Semantic Ranking and Result Visualization for Life Sciences Publications

Julia Stoyanovich [1], William Mee [2], Kenneth A. Ross [3]

Columbia University
New York, NY USA

[1]jds1@cs.columbia.edu
[2]wjm2107@columbia.edu
[3]kar@cs.columbia.edu

Abstract— **An ever-increasing amount of data and semantic knowledge in the domain of life sciences is bringing about new data management challenges. In this paper we focus on adding the semantic dimension to literature search, a central task in scientific research. We focus our attention on PubMed, the most significant bibliographic source in life sciences, and explore ways to use high-quality semantic annotations from the MeSH vocabulary to rank search results. We start by developing several families of ranking functions that relate a search query to a document's annotations. We then propose an efficient adaptive ranking mechanism for each of the families. We also describe a two-dimensional Skyline-based visualization that can be used in conjunction with the ranking to further improve the user's interaction with the system, and demonstrate how such Skylines can be computed adaptively and efficiently. Finally, we evaluate the effectiveness of our ranking with a user study.**

I. Introduction

Many scientific domains, most notably the domain of life sciences, are experiencing unprecedented growth. The recent complete sequencing of the Human Genome, and the tremendous advances in experimental technology are rapidly bringing about new scientific knowledge. The ever-increasing amount of data and semantic knowledge in life sciences requires the development of new semantically rich data management techniques that facilitate scientific research and collaboration.

Literature search is a central task in scientific research. PubMed (`www.pubmed.gov`) is the most significant bibliographic source in the domain of life sciences, with over 18 million articles at the time of this writing. Indexed articles go back to 1865, and the number of articles grows daily. PubMed articles are annotated by a staff of indexers with terms from the Medical Subject Headings (MeSH) controlled vocabulary. MeSH organizes term descriptors into a hierarchical structure, allowing searching at various levels of specificity. The 2008 version of MeSH contains 24,767 term descriptors that refer to general concepts such as *Anatomy* and *Mental Disorders*, as well as to specific concepts such as *Antiphospholipid Syndrome* and *Cholesterol*. Indexers are instructed to assign the most specific terms possible to a PubMed article.

MeSH terms are classified into an *is-a polyhierarchy*: the hierarchy defines is-a relationships among terms, and each term has one or more parent terms [9]. Figure 1 presents a portion of MeSH that describes autoimmune diseases and connective tissue diseases. The hierarchy is represented by a tree of *nodes*, with one or several nodes mapping to a single *term label*. For example, the term *Rheumatic Diseases* is represented by the node $C17.300.775.099$.

Interestingly, the MeSH hierarchy is *scoped*: two tree nodes that map to the same term label may not always induce isomorphic subtrees. The term *Rheumatoid Arthritis (RA)* maps to two nodes in Figure 1, and induces subtrees of different sizes. Node $C20.111.199$ represents the autoimmune aspect of *RA* and induces a subtree of size 5, while $C17.300.775.099$ refers to *RA* as a rheumatic disease, and induces a subtree of size 7. (Subtree size is noted next to the name of the node.) Scoping is an important technique for modeling complex polyhierarchies. Placing a concept in several parts of the hierarchy models different aspects of the concept, while accommodating different context in different parts of the hierarchy adds to the expressive power and reduces redundancy.

PubMed can be searched with Entrez, the Life Sciences Search Engine. Entrez implements sophisticated query processing, allowing the user to specify conjunctive or disjunctive Boolean semantics for the search query, and to relate the search terms to one or several parts of the document: title, MeSH annotations, text of the document, etc. In order to improve recall, Entrez automatically expands query terms that are related to MeSH annotations with synonymous or near-synonymous terms. For example, the simple query *mosquito* will be transformed by Entrez to *"culicidae"[MeSH Terms] OR "culicidae"[All Fields] OR "mosquito"[All Fields]*. Entrez also expands the query with descendants of any MeSH terms. For example, the query *"blood cells"[MeSH Terms]* will match articles that are annotated with *"blood cells"* or with *"erythrocytes"*, *"leukocytes"*, *"hemocytes"* etc.

The need to improve recall differentiates bibliographic search from general web search. In web search it is often assumed that many documents equivalently satisfy the user's information need, and so high recall is less important than high precision among the top-ranked documents. Conversely, in bibliographic search the assumption (or at least the hope) is that every scientific article contributes something novel to the state of the art, and so no two documents are interchangeable when it comes to satisfying the user's information need. In this scenario the Boolean retrieval model, such as that used by Entrez, guarantees perfect recall and is the right choice.

However, there is an important common characteristic of bibliographic and general web search: many queries return hundreds, or even thousands, of relevant results. Query expan-

978-1-4244-5446-4/10/$26.00 © 2010 IEEE

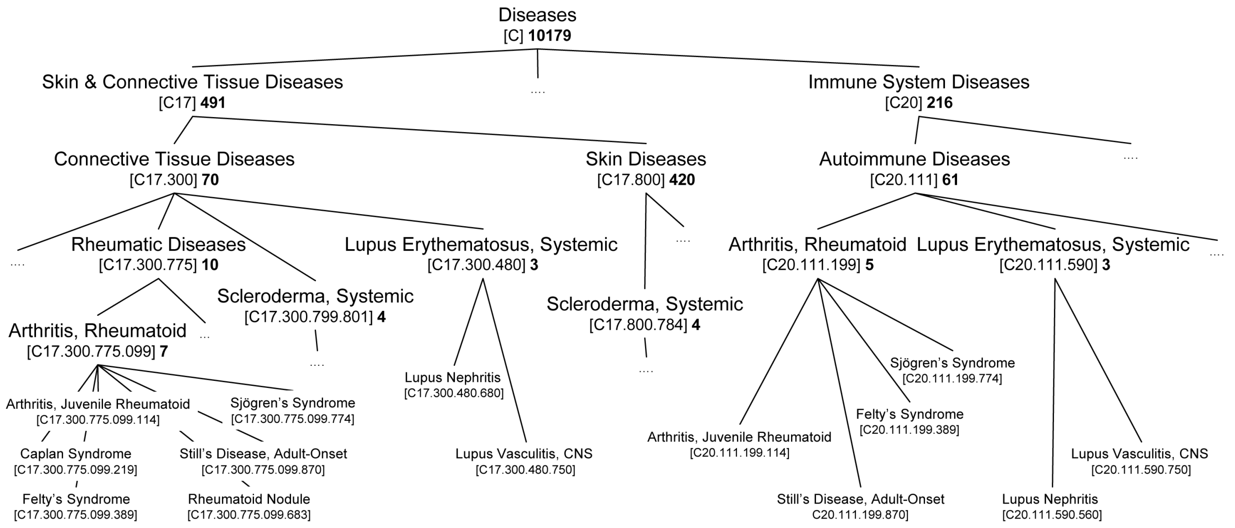


Fig. 1. Portion of the MeSH polyhierarchy.

sion techniques that maximize recall exacerbate this problem by producing yet more results. For example, the fairly specific query *Antiphospholipid Antibodies AND Thrombosis*, which looks for information about a particular clinical manifestation of Antiphospholipid syndrome, returned 2455 matches using the default query translation in January 2009. A more general query that looks for articles about connective tissue diseases that are also autoimmune returns close to 120,000 results.

