{from } Email E, \text{Author A, PhoneNumber T} \\
\text{where } E = T.doc \text{ AND } E = A.doc \text{ AND } \text{match}(A.name, ‘tom’) \\
\]

**Query \( I_3 \).**

\[
\text{select } E \\
\text{from } Email E, \text{Signature S} \\
\text{where } E = S.doc \text{ AND } \text{match}(S.person.name, ‘tom’) \\
\]

We use the convention that the special string “**” matches any non-null value. Therefore, in the above query, the match predicate \( \text{match}(S.phone,**) \) enforces the condition that \( S.phone \) must not be null.

We have made several careful choices in defining the class of queries used to represent interpretations. Specifically, queries always return documents and whether a particular document is in the result depends only on the properties of that document and its annotations. Value-based joins, joins across documents, and joins across annotations that are extracted from different documents are all disallowed. We believe that these choices represent a fair trade-off between the expressivity required to model a useful class of precision-oriented search tasks and the restrictions that enable efficient enumeration algorithms (cf. Section 4.1) and index structures (see below).

**Data Index.** The class of queries in Figure 6 involve structural predicates (i.e., path expressions) as well as text matching predicates. To efficiently execute such queries, we build a data index over the annotation store using the indexing capabilities of IBM’s OmniFind product [22]. OmniFind’s XML indexes combine support for XPath with support for all of the standard features of a text retrieval engine (stemming, multi-word tokenization, stop word dictionaries, etc.). To use OmniFind’s XPath support, we generate a collection of XML documents, one for each original document, using the outlined representation shown in Figure 1.2. While a detailed explanation of the query translation algorithms is omitted in the interest of space, it is easy to show that any query that fits the template of Figure 6 can be translated into an XPath query over this XML collection. For instance, query \( I_2 \) can be translated into

\[
\text{select } d \\
\text{from } D, T_1.t_1, T_2.t_2, ..., T_n.t_n \\
\text{where } t_1.doc = d \text{ AND } ... \text{ AND } t_n.doc = d \text{ AND } \text{match}(p_1,c_1) \text{ AND } ... \text{ AND } \text{match}(p_m,c_m) \\
\]

The “match” predicate encapsulates all the aspects of approximate matching that is common in text retrieval systems including stemming, stop word elimination, relaxed case sensitivity, etc.
shows that the keyword 'tom' has a value match with Author.name, Person.name, Signature.person.name and AuthorPhone.author.name indicating that 'tom' has appeared as the name of the author of an email, as the name of a person who was mentioned in the signature block of an email, and so on.

In our implementation, the TypePath index was implemented as a custom main-memory data structure whereas the Value index was implemented using a standard disk-based inverted index.

### 3.3 Justification

In Section 3.1, we defined a broad class of queries for representing interpretations but did not specify when a particular interpretation from this class is “valid” for a given keyword query. It is precisely this connection between a keyword query and a corresponding valid interpretation that we formalize using the concept of justification. In this context, the concept of interpretation components is useful.

**Definition 2. (Interpretation Components).** The components of an interpretation $I$ that belongs to the generic query class described in Figure 6 is defined as $\text{comps}(I) = \text{types}(I) \cup \text{preds}(I)$, where

$$
\text{types}(I) = \{(D, d), (T_1, t_1), (T_2, t_2), \ldots, (T_n, t_n)\},
$$

$$
\text{preds}(I) = \{(p_1, c_1), (p_2, c_2), \ldots, (p_m, c_m)\}.
$$

Any interpretation that fits the template of Figure 6 is fully specified by listing its components. For example, query $I_2$ is fully specified by the two sets $\{(\text{Email, E}), (\text{PhoneNumber, T}), (\text{Author, A})\}$ and $\{(\text{A.name}, \text{tom})\}$. Using the notion of interpretation components, the intuition underlying justification is as follows: if $I$ is a justified interpretation of some keyword query $k = \{k_1, \ldots, k_r\}$, every component of $I$ must be “explained” by a match between one of the query terms ($k_i$) and the annotation store. For example, $I_2$ is a justified interpretation of 'tom phone' since the predicate match(Author.name, 'tom') is explained by a value match between 'tom' and Author.name whereas the type PhoneNumber is explained by a type match between 'phone' and PhoneNumber.