Because so many results are returned per query, the system needs to help the user explore the result set. Entrez currently allows the results to be sorted by several metadata fields: publication date, first author, last author, journal, and title. This may help the user look up an article with which he is already familiar (i.e., knows some of the associated metadata), but does not support true information discovery.

A useful and well-known way to order results in web information retrieval is by query relevance. Retrieval models such as the Vector Space Model [1] have the query relevance metric built in, while the Boolean retrieval model does not. In this paper we propose to measure the relevance of a document to the query with respect to the MeSH vocabulary. We illustrate some semantic considerations and challenges with an example.

Example 1.1: Consider the Entrez query *"Connective Tissue Diseases" [MeSH Terms] AND "Autoimmune Diseases" [MeSH Terms]*, evaluated against PubMed. Figure 1 represents these query terms in the context of the MeSH hierarchy. The query will match all documents that are annotated with at least one term from each of the induced subtrees of the query terms.

One of the results, a document with $pmid = 17825677$, is a review article that discusses the impact of autoimmune disorders on adverse pregnancy outcome. It is annotated with the query terms *"Autoimmune Diseases"* and *"Connective Tissue Diseases"*, and also with several terms from the induced subtrees of the query terms: *"Arthritis, Rheumatoid"*, *"Lupus Erythematosus, Systemic"*, *"Scleroderma, Systemic"*, and *"Sjögren's Syndrome"*. The article is also annotated with general terms that are not related to the query terms via the hierarchy: *"Pregnancy"*, *"Pregnancy Complications"*, *"Female"*, and *"Humans"*.

Another result, an article with $pmid = 19107995$, describes neuroimaging advances in the measurement of brain injury in Systemic Lupus. This article matches the query because it is annotated with *"Lupus Erythematosus, Systemic"*, which is both a connective tissue disease and an autoimmune disease. The article is also annotated with broader terms *"Brain"*, *"Brain Injuries"*, *"Diagnostic Imaging"*, and *"Humans"*.

Based on this example, we observe that, while both articles are valid matches for the query, they certainly do not carry equal query relevance. The first article covers the fairly general query terms, as well as several specific disorders classified below the query terms in MeSH. In contrast, the second article answers a limited portion of the query, since it focuses on only one particular disorder. In this work we propose several ways to measure semantic relevance of a document to a query, and demonstrate how our semantic relevance can be computed efficiently on the scale of PubMed and MeSH.

An important dimension in data exploration, particularly in a high-paced scientific field, is time. An article that contributes to the state of the art at the time of publication may quickly become obsolete as new results are published. Semantic relevance measures of this paper can be used to retrieve ranked lists of results, or they can be combined with data visualization techniques that give an at-a-glance overview of thousands of results. We develop a two-dimensional skyline visualization that plots relevance against publication date, and show how such skylines can be computed efficiently on the large scale.

Ranking that takes into account hierarchical structure of the domain has been considered in the literature [6], [12],

[13]. Such ranking typically relates two terms via a common ancestor; see Section VI for a discussion of these methods. When terms appear in the hierarchy in multiple places, with subtly different meanings, it is unclear how such distance-based measures should be generalized. Instead, in this paper we develop new families of ranking measures that are aimed specifically at ranking with scoped polyhierarchies like MeSH, where terms may occur in multiple (partially replicated) parts of the hierarchy. We argue that the semantics of a term is best captured by its set of descendants across the whole hierarchy, and develop measures of relatedness that depend on the nature of the overlap between these sets of descendants.

Computing similarity based on sets of descendants is algorithmically more complex than simpler graph distance measures. We pay particular attention to efficiency, and provide an extensive experimental evaluation of our methods with the complete PubMed dataset and the full MeSH polyhierarchy, demonstrating that interactive response times are achievable.

The rest of this paper is organized as follows. We formalize semantics of query relevance for scoped polyhierarchies in Section II. We present the data structures and algorithms that implement the query relevance measures on the large scale in Section III. Section IV describes an evaluation of efficiency, and Section V presents a user study. We present related work in Section VI, and conclude in Section VII.

II. Semantics of Query Relevance

We now formalize the data model, and define the semantics of several similarity measures, using the polyhierarchy in Figure 2 for demonstration. Term labels are denoted by letters $A, B, C, \ldots$, and nodes are denoted by numerical ids $1, 2, 3, \ldots$. Term $\top$ represents the root of the hierarchy.

A. Motivation

We wish to assign a score to documents whose MeSH terms overlap with the query terms. Our notion of “overlap” includes cases where a document term represents a sub-concept of a query term. If a query is $\{A, B\}$ in Figure 2, and the document contains MeSH terms C and D, then both C and D contribute to the overlap because they are sub-concepts of A and B.

Our first similarity measure, which we formalize in Section II-C, simply counts the number of terms in common between the descendants of the MeSH terms in the query and those in the document. According to this measure, concepts such as C that appear in multiple parts of the hierarchy count once. However, we might want to count C more than once because it contributes to the matching of both query terms.

The alternative of simply counting every occurrence of a term label can be naive. Suppose that the query is $\{C\}$ and that the document mentions term G but not C or H. One could argue that double-counting G is inappropriate, since the only reason we have two G instances is because C appears in multiple parts of the hierarchy. *Within the context of* C, G only appears once. This observation motivates us to only double-count when the ancestor concept in the query is different. We develop a similarity measure that models this intuition in Section II-D.

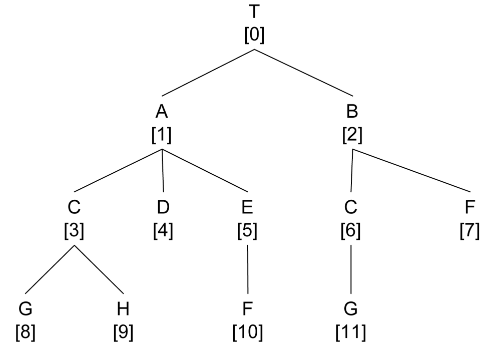


Fig. 2. A sample scoped polyhierarchy.

The measures mentioned so far are sensitive to the size of the hierarchy. Because A has more descendants than B, an intermediate-level match in the A subtree may give a much larger score than a high-level match in the B subtree. The effect of this bias would be that highly differentiated concepts would be consistently given more weight than less differentiated concepts. To overcome this bias, we consider a scoring measure in Section II-E that weights matches in such a way that each query term contributes equally to the score.

B. Terminology

Definition 2.1: A *scoped polyhierarchy* is a tuple $\mathcal{H} = \{\mathcal{T}, \mathcal{N}, ISA, L\}$, where $\mathcal{T}$ is a set of term labels, $\mathcal{N}$ is a set of nodes, $ISA : \mathcal{N} \rightarrow \mathcal{N}$ is a many-to-one relation that encodes the generalization hierarchy of nodes, and $L : \mathcal{N} \rightarrow \mathcal{T}$ associates a term with each node. When $ISA(n, n')$ holds, we say n' is a *parent* of n, and n is a *child* of n'. Every node except the root has exactly one parent node. Node n' is an *ancestor* of n if (n, n') is in the reflexive transitive closure of ISA. (Thus a node is its own ancestor and its own descendant.)

For a term $t \in \mathcal{T}$, we denote by $N(t)$ the set of nodes n with label t (i.e., having $L(n) = t$). For a set of terms $T \subseteq \mathcal{T}$, we denote by $N(T)$ the set of nodes in $\bigcup_{t \in T} N(t)$. Likewise, for a set of nodes $M \subseteq \mathcal{N}$, we denote by $L(M)$ the set of labels of nodes in M.

Definition 2.2: The *node-scope* of a term $t \in \mathcal{T}$, denoted by $N^*(t)$, is the set of nodes that have an ancestor with the label t: $N^*(t) = \{n | \exists n', t = L(n') \wedge ancestor(n', n)\}$.

The *node-scope* of a set of terms $T \subseteq \mathcal{T}$, denoted by $N^*(T)$, is the set of nodes that have an ancestor with the label in T: $N^*(T) = \bigcup_{t \in T} N^*(t)$.

In Figure 2, the node-scope of the term C is $N^*(C) = \{3, 8, 9, 6, 11\}$, the same as the node scope of a set $\{C, G, H\}$.

Definition 2.3: The *term-scope* of a term $t \in \mathcal{T}$, denoted by $L^*(t)$, is the set of term labels that appear among the nodes in $N^*(t)$: $L^*(t) = \bigcup_{n \in N^*(t)} L(n)$.

We define the *term-scope* of a set of terms $T \subseteq \mathcal{T}$ analogously to the node-scope, and denote it by $L^*(T) = \bigcup_{t \in T} L^*(t)$.

The term-scope of the term C in Figure 2 is $L^*(C) = \{C, G, H\}$, while $L^*(\{B, C\}) = \{B, C, G, H, F\}$.

We use *node-scope* and *term-scope* to compare two sets of terms D and Q, where D is the set of terms that annotate a PubMed document, and Q is the set of query terms.

C. Set-Based Similarity

Our first measure, *term similarity*, treats the sets D and Q symmetrically, and quantifies how closely the two sets are related by considering the intersection of their *term-scopes*:

$$TermSim(D, Q) = |L^*(D) \cap L^*(Q)| \tag{1}$$

Term similarity may be used on its own, or it may be normalized by another quantity, changing the semantics of the score. For example, normalizing term similarity by the size of the *term-scope of the query* expresses the extent to which the query is answered by the document. We refer to this quantity as *term coverage*. Dividing the term similarity by the *term-scope of the document* expresses how specific the document is to the query. We refer to this quantity as *term specificity*. Finally, we may divide term coverage by the size of the union of the two term scopes, deriving *Jaccard similarity*.

D. Conditional Similarity

Set-based similarity treats the query and the document symmetrically, although it may prioritize one set over the other in the final step, as is done in *term coverage* and *term specificity*. *Conditional similarity* prioritizes the query over the document from the start, by placing the term-scope of the document within the context of the term-scope of the query.

As we argued in Section II-A, simply counting the paths between two terms can be naive, as we may be double-counting due to structural redundancy in the hierarchy. We thus define *conditional term-scope* by using ancestor-descendant pairs of terms, not full term paths. In the following definition, q is a query term and d is a document term.

Definition 2.4: Let d and q be terms, and let $P_{d,q}$ be the set of node pairs (n_d, n_q) satisfying the following conditions:

- $n_d \in N^*(d)$, i.e., n_d has an ancestor with label d;
- $n_q \in N^*(q)$, i.e., n_q has an ancestor with label q;
- n_q is an ancestor of n_d.

Conditional term-scope of d given q, denoted by $L^*(d|q)$, is the set of label pairs $(L(n_1), L(n_2))$, where $(n_1, n_2) \in P_{d,q}$.

Conditional term-scope of a set D given a set Q, denoted $L^*(D|Q)$, is the union of conditional term-scopes of all $d \in D$ given all $q \in Q$: $L^*(D|Q) = \bigcup_{d \in D, q \in Q} L^*(d|q)$.

For example, $L^*(G|C) = \{(C, G), (G, G)\}$, while $L^*(G|\{A, B\}) = \{(A, G), (B, G), (C, G), (G, G)\}$.

Note that $L^*(q|q)$ enumerates all pairs of terms (s, t), where $s, t \in L^*(q)$ such that there is a term-path from a node labeled with t to a node labeled with s. So, $L(C|C) = \{(C, G), (C, H), (C, C), (G, G), (H, H)\}$.

We define *conditional similarity* as:

$$CondSim(D, Q) = |L^*(D|Q)| \tag{2}$$

E. Balanced Similarity

Balanced similarity is a refinement of *conditional similarity* that balances the contributions of query terms to the score.

$$BalancedSim(D, Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{CondSim(D, q)}{CondSim(q, q)} \tag{3}$$

The relative contribution of each query term q to the score is normalized by the number of terms in the query, $|Q|$. For each term q, we compute the conditional similarity between the document D and the term q (as per Equation 2), and normalize this value by the maximum possible conditional similarity that any document may achieve for q, which is $CondSim(q, q)$.

III. Efficient Computation of Query Relevance

In this section we describe the data structures and algorithms that support computing similarity measures of Section II at the scale of PubMed and MeSH. We do all processing in main memory to achieve interactive response time, and must control the size of our data structures so as to not exceed reasonable RAM size. Our data structures are at most linear in the size of PubMed, and at most quadratic in the size of MeSH.