It is convenient to capture the relationship between a keyword query and an interpretation using a justification graph. Given a keyword query $k = \{k_1, k_2, \ldots, k_r\}$ and an interpretation $I$ over an annotation store $\mathcal{S}$, the justification graph $G_{k, I}^\mathcal{S}$ is constructed as follows (justification graphs for $I_2$ and $I_3$ are given in Figure 8):

- Create a distinct vertex corresponding to each element of $k \cup \text{comps}(q)$.
- Create an edge labeled “A” between $k_i$ and $(p, c) \in \text{preds}(q)$ if $(p, c)$ can be explained by either a path match or value match on $k_i$. For example, in interpretation $I_3$ from Section 3.1, predicate (S.person.name, tom) is explained by keyword 'tom' be-
cause of a value match, and (S.phone, *) is explained by ‘phone’ because of a path match.

- Create an edge labeled ‘A’ between $k_i$ and $(T, t) \in \text{types}(q)$ if $T$ is a type match for $k_i$ (e.g., in query $I_2$, the type entry (PhoneNumber, T) is explained by keyword ‘phone’ through a type match).
- Create an edge labeled “B” between each $k_i$ and the type entry $(D, d)$ corresponding to the document type.
- Create an edge labeled “B” between $k_i$ and $(T, t) \in \text{types}(q)$ if there exists $(p, c) \in \text{preds}(q)$ such that there is an A-edge between $k_i$ and $(p, c)$ and path $p$ begins with $t$. For example, in query $I_2$, there is a B-edge between ‘tom’ and type entry (Signature, S) since there is an A-edge between ‘tom’ and (S.person.name, ‘tom’).

$G_{S,k}^1$ is a bipartite graph over the two disjoint vertex sets: $k$, the set of keyword vertices, and $\text{comps}(q)$, the set of component vertices. Edges labeled “A” are incident on components that directly “explain” keywords whereas edges labeled “B” are incident on components that “indirectly explain” keywords. We can now formally characterize a justified interpretation as follows:

**Definition 3.** Given a keyword query $k = \{k_1, \ldots, k_r\}$ and an annotation store $S$, $I$ is a justified interpretation of $k$ under $S$ iff the degree of every component vertex in $G_{S,k}^1$ is at least one.

**Covering and Matching Interpretations.** Definition 3.3 does not require all of the keywords in a query to participate in justification. By placing the restriction that every keyword must be involved in justification (i.e., assuming an implicit AND across all the keywords in the query), we obtain the set of covering interpretations.

**Definition 4.** (Covering Interpretation). Interpretation $I$ is a covering interpretation of a keyword query $k$ under $S$ iff the degree of every vertex in $G_{S,k}^1$ is at least one.

Note that in a covering interpretation, it is possible for a keyword to be involved in multiple justifications. For example, consider the following interpretation for our canonical query ‘tom phone’ – retrieve emails from tom that contain tom’s signature with a telephone number. Notice that in this query, the keyword ‘tom’ will be involved in the justification of two predicates – one involving Author.name and another involving Signature.person.name.

For a two keyword query $k = \{k_1, k_2\}$, let $n_1$ and $n_2$ be the total number of matches returned by the translation index for $k_1$ and $k_2$ respectively. It is easy to see that the number of covering interpretations is $O(2^{n_1} + n_2)$. To obtain a more tractable set of interpretations, we impose a further restriction that a keyword must justify exactly one component.

**Definition 5.** (Matching Interpretation.). Query $q$ is a matching interpretation of a keyword query $k$ under $S$ iff there exists a matching $M$ on $G_{S,k}^1$ such that (i) $M$ includes all edges labeled “A”, (ii) $M$ covers all keyword vertices, and (iii) there is at least one edge labeled “B” incident on every vertex not covered by $M$.

It is easy to verify that queries $I_1$, $I_2$, and $I_3$ are all matching interpretations of the keyword query ‘tom phone’. In our implementation, we restricted the interpretation space to the set of matching interpretations of a keyword query.

**Procedure ConstructQuery($S$)**

// $S$ is a set of translation index matches

Let $q$ be a query that conforms to the template in Figure 6.

Set $\text{types}(q) = \{\}$ and $\text{preds}(q) = \{\}$.

For each type match in $S$,

- Add an entry to $\text{types}(q)$.

For each value and path match in $S$,

- Add an entry to $\text{types}(q)$ and a corresponding entry to $\text{preds}(q)$.

Add the document type to $\text{types}(q)$.

Return $q$.