We maintain annotations and publication date of PubMed articles in a hash table $Articles$, indexed by $pmid$, a unique document identifier in PubMed. The version of PubMed to which we were given access by NCBI consists of about 17 million articles, published up to September 2007, and we are able to store publication date and annotations of all these articles in RAM. There are between 1 and 96 annotations per article, 9.7 on average.

In this work we focus on queries that are conjunctions or disjunctions of MeSH terms, and rely on the query processing provided by Entrez to retrieve query matches. Note that, while the query semantics is Boolean, it incorporates ontology expansion, blurring the line between strictly Boolean and set-oriented processing. So, a document D will match a query $Q = \{q_1, q_2\}$ if D is annotated with at least one term in the term-scope of each of q_1 and q_2. Article $pmid = 19107995$ in Example 1.1 matches the query *"Connective Tissue Diseases" [MeSH Terms] AND "Autoimmune Diseases" [MeSH Terms]* because it is annotated with *"Lupus Erythematosus, Systemic"*, satisfying both query terms. We do not discriminate between AND and OR queries for the purposes of ranking.

A. Exact Computation

We maintain the following data structures that allow us to compute values for the relevance metrics in Section II. There are 24,767 terms and 48,442 nodes in MeSH 2008, the version of MeSH that we use in this work. For each term $t \in \mathcal{T}$, we precompute and maintain the following information in one or several hash tables, indexed on the term label.

- $N(t)$, the set of nodes that have t as its label.
- $L^*(t)$, the set of term labels in the term-scope of t.
- $N^*(t)$, the set of nodes in the node-scope of t.
- $|L^*(t|t)|$, the size of conditional term scope of t.

For each node $n \in \mathcal{N}$, we maintain its term label $L(n)$, and the path from the top of the hierarchy to n.

Algorithm 1 Procedure **TermSim**

```
Require: $Q = \{q_1 \dots q_n\}$, $R = \{pmid_1 \dots pmid_m\}$
 1: Compute $L^*(Q) = \bigcup_i L^*(q_i)$
 2: for $pmid \in R$ do
 3:     Retrieve $D = \{d_1 \dots d_m\}$ from $Articles$
 4:     Compute $L^*(D) = \bigcup_i L^*(d_i)$
 5:     $termSim(D, Q) = |L^*(D) \cap L^*(Q)|$
 6: end for
```

Algorithm 1 describes how *term similarity* (Eq. 1) is computed for a query Q and a set of documents R. To compute the term-scope of a term t (lines 1 and 4), we retrieve $L^*(t)$ with a hash table lookup. Each lookup returns a set of terms, and the size of each such set is linear in the size of the hierarchy. In practice, for terms that denote general concepts, $L^*(t)$ may contain hundreds, or even thousands of term labels, while for terms that denote very specific concepts, $L^*(t)$ will contain only a handful of labels. Next, we take a union of the term-scopes of individual terms, which requires time linear in the size of the input data structures in our implementation. This computation happens once per query, and once for every document. Finally, having computed the term-scope of the document, we determine the intersection $L^*(D) \cap L^*(Q)$ (line 5). This operation takes time linear in the size of the data structures, and is executed once per document.

Algorithm 2 computes *conditional similarity* (Eq. 2) for a query Q and a document D. Term-scope and node-scope of Q are computed on lines 1 and 2. Then, for each document, we compute D_Q, the set of its terms that are in the term-scope of the query, and retrieve the node-scope of D_Q (lines 5 and 6). We then find all pairs of nodes $n' \in N^*(Q)$ and $n \in N^*(D_Q)$ such that there is a path from n' to n. Each document is processed in time proportional to $|N^*(Q)| * |N^*(D_Q)|$, which can be high for queries and documents with large node-scopes.

Algorithm 2 Procedure **CondSim**

```
Require: $Q = \{q_1 \dots q_n\}$, $R = \{pmid_1 \dots pmid_m\}$
 1: Compute $L^*(Q) = \bigcup_i L^*(q_i)$
 2: Compute $N^*(Q) = \bigcup_i N^*(q_i)$
 3: for $pmid \in R$ do
 4:     Retrieve $D = \{d_1 \dots d_m\}$ from $Articles$
 5:     Compute $D_Q = D \cap L^*(Q)$
 6:     Compute $N^*(D_Q)$
 7:     $\mathcal{S} = \emptyset$
 8:     for $n' \in N^*(Q)$ do
 9:         for $n \in N^*(D_Q)$ do
10:             if $ancestor(n', n)$ then
11:                 $\mathcal{S} = \mathcal{S} \cup (L(n'), L(n))$
12:             end if
13:         end for
14:     end for
15:     $condSim(D, Q) = |\mathcal{S}|$
16: end for
```

Algorithm 3 computes *balanced similarity* (Eq. 3) by considering each query term q separately, and invoking *CondSim* for each document. Computing conditional similarity one query term at a time has lower processing cost than the corresponding computation for the query as a whole, as is done in *CondSim*, as we will see during our experimental evaluation.

Algorithm 3 Procedure **BalancedSim**

```
Require: $Q = \{q_1 \dots q_n\}$, $R = \{pmid_1 \dots pmid_m\}$
 1: Compute $weight_i = |Q| * L^*(q_i|q_i)$ for each $q_i \in Q$
 2: for $pmid \in R$ do
 3:     $score = 0$
 4:     for $q_i \in Q$ do
 5:         $score = score + weight_i * CondSim(q_i, pmid)$
 6:     end for
 7:     $balancedSim(D, Q) = score$
 8: end for
```

B. Computation with Score Upper-Bounds

In the previous section we saw that evaluating similarity of a set of documents with respect to a query can be expensive, particularly for queries and documents that are annotated with general MeSH terms. We now show how score upper-bounds can be computed more efficiently than exact scores.

Score upper-bounds can be used to limit the number of exact score computations in ranked retrieval, where only k best entries are to be retrieved from among N documents, and $k \ll N$. If score upper-bounds are cheaper to compute than actual scores, then we can compute score upper-bounds for all documents, order documents in decreasing order of score upper-bounds, and compute exact score values as needed, until the k best documents have been retrieved. Processing, and thus exact score computation, can stop when the score upper-bound of the document being considered is lower than the actual score of the current k^{th} best document. In addition to computing score upper-bounds for all documents, and evaluating exact scores for M documents, where $k \leq M \leq N$, the algorithm must perform a certain number of sorts, to determine the current k^{th} score at every round.

Consider again the computation of *term similarity* in Algorithm 1, which computes the value of the expression in Equation 1. We can transform this equation using distributivity of set intersection over set union, and observe that a natural upper-bound holds over the value of *term similarity*:

$$TermSim(D, Q) = |(\bigcup_d L^*(d)) \cap (\bigcup_q L^*(q))| =$$
$$|\bigcup_{d,q} L^*(d) \cap L^*(q)| \leq \sum_{d,q} |L^*(d) \cap L^*(q)|$$

The value of $TermSim(D, Q)$ cannot be higher than the sum of the sizes of pair-wise intersections of term-scopes of terms from D with terms from Q. To enable fast computation of this upper bound, we precompute $|L^*(s) \cap L^*(t)|$ for all pairs of terms s and t. The number of entries in this data structure, which we call *PairwiseTermSim*, is quadratic in the size of MeSH. In practice, we only need to record an entry for the terms s and t if $L^*(s) \cap L^*(t) \neq \emptyset$. There are over 613 million possible pairs of MeSH terms, but only 158,583 pairs have a non-empty intersection of their term-scopes.

For a query of size $|Q|$ and a document of size $|D|$, we need to look up $|Q| * |D|$ entries in *PairwiseTermSim*, and compute a sum of the retrieved values. The difference between the size of a set of terms, and the size of the term-scope of that set can be quite dramatic, and so computing upper-bounds is often much cheaper than computing actual scores. We will demonstrate this experimentally in Section IV.

Let us now consider how score upper-bounds can be computed for *conditional similarity* (Eq. 2), which counts the number of pairs of terms $q \in L^*(Q)$ and $d \in L^*(D)$ such that there is a node-path from $q \rightarrow d$. This quantity is bounded by the sum of sizes of $L^*(d|q)$ for all pairs of terms d and q.

$$CondSim(D, Q) = |\bigcup_{d,q} L^*(d|q)| \leq \sum_{d,q} |L^*(d|q)|$$

To facilitate the computation of this upper-bound, we store the value of $L^*(s|t)$ for all pairs of terms s and t with intersecting term-scopes. We call this data structure $PairwiseCondSim$. This data structure has the same number of entries as *PairwiseTermSim*.

Finally, for *balanced similarity*, we observe that:

$$BalSim(D, Q) = \frac{1}{|Q|} \sum_{q} \frac{L^*(D|q)|}{L^*(q|q)} = \frac{1}{|Q|} \sum_{q,d} \frac{L^*(d|q)}{L^*(q|q)}$$

We re-use the *PairwiseCondSim* data structure for the computation of score-upper bounds for *balanced similarity*. We evaluate the performance improvements achieved by using score upper-bounds for ranked retrieval in Section IV.

C. Adaptive Skyline Computation with Upper-Bounds

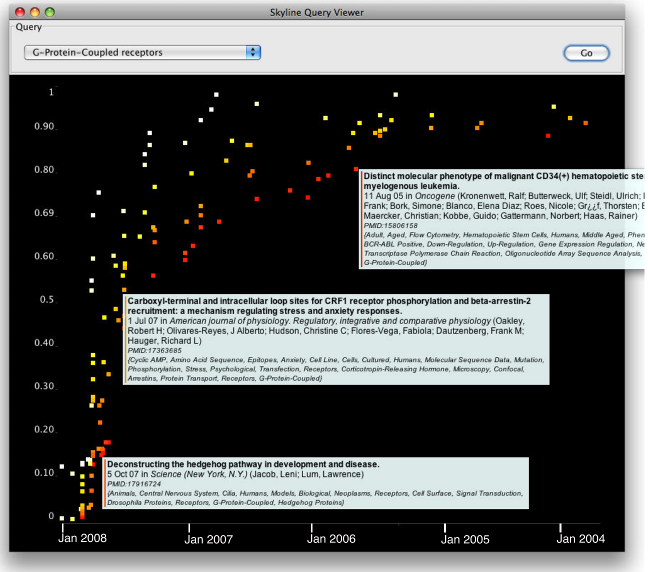
As we argued in the Introduction, it is sometimes important to present more than a handful of query results. We propose to use a two-dimensional *skyline* visualization [3] that is based on the concept of dominance. A point in multi-dimensional space is said to belong to the skyline if it is not *dominated* by any other point, i.e., if no other point is as good or better in all dimensions, and strictly better in at least one dimension.

A skyline *contour* is defined inductively as follows:

- A point belongs to the first skyline contour if and only if it belongs to the skyline of the whole data set.
- A point belongs to the k^{th} contour if and only if it belongs to the skyline of the data set obtained by removing points from the first through $k - 1^{st}$ contours.

Skyline contours are useful for highlighting points that are close to the skyline, and that might be of interest to the user.

Publication date is a natural attribute in which to consider bibliography matches, and we use it as the x-axis of our visualization. The y-axis corresponds to one of the similarity measures described in Section II. Figure 3 shows a skyline of results for the query *G-Protein-Coupled receptors*, for *term specificity* with 5 skyline contours. Points of highest quality are close to the origin on the x-axis and away from the origin on the y-axis. Points on the first contour are marked in white, points on the second contour are beige, and points on the last contour are red. When points are selected using the mouse, a window showing the full citation is displayed.



Distinct molecular phenotype of malignant CD34(+) hematopoietic ste myelogenous leukemia.
11 Aug 05 in *Oncogene* (Kronenwett, Ralf; Butterweck, Ulf; Steidl, Ulrich; I Frank; Bork, Simone; Blanco, Elena Diaz; Roes, Nicole; Gr¿¿f, Thorsten; E Maercker, Christian; Kobbe, Guido; Gattermann, Norbert; Haas, Rainer)
PMID:15806158
{Adult, Aged, Flow Cytometry, Hematopoietic Stem Cells, Humans, Middle Aged, Phen BCR-ABL Positive, Down-Regulation, Up-Regulation, Gene Expression Regulation, Ne Transcriptase Polymerase Chain Reaction, Oligonucleotide Array Sequence Analysis, G-Protein-Coupled}

Carboxyl-terminal and intracellular loop sites for CRF1 receptor phosphorylation and beta-arrestin-2 recruitment: a mechanism regulating stress and anxiety responses.
1 Jul 07 in *American journal of physiology. Regulatory, integrative and comparative physiology* (Oakley, Robert H; Olivares-Reyes, J Alberto; Hudson, Christine C; Flores-Vega, Fabiola; Dautzenberg, Frank M; Hauger, Richard L)
PMID:17363685
{Cyclic AMP, Amino Acid Sequence, Epitopes, Anxiety, Cell Line, Cells, Cultured, Humans, Molecular Sequence Data, Mutation, Phosphorylation, Stress, Psychological, Transfection, Receptors, Corticotropin-Releasing Hormone, Microscopy, Confocal, Arrestins, Protein Transport, Receptors, G-Protein-Coupled}

Deconstructing the hedgehog pathway in development and disease.
5 Oct 07 in *Science (New York, N.Y.)* (Jacob, Leni; Lum, Lawrence)
PMID:17916724
{Animals, Central Nervous System, Cilia, Humans, Models, Biological, Neoplasms, Receptors, Cell Surface, Signal Transduction, Drosophila Proteins, Receptors, G-Protein-Coupled, Hedgehog Proteins}

Fig. 3. Two-dimensional skyline representation of results for the query *G-Protein-Coupled receptors*. Please view on a color display or on a color printer.