**Figure 9: Constructing an interpretation**

**Procedure TypeMerge($q$)**

foreach pair $(T, t_i), (T, t_j) \in \text{types}(q)$

let $q'$ be the query obtained by modifying $q$ as follows.

- The entry $(T, t_i)$ is removed from $\text{types}(q)$.
- All references to $t_j$ in the predicates are replaced with $t_i$.

$Q = Q \cup \{q'\}.$

foreach $q_i \in Q$,

return $Q$.

**Figure 10: Performing type merge**

4. **Semantic Optimizer**

In this section, we describe each of the three stages of the semantic optimizer shown in Figure 5.

4.1 Enumeration

In this section, we present algorithms for enumerating the set of justified, covering, and matching interpretations of a keyword query. These enumeration algorithms use two subroutines called ConstructQuery and TypeMerge that are listed in Figures 9 and 10 respectively.

Subroutine ConstructQuery constructs an interpretation from a given set of type, path, and value matches. Essentially, for each value or path match, ConstructQuery adds a corresponding match predicate in the where clause, and for a type match, an entry in the from clause. For instance, given Author.name is a value match for tom and PhoneNumber is a type match for phone, ConstructQuery produces query $I_2$ from Section 3.1. Since ConstructQuery assigns a new entry in the from clause for every match, given the matches Signature.person.name with tom and Signature.phone with phone, the resulting interpretation will be:

**Query $I_4$**

select E from Email E, Signature S1, Signature S2
where $E = S1.doc$ AND $E = S2.doc$ AND
match(S1.person.name, tom) AND match(S2.phone, *)

Note that the above interpretation involves two Signature entries. Therefore, this interpretation will also match emails that happen to have two signatures – one with tom’s name and another containing a telephone number. While the above interpretation is certainly valid, a semantically stronger and more likely interpretation is one where a single Signature instance satisfies both conditions. Forcing this condition (essentially setting S1 = S2 in the above) will result in the interpretation $I_3$ that we have seen before in Section 3.1. In order to automatically generate such interpretations as part of enumeration, we must fuse together pairs of instances of the same type as described in the TypeMerge subroutine.

- It is possible for an email that contains a forwarded message to have two signatures
procedure Enumerate\((k = \{k_1, \ldots, k_r\}, S)\)

Compute \(\Psi(k)\), the union of translation index matches for all keywords in \(k\)

for each \(S \subseteq \Psi(k)\) such that \(S\) is “valid”

\(Q = \psi \cup \text{ConstructQuery}(S)\)

while (\(Q\) has not reached fixed point)

\(Q = Q \cup \text{TypeMerge}(q)\).

**Figure 11: Generic algorithm for enumerating interpretations**

Using these subroutines, algorithms for enumerating the set of justified, matching, and covering interpretations of a keyword query \(k = \{k_1, k_2, \ldots, k_r\}\) have the following general form (see Figure 11):

- Probe the translation index using each \(k_i\) to retrieve the set of type, path, and value matches for \(k_i\). Let \(\Psi(k)\) be the union of these matches across all \(k_i\).
- Choose a valid subset of matches from the set \(\Psi(k)\) where valid is defined as follows:
  - Justified \(\rightarrow\) any non-empty subset of \(\Psi(k)\)
  - Covering \(\rightarrow\) subset must contain at least one element for each keyword
  - Matching \(\rightarrow\) subset must contain exactly one element for each keyword
- Construct an interpretation based on the chosen matches
- Repeat the previous step by choosing a different valid subset
- Stop when all possible valid subsets have been considered
- Do type merging repeatedly until no more are possible

**4.2 Pruning**

The enumeration algorithms presented in the previous section are generic and intended to work on annotation stores produced from arbitrary document collections. As a result, they do not reflect any domain-specific constraints and properties that might hold true for a particular annotation store. Such constraints may be used to recognize that certain interpretations are either invalid or guaranteed not to return any matching documents. The pruning stage of the semantic optimizer (Figure 5) uses a rule-driven mechanism to discard such interpretations. We illustrate using two such pruning rules in this section.

To motivate the first rule, consider the single word query ‘tom’. Since there is a value match for ‘tom’ on AuthorPhone.person.name, one of the interpretations generated for this query by the algorithm in Figure 11 will be:

**Query 1:**

select \(E\) from Email E, AuthorPhone AF where \(E = AF.doc \text{AND match(AF.person.name, ‘tom’)}\)

The above interpretation (“return emails sent by tom that mention his phone number”) implicitly involves the concept of a phone number even though there is no such indication in the keyword query ‘tom’. This issue arises because a “binary relationship” such as AuthorPhone is being used even when the keyword query itself does not reflect that relationship. Similar problems arise with the other relationships such as PersonPhone (Figure 2). Thus, generalizing from this example, we propose the following rule:

**RULE 1** (RELATIONSHIP PRUNING RULE). Discard any interpretation in which an instance of a binary relationship is justified by fewer than two keywords.