Our prototype implementation is running outside of the NCBI infrastructure, and we are using the Entrez query API, `eUtils`, to evaluate queries, and receive back ids of PubMed articles that match the query. The `eUtils` API can be asked to return query results in order of publication date. NCBI requests that large result sets be retrieved in batches, so as not to overload their system. In the remainder of this section we describe a progressive algorithm that computes a two-dimensional skyline of results using score upper-bounds.

We implemented a divide-and-conquer algorithm based on techniques described by Bentley [2]. Our algorithm processes results one batch at a time, with batches arriving in order of article publication date, from more to less recent. Articles within each batch are also sorted on publication date, and we use this sort order as basis for the divide-and-conquer.

The algorithm receives as input a sorted list of documents, a query Q, an integer k that denotes the number of skyline contours to be computed, a similarity measure Sim, and $SkylineSoFar$: a list of documents, sorted on publication date, that were identified as belonging to the skyline when processing previous batches, along with the contour number. Note that a result that was assigned to the skyline during a previous batch will remain on the skyline, with the same contour number, for the remainder of the processing. This is because documents are processed in sorted order on publication date.

The divide-and-conquer algorithm processes the batch by recursively dividing the points along the median on the x-axis. When all points within an x-interval share the same x value, the algorithm sorts the points on the y coordinate,

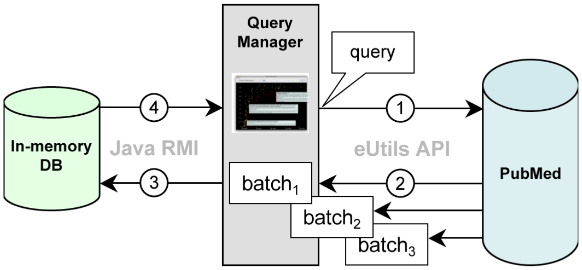


Fig. 4. System architecture.

identifies contour points as the k best points in the interval, and assigns to each of the top-k points a contour number. Let us refer to this sub-routine as *AssignLinearDominance*. Contour number assignments are then merged across intervals, from left to right, and contour numbers of points on the right are adjusted. The $SkylineSoFar$ data structure is supplied to the left-most interval when a batch is processed.

The algorithm assumes that the values of the x and the y coordinates are readily available for each document. However, as we discussed in Section III, the similarity score of the document may be expensive to compute, while the score upper-bound may be computed more efficiently. We therefore modify the *AssignLinearDominance* subroutine to use score upper-bounds as in Section III-B. Exact scores are still computed, but the number of these computations is reduced. Using score upper-bounds allows us to compute the two-dimensional skyline more efficiently, as we demonstrate next.

IV. Experimental Evaluation

A. Experimental Platform

We evaluated the performance of our methods on a Java prototype. Figure 4 describes the system architecture and the data flow. Processing is coordinated by the *Query Manager* that receives a query from the user and communicates with PubMed via the `eUtils` API (arrow 1). The Entrez search engine evaluates the query against the live PubMed database, and returns $pmid$s of results in batches, sorted in decreasing order of publication date (arrow 2). *Query Manager* communicates with the *In-Memory DB*, which stores MeSH annotations of all articles up to September 2007, and implements the data structures and algorithms of Section III. *In-Memory DB* and *Query Manager* communicate via Java RMI (arrows 3, 4). *In-Memory DB* runs on a 32-bit machine with a dualcore 2.4GHz Intel CPU and 4GB of RAM, with RedHat EL 5.1. Given a query and a list of $pmid$s, *In-Memory DB* can compute score upper-bounds or actual scores for each document, or it can compute the set of skyline contours. Results are read by *Query Manager* (arrow 4), which can optionally pass them to the visualization component.

The total execution time of most queries is interactive in our implementation, and depends in part on the response time of Entrez. For the purposes of our evaluation, we measure the processing time inside *In-Memory DB*, ignoring communication costs. We report performance in terms of wall-clock time. All results are averages of three executions.

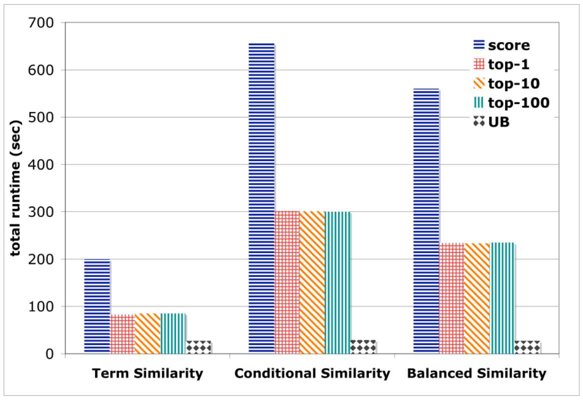


Fig. 5. Total runtime of ranked retrieval.

B. Workload

Our performance experiments are based on a workload of 150 queries. We were unable to get a real PubMed query workload from NCBI due to privacy regulations, and so we generated the workload based on pairwise co-occurrence of terms in annotations of PubMed articles. The rationale is that, if two or more terms are commonly used to annotate the same document, then these terms are semantically related and may be used together in a query. See [15] for a detailed description of the workload selection procedure. Queries in our workload mapped to MeSH subtrees of different sizes: $L^*(Q)$ was between 2 and 454, median 22. Queries returned between 1,024 and 179,450 results, median 9,562.

C. Ranked Retrieval with Score Upper-Bounds

Table I summarizes the performance of 150 queries with *term similarity*, *conditional similarity*, and *balanced similarity*. We compare the execution time of computing exact scores for all results (**Score**) against the time of computing score upper-bounds for all results (**UB**). We then report the runtime of computing the top-1, top-10, and top-100 results, in which upper bounds are computed for all items, and exact scores are computed only for the promising items. We observe that execution time of **Score** can be high, particularly for conditional and balanced similarity. In contrast, upper bounds can be computed about an order of magnitude faster, in interactive time even in the worst case. This is expected, since, as we discussed in Section III-B, the time to compute upper bounds is proportional to $|D| * |Q|$, while the time to compute scores is a function of the size of the term-scope of the query and of the document, which is typically much higher.