To motivate the second pruning rule, consider the following in-
Assumption A2. To locate the specific document(s) of interest, the user must examine, on average, half the number of documents in the result list of the chosen interpretation.

Assumption A3. We assume that there is a fixed cost for examining any document and further, that the cost of examining an interpretation is significantly smaller than the cost of examining all of its results.

To fully qualify A1, we must define when an interpretation is of interest to a user. Since the eventual goal of the user is to locate documents, our definition of “interpretation of interest” is based on query containment:

**Definition 6 (Interpretation of Interest).** We say that interpretation \( I \) is of interest to a user whose intended interpretation is \( I_u \) if \( I_u \subseteq I \) (where \( \subseteq \) is the symbol for query containment).

Typically, pairs of interpretations in which one is contained in another are produced primarily due to the presence of “relationships” in the concept space. For instance, in Figure 2, since AuthorPhone is a relationship, interpretations \( I_1 \) (“emails from tom that mention his phone number”) is contained in interpretation \( I_2 \) (“emails from tom that mention some phone number”).

Cost function. Let \( L = \text{SemOpt}(k) \) represent the ordered list of interpretations produced by the semantic optimizer in response to a keyword query \( k \). Let \( P_L(A) \in [1...|L|] \) denote the position of interpretation \( A \) in list \( L \). Let \( \mu_I(I) \) denote the interpretation in \( L \) chosen by an user whose search task is \( I \). By Assumption A1, any user whose intent is \( I \) must examine all the interpretations up to and including the interpretation in position \( P_L(I) \). Therefore, the cost involved in navigating \( L \) is proportional to \( P_L(I) \).

Using Assumption A2, the cost in locating a document within the result set of the chosen interpretation is proportional to \( \frac{|R(L, I)|}{2} \) (where \( R(I) \) is the result set of interpretation \( I \)). Using \( \beta \) to denote the ratio between the cost of examining an interpretation and the cost of examining a document, the total cost of \( L \) for a user whose intended interpretation is \( I \) is given by:

\[
\beta P_L(I) + \frac{|R(I)|}{2} \tag{1}
\]

The true cost of \( L \) for the keyword query \( k \) is obtained by averaging the above expression across all possible intended interpretations. Let \( p(I|k) \) denote the probability that a user who supplied the keyword query \( k \) intended \( I \) (we refer to \( p \) as the intent distribution). Using this distribution, the overall cost function can be expressed as:

\[
\text{cost}(L, k) = \sum_I p(I|k) \left( \beta P_L(I) + \frac{|R(I)|}{2} \right) \tag{2}
\]

Using this cost function, we state two propositions (proof omitted for lack of space) involving the relative order between pairs of interpretations. These propositions naturally lead to rules for ranking interpretations.

**Proposition 1.** Given keyword query \( k \), consider a pair of interpretations \( I_1 \) and \( I_2 \) such that \( p(I_1|k) = p(I_2|k) \) and \( I_1 \not\subseteq I_2 \). Let \( L_1 \) and \( L_2 \) be two interpretation lists that are identical except that the positions of \( I_1 \) and \( I_2 \) are swapped. Also, let \( L_1 \) be the list where \( I_1 \) is ranked before \( I_2 \) and let \( D = P_{L_1}(I_2) - P_{L_1}(I_1) = P_{L_2}(I_1) - P_{L_2}(I_2) \). Then,

\[
\beta < \frac{|R(I_2)| - |R(I_1)|}{2D} \implies \text{cost}(L_1, k) < \text{cost}(L_2, k)
\]

Note that \( \frac{|R(I_2)| - |R(I_1)|}{2} \) is the additional number of documents to be examined because of the larger result set of \( I_2 \). However, \( D \) is the penalty (in terms of number of interpretations) paid for an incorrect ordering between \( I_1 \) and \( I_2 \). The proposition essentially states that as long as the ratio of these quantities is at least as large as \( \beta \), ranking the interpretation with the smaller result set higher reduces cost. In particular, if \( D = 1 \) and if we assume that the cost of examining a document and an interpretation are comparable (\( \beta \approx 1 \)), then ordering \( I_1 \) before \( I_2 \) is always more effective as long as \( R(I_2) \) is even marginally larger than \( R(I_1) \). We therefore adopt the following ranking rule:

**Rule 3 (Containment Ranking Rule).** Given a keyword query \( k \) and two interpretations \( I_1 \) and \( I_2 \) such that \( p(I_1|k) = p(I_2|k) \) and \( I_1 \not\subseteq I_2 \), rank \( I_1 \) above \( I_2 \).

To formulate our second proposition, the following definition is useful:

**Definition 7 (Disparate Interpretations).** Two interpretations \( I_1 \) and \( I_2 \) are disparate, denoted \( I_1 \not\sim I_2 \), if \( I_1 \) is not of interest to a user whose intent is \( I_2 \) and vice versa.

**Proposition 2.** Given keyword query \( k \) and a pair of interpretations \( I_1 \) and \( I_2 \) such that \( I_1 \not\sim I_2 \). Let \( L_1 \) and \( L_2 \) be two interpretation lists that are identical except that the positions of \( I_1 \) and \( I_2 \) are swapped and let \( L_2 \) be the list where \( I_1 \) is ranked before \( I_2 \). Then,

\[
p(I_1|k) > p(I_2|k) \implies \text{cost}(L_1, k) < \text{cost}(L_2, k)
\]

From Proposition 3.4, we derive the following ranking rule:

**Rule 4 (Disparate Ranking Rule).** Given a keyword query \( k \) and two interpretations \( I_1 \) and \( I_2 \) such that \( I_1 \not\sim I_2 \) and \( p(I_1|k) > p(I_2|k) \), rank \( I_1 \) above \( I_2 \).

Rules 4.3 and 4.3 provide the foundations for our ranking algorithm. However, to apply both rules, we need to be able to compare the \( p(I|k) \) values of pairs of interpretations. In general, accurate estimation of the intent distribution is non-trivial. We therefore use the following heuristic as a substitute.

**Semantic Value.** Consider the following list of interpretations for our canonical query ‘tom phone’:

1. **I_1**: retrieve emails that contain the words ‘tom’ and ‘phone’
2. **I_2**: retrieve emails from tom that contain the word ‘phone’
3. **I_3**: retrieve emails from tom that mention a phone number
4. **I_4**: retrieve emails from tom that mention his/her phone number

Notice that as we walk down this list, more keywords become associated with concepts. While \( I_1 \) is nothing more than plain keyword search, \( I_2 \) associates ‘tom’ with the concept of author whereas \( I_3 \) and \( I_4 \) further associate ‘phone’ with the concept of a phone number. Intuitively, each association of a keyword with a semantic concept increases the “semantic value” of the corresponding interpretation.

**Definition 8 (Semantic Value).** Given an interpretation \( I \) for the keyword query \( k \), let \( \gamma(I, k) \) denote the number of keyword vertices in \( G_k \) that have at least one edge to a component vertex \((T, t)\) where \( T \) is not the document type. We refer to \( \gamma(I, k) \) as the semantic value of \( I \) for query \( k \).
Essentially, $\gamma(I, k)$ measures the number of keywords in $k$ that are mapped to concepts as opposed to plain keywords. It is easy to see that for our earlier example with $k = \text{`tom phone'}, \gamma(I_1, k) = 0, \gamma(I_2, k) = 1,$ and $\gamma(I_3, k) = \gamma(I_4, k) = 2$.

We now make the following reasonable assumption that interpretations with higher semantic value are more likely to match users intention. Specifically,

**Assumption A4.** Given two interpretations $I_1$ and $I_2$ of a keyword query $k$, $\gamma(I_1, k) = \gamma(I_2, k) \implies \hat{p}(I_1|k) = \hat{p}(I_2|k)$ and $\gamma(I_1, k) > \gamma(I_2, k) \implies \hat{p}(I_1, k) > \hat{p}(I_2, k)$.

Thus, Rules 4.3 and 4.3 in conjunction with Assumption A4 provide a complete set of rules for ordering interpretations. When applied to a particular set of interpretations, if the rules only produce a partial order, ties are broken arbitrarily to generate a totally ordered list.