Figure 5 compares the *total run-time* of **Score**, **UB**, and ranked retrieval with $k = 1, 10, 100$, for all queries. Observe that *term similarity* computes fastest, while *conditional similarity* is slowest. It takes approximately the same amount of time to compute the top-k for different values of k.

TABLE I

RANKED RETRIEVAL: MEDIAN, AVERAGE, MINIMUM AND MAXIMUM PROCESSING TIMES FOR 150 QUERIES.

	Term Similarity(sec)				Conditional Similarity(sec)				Balanced Similarity(sec)			
	med	avg	min	max	med	avg	min	max	med	avg	min	max
Score	0.412	1.342	0.013	13.238	0.387	4.408	0.004	274.230	0.372	3.760	0.006	195.420
UB	0.062	0.177	0.005	1.242	0.060	0.195	0.005	2.210	0.059	0.177	0.005	1.236
top-1	0.228	0.557	0.009	5.127	0.273	2.016	0.010	83.063	0.246	1.558	0.009	55.365
top-10	0.228	0.566	0.009	5.128	0.273	2.010	0.010	84.063	0.245	1.550	0.010	55.441
top-100	0.228	0.568	0.010	5.092	0.273	2.001	0.014	83.132	0.246	1.566	0.012	55.444

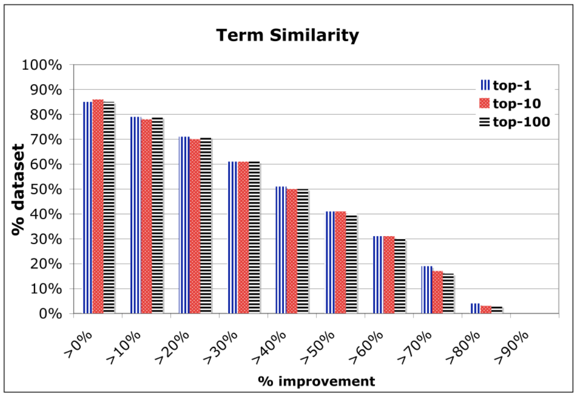


Fig. 6. *Term similarity*: percent improvement in runtime of top-K when score upper-bounds are used.

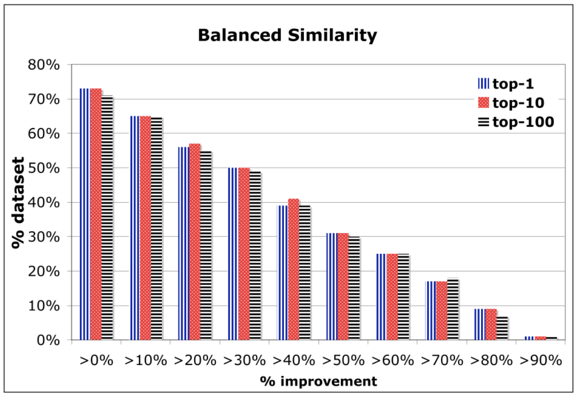


Fig. 8. *Balanced similarity*: percent improvement in runtime of top-K when score upper-bounds are used.

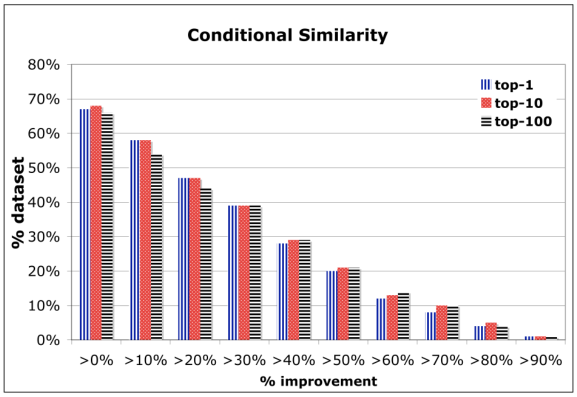


Fig. 7. *Conditional similarity*: percent improvement in runtime of top-K when score upper-bounds are used.

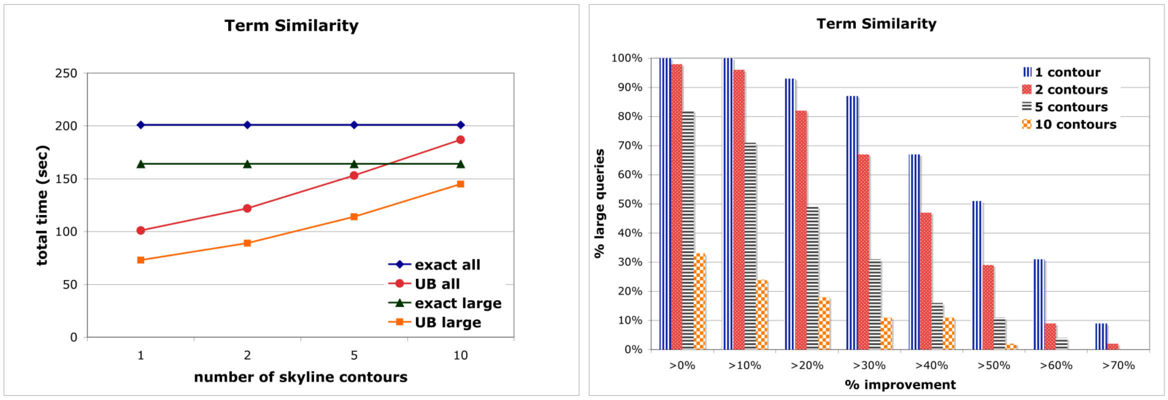
Figures 6, 7 and 8 present run-time improvement of using score upper-bounds for top-k computation vs. computing exact scores, for three similarity measures. Performance of the vast majority of queries is improved due to using upper-bounds, for all similarity measures. The actual run-time improvement was up to 9.1 sec for *term similarity*, and between 0.7 and 0.8 sec on average for different values of k. For *conditional similarity*, the improvement was up to a dramatic 191 sec, and the average improvement was about 2.4 sec. For *balanced similarity*, using score upper-bounds improved run-time by up to 140 sec, and between 2.0 and 2.2 sec on average, for different values of k.

While performance improved for most queries, it degraded for some queries due to the overhead of sorting. This overhead was noticeable only in short-running queries, and absolute degradation was insignificant: at most 0.081 sec for *TermSim*, 0.254 sec for *CondSim* and 0.213 sec for *BalancedSim*.