5. **EXPERIMENTS**

The use of an interpretation-based search engine (referred to as INTERP in this section) is predicated on the existence of a domain schema and associated annotators. We chose the Enron email collection and hand-crafted a domain schema (Figure 2). Our experimental set-up is broadly divided into two categories: (a) user study to evaluate the effectiveness of an INTERP system against a traditional keyword engine, and (b) experiments to study the various stages of the semantic optimizer.

To collect statistics, for both categories of experiments, we defined 6 tasks shown in Table 5. Although we believe our choice of tasks is representative of the real-world, our primary motivation was to evaluate the need for annotations and interpretations in precision-oriented search. Each task was based on a concept from the annotation schema (e.g., Task 1 has Schedule) that we refer to as the primary concept for the task. The complete task specification may have other potential concept matches (secondary concept) for certain user keyword queries. For example, Task 2 has Person as a secondary concept. Note that for all the tasks the primary concept match cannot be obtained directly by keyword search. As an example, in Tasks 3 and 4, the word “directions” does not appear in the email.

All experiments were conducted on a 4-way 2.8GHz Intel Xeon machine with 4GB of main memory running RedHat Linux. The Enron email data set from [8] consists of over 500,000 emails. However, after duplicate removal (based on author and body of the email), we were left with a collection of 251478 emails with an average size of about 900 bytes.

5.1 **User Study**

To compare the effectiveness of the INTERP approach against traditional keyword search, we compared our prototype with the Berkeley Enron Search interface [5], which supports fielded search over the enron corpus. (henceforth referred to as FIELD). The FIELD system consists of a Lucene index over three distinct email fields (Subject, From and Body). The INTERP UI consists of a simple keyword interface. Ranked interpretations are displayed to the user as English Questions (Figure 12). When a particular interpretation is selected by the user the corresponding result documents are listed. The overall statistics for the tasks are listed in Figure 14.

In the user study, 17 users performed a set of search tasks by issuing one or more keyword queries for each task. There were no restrictions on the number of queries that a user may issue per task, but a five minute time limit was imposed. Users were given the option to give up on a task earlier than the time limit if they felt that there was no reasonable hope of finding the answer. Times reported are wall-clock times for the duration of each task. For Task 2, the users were instructed to find as many answers as they could with reasonable effort.

The results are summarized in Figure 14. Notice how users spent about two minutes on average with INTERP to obtain a success rate of 84%, while they spent about three minutes on average with FIELD to obtain a success rate of 58%. This confirms the claim made in Section 1 that the combination of annotations and interpretations improves the results for precision-oriented tasks significantly.

A breakdown of the average time spent and the success rate per task are shown in Figures 5.1 and 5.1 respectively. We observe the following from these results:

- For tasks 2, 3, 4 and 5, users spent less time while using INTERP and obtained a better success rate as well. For task 1 they spent more time with INTERP to get a perfect success rate of 1.0 and for task 6, they spent more time with INTERP and got a lower success rate. For task 1, the query ‘omaha party’ was highly selective over the collection (similarly ‘Paul DFW’ and ‘Paul Dallas’ for task 6). Since the task specification had these keywords, user queries included them most of the time. This shows that when the user knows sufficient number of keywords that uniquely identify the document, traditional keyword search does well even for precision-oriented tasks. Otherwise, a more specialized approach is required to improve the success rate.
- For tasks 3, 4 and 5, the success rate with FIELD was much lower than with INTERP. These tasks involved looking for concepts with additional semantics (Directions and Conference-Call). While users tried keyword queries with various synonyms for these concepts (e.g., ‘wedding driving direction’ and ‘strate-
5.2 Evaluating the Role of Interpretations

We conducted a set of experiments to (a) measure the effectiveness of the semantic optimizer and (b) understand the value of interpretations in exploiting annotations during precision-oriented search.

5.2.1 Evaluating the Semantic Optimizer

A complete evaluation of the semantic optimizer should involve explicit evaluation of all three stages enumeration, pruning and ranking. We first report on the enumeration and pruning stages and then discuss about the ranking stage.

The left columns in Table 2 report average number of interpretations after the enumeration and the pruning stages. We make the following observations from these numbers.

- Enumerating the set of valid interpretations in a formal way results in a large number of interpretations for every task. This validates our claim that keywords in user queries when matched individually over the set of concepts may have a large number of interpretations.

- The pruning techniques are very efficient in bringing the number of interpretations down to a manageable value (up to an order of magnitude reduction). This supports our claim that generic pruning rules are an essential part of an interpretation-based architecture.

We next evaluated the effectiveness of the ranking stage of the semantic optimizer as follows. Recall from Section 4.3, that given a list of interpretations, the user picks the first interpretation of interest. We manually labeled the interpretations as (i) correct interpretations (that match the task description completely) (ii) interpretation of interest (whose query is guaranteed to contain the result for the task, but will return more results) and (iii) incorrect interpretations. We then computed the Mean Reciprocal Rank (MRR) for the list of interpretations returned by the system. Since “interpretations of interest” contain more results, we assigned a weight of 0.5 for them (correct interpretations were assigned a weight of 1.0).

Consider now the results of ranking presented in the right columns of Table 2. The numbers reported are the mean-reciprocal-rank (MRR) for the list of interpretations returned by the enumerate and ranking stages. Notice the significant improvement in MRR for the ranked interpretations over the enumeration stage. For the first three tasks, the correct interpretation was found in the first two interpretations. The lowest MRR was for task 5, where the average MRR was 0.003. This number was low due to two factors (a) users issued partial queries in many cases (e.g., ‘strategic’, ‘strategic planning’ and ‘corporate’), which do not result in any interpretation of interest, (b) the tie between the concepts PhoneNumber and ConferenceCall is broken arbitrarily in the ranking stage.

### Table 2: Effects of ranking and pruning

<table>
<thead>
<tr>
<th>Task</th>
<th>size of ( L )</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerate(E)</td>
<td>Prune(P)</td>
<td>(P-E)</td>
</tr>
<tr>
<td>1</td>
<td>87</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>117</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>78</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>5</td>
</tr>
</tbody>
</table>

### Figure 13: Tasks performed by users

![Tasks performed by users](image)

### Figure 14: User Study Parameters and Summary of Results

![User Study Parameters and Summary of Results](image)
• **Alternative QUERY EXPANSION approach:** The definition of a baseline approach that utilizes annotations during search is non-trivial, as it involves multiple aspects such as representation of annotations and how they are exploited in the alternative approach. We chose to retain the XML representation used by our approach and simulate the use of annotations through query expansion. We expand each token in the user query to include all the concept names that match that token. For example, the query ‘barbara phone’ gets expanded to the query ‘barbara (phone | <PhoneNumber> | <ConferenceCall>).’ Since the underlying engine used (Omnifind [22]) supported XML predicates in the query, we were able to expand the query in this fashion.

<table>
<thead>
<tr>
<th>Task</th>
<th>Worst-case</th>
<th>INTERP</th>
<th>QUERY EXPANSION</th>
<th>improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.502</td>
<td>0.312</td>
<td>60.90%</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.559</td>
<td>0.139</td>
<td>302.16%</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.585</td>
<td>0.001</td>
<td>58400.00%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.211</td>
<td>0.044</td>
<td>379.55%</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.087</td>
<td>0.013</td>
<td>569.23%</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.611</td>
<td>0.424</td>
<td>44.10%</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Comparing (worst-case) INTERP with QUERY EXPANSION (MRR for each task)

The results for the various tasks are shown in Table 3. We see that for all the tasks, there is an at least 44% improvement in the MRR.

6. RELATED WORK

As mentioned in the introduction, specific precision-oriented search tasks such as HomePage Finding and Known-Item Search [26, 31, 17, 19, 15] have been addressed in the literature. However, the work presented in this paper is a generic query interpretation infrastructure that is applicable to any precision-oriented retrieval system and not restricted to specific tasks or document collections.

**XML and IR.** The field of XML information retrieval (XML-IR) has been an active area of research within the database and information retrieval communities. Work in this area ranges from languages and techniques for integration of IR features into XML queries to algorithms for ranking keyword queries over XML document collections [2, 3, 11, 27, 30, 12]. In Section 1.3, we showed that in the context of precision-oriented search, our interpretations-based approach has distinct advantages over the direct application of XML-IR techniques. However, since interpretations involve both structural and text matching predicates, XML-IR techniques can be exploited in our system to realize efficient data indexes.

The SphereSearch engine [12] applies XML-IR techniques in the context of heterogeneous Web data and incorporates information extraction techniques to identify concepts from text. However, the system exposes a complex query language which is in direct contrast to our approach of exploiting annotations “under the covers.”

**Extended IR models.** Several extended IR models have been proposed to improve result quality by incorporating document structure and external ontologies into the retrieval process [18, 24, 16]. Some of these approaches require the user to explicitly use a complex query language in order to fully exploit the external ontologies. Other approaches exploit these ontologies implicitly through query expansion or as extensions to the retrieval model. However, to the best of our knowledge, none of these approaches are specifically designed for precision-oriented search tasks.

**Information discovery.** Several research projects in the area of information discovery, such as DBXplorer [1], DISCOVER [14], BANKS [6], and [13], have addressed the problem of supporting keyword queries over structured data. Given a keyword query, these systems join tuples from multiple relations in the database to identify tuple trees with all the query keywords. While there is a parallel between this operation and the task of translating keyword queries into interpretations over annotation store, the end–goals are quite different. These differences are reflected both in the choice of result elements (tuple sets versus documents) and in the class of queries (arbitrary SQL queries versus the restricted class described in Section 3).

**Question–answering systems.** Question answering systems are an active area of research in the information retrieval and AI communities [23, 9, 29]. Several QA systems use a process known as predictive annotation in which annotators are employed to extract concepts (e.g., the Piquante [23] system recognizes about 95 types). The runtime analysis performed by QA systems is similar in spirit to interpretations. However, an important difference lies in the fact that the first step in QA system is oriented towards recall, i.e., geared to return all documents that may potentially contain the answer. In direct contrast, our interpretation approach targets precision.

**Relevance Feedback.** In the two-stage retrieval model, allowing the user to select the interpretation of choice is similar in spirit to relevance feedback [4]. However, unlike (explicit) relevance feedback, where the user selects a set of relevant documents from the results, in our approach the user selects the interpretation of interest in the first stage.

7. SUMMARY

In this paper, we addressed the important class of precision-oriented search tasks. We presented an architecture for precision-oriented retrieval and demonstrated the value of explicitly modeling the search task in terms of structured queries known as interpretations. We formally defined the concept of justified interpretations of a keyword query and described a sequence of algorithms for enumerating, pruning, and ranking such interpretations. Our experimental results have confirmed the validity of our algorithms as well as the overall framework.

In terms of future work, there are several interesting challenges that remain to be addressed with respect to efficient pruning and ranking strategies. Recall that the intent distribution \( \hat{p}(I|k) \) was a key factor in the ranking algorithm presented in Section 4.3 and that we used the “semantic value” heuristic to avoid computing this distribution. However, by incorporating statistics from the annotation store (e.g., how many times was ‘tom’ annotated as a person versus as an organization) or from other sources such as query logs and user profiles, it may be possible to develop reasonable approximations of this distribution. An investigation of such techniques is an interesting avenue for future work.

In this paper, we chose the set of matching interpretations to represent user intent. However, it is not obvious when more expressive interpretations (e.g., covering) will be useful. An understanding of the queries for which more expressive interpretations must be generated and efficient pruning and ranking algorithms for these scenarios are interesting research questions.

8. REFERENCES


**APPENDIX**

**A. ANNOTATORS**

The schema shown in Figure 2 was explicitly hand constructed for email search. Corresponding to each type in the schema is an annotator – an information extraction program capable of identifying instances of that type. Our schema for email consists of 20 types of which 17 are populated using base annotators and three – AuthorPhone, PersonPhone and Signature – using relationship annotators (described below).

In our implementation, we used the publicly available UIMA framework [32] to define an analysis workflow consisting of these analysis engines. Documents are fed in at one end of the workflow and the resulting annotations are persisted in an annotation store as shown in Figure 4.

**Base Annotators** are annotators that run over the raw document text. The actual algorithms depend on the task being performed. Consider for example the task of recognizing persons and organizations. These annotators use grammar-like rules in conjunction with configurable domain dictionaries to accomplish their task. For our experiments, we have used the particular named-entity recognizer described in [25]. However, several other implementations exist and are described in [20]. We also employ pattern-matching annotators to recognize telephone numbers, street addresses and flight details. For example, the FlightInfo annotator identifies instances in documents where flight details are present. Another pattern matching annotator is the Directions annotator, which identifies whether a document contains directions to some place.

**Relationship Annotators**. As seen in Figure 2, AuthorPhone and PersonPhone relate different concepts. Such relationships are extracted by annotators that operate on the raw text and other previously defined annotations. For instance, consider the AuthorPhone relationship which indicates that a phone number mentioned in the body of an email belongs to the author of that email. This relationship is extracted by first recognizing phone numbers (base-annotator) and then relating this phone number to the author by matching one of several regular expressions. One example pattern is: I can be (reached|contacted) at <PhoneNumber>.