D. Skyline Computation with Upper-Bounds

In this section we consider the performance impact of using score upper-bounds for skyline computation, described in Section III-C. We computed the skyline with 1, 2, 5, and 10 contours for 150 queries in our workload. Table II presents the median, average, minimum, and maximum execution time for three similarity measures. For each number of contours, and for each similarity measure, we list two sets of numbers. The **Exact** line lists the performance of computing the skyline without the upper-bounds optimization, and the **UB** line lists the performance with the optimization. Recall that, whether we first compute exact scores for all documents (as in **Exact**), or first compute score upper-bounds for all documents, and then compute exact scores only for promising documents (as in **UB**), the result will be the same correct set of skyline points.

Based on Table II we observe that the **Exact** skyline performs in interactive time for the majority of queries, for all similarity measures. Median results are sub-second in all cases. We also observe that **UB** skyline outperforms **Exact** skyline. Note that these results are for the total execution of each query. Long-running queries typically execute in multiple batches, and the user is presented with the initial set of results as soon as the skyline of the first batch is computed.



(a) Total run-time.

(b) % improvement with UB for *large queries*.

Fig. 9. Run-time performance of skyline computation for *term similarity.*

TABLE II

SKYLINE COMPUTATION: MEDIAN, AVERAGE, MINIMUM AND MAXIMUM PROCESSING TIMES FOR 150 QUERIES.

	K	Term Similarity(sec)				Conditional Similarity(sec)				Balanced Similarity(sec)			
		med	avg	min	max	med	avg	min	max	med	avg	min	max
Exact	1	.4295	1.356	.016	13.225	.4305	4.471	.009	274.301	.4275	3.861	.01	199.521
UB	1	.299	.684	.014	5.687	.343	2.377	.017	107.673	.3345	1.925	.018	72.955
Exact	2	.43	1.355	.016	13.202	.424	4.471	.008	274.296	.433	3.833	.01	195.194
UB	2	.3705	.825	.018	6.771	.4055	2.796	.019	136.881	.389	2.345	.019	95.654
Exact	5	.4285	1.356	.016	13.209	.4285	4.473	.009	274.311	.4365	3.803	.01	195.13
UB	5	.448	1.036	.022	8.263	.4855	3.351	.019	167.917	.4785	2.841	.02	116.974
Exact	10	.4295	1.358	.016	13.222	.4275	4.472	.008	274.305	.4365	3.806	.01	195.163
UB	10	.4835	1.265	.022	9.647	.546	3.859	.019	197.304	.506	3.17	.019	132.334

In our experiments, we are able to predict whether a query will be long-running based on the number of results that the query returns. In fact, exact skyline computation for all queries that return fewer than 20,000 results completes in under 2 seconds. The information about the size of the result set is provided to us at the start of the execution by the **eUtils** API, and we can use this information to decide whether to apply the upper-bounds optimization. 45 out of 150 queries in our workload return over 20,000 results, and we refer to these as the *large queries* in the remainder of this section.

Figure 9(a) summarizes the total cumulative run-time of **Exact** and **UB** skylines for *term similarity* for all queries (*exact all* and *UB all* entries), and for the large queries (*exact large* and *UB large*). We note that over 75% of the time is spent processing 30% of the workload. The time to compute the exact skyline stays approximately the same as the number of contours changes, while the time to compute the UB skyline increases with increasing number of contours. Finally, observe that UB skylines compute faster in total than do exact skylines. The same trends hold for *conditional similarity* and *balanced similarity* (see [15]). Figure 9(b) plots the percent-improvement of **UB** skyline over **Exact** against the percentage of the *large queries* for which this improvement was realized. Query execution time was improved for the vast majority of large queries. Similar trends hold for *conditional similarity* and *balanced similarity* (see [15]).

V. EVALUATION OF EFFECTIVENESS

We now present a qualitative comparison between our similarity measures, and evaluate them against two baselines.

A. Baselines

Our first baseline is a *distance-based measure*, designed explicitly for MeSH, that compares two sets of terms based on the mean path-length between the individual terms [13]. For terms d and q, $dist(d, q)$ is the minimal number of edges in a path from any node in $N^*(d)$ to and node in $N^*(q)$. Consider nodes C and F in Figure 2. There are two paths between these nodes: $C \rightarrow A \rightarrow E \rightarrow F$ of length 3, and $C \rightarrow B \rightarrow F$ of length 2, and so $dist(C, F) = 2$. We define path-length as: $MeanPathLen(D, Q) = \frac{1}{|D||Q|} \sum_{d \in D} \sum_{q \in Q} dist(d, q)$. This measure captures the distance between document D and query Q, and we transform it into a similarity:

$$MeanPathSim(D, Q) = \frac{1}{1 + MeanPathLen(D, Q)} \quad (4)$$

A known limitation of distance-based measures is an implicit assumption that edges in the taxonomy represent uniform conceptual distances, which does not always hold in practice. In Figure 2, the path distance between G and A is 2, the same as between G and B. However, one can argue that G is more closely related to B than to A because B has a smaller subtree, and so G represents a larger portion of the *meaning* of B than of A. Several information-theoretic measures have been proposed to overcome this limitation, and we use the one proposed by Lin [12] to derive our second baseline.

For two taxonomy nodes s and t, we denote the lowest common ancestor by $LCA(s,t)$. The *information content* of a node s, denoted by $P(s)$, is the size of the subtree induced by s. Lin [12] defines similarity between nodes s and t as: $sim(s,t) = \frac{2 \times logP(LCA(s,t))}{logP(s) + logP(t)}$.

To use this similarity for MeSH, we need to apply it to a polyhierarchy, with multiple nodes per term. We take a similar approach as in $MeanPathSim$, and say that the similarity between terms d and q is the highest similarity between any two nodes s and t, where $s \in N^*(d)$ and $t \in N^*(q)$. To handle multiple terms per query and per document, we define:

$$MeanInfoSim(D,Q) = \frac{1}{|D||Q|} \sum_{d \in D} \sum_{q \in Q} sim(d,q) \qquad (5)$$

B. User Study

1) Methodology: We recruited 8 researchers, all holding advanced degrees in medicine, biology, or bioinformatics. All are experienced PubMed users, with usage between several times a week and several times a day. Users were asked to come up with one query in their field of expertise, and to subsequently rate results returned by our system.

Rather than rating articles in the result, we asked our users to rate *annotation sets*: sets of MeSH terms that occur together as annotations of these articles, for two reasons. First, MeSH annotations of some articles are imprecise, that is, more general or more specific than the content of the article warrants. Second, abstracts of articles are often unavailable, making it difficult to judge the quality of content.

For a fixed query, and for a fixed similarity, all articles that are annotated with the same set of terms receive the same score. Additionally, several different annotation sets may map to the same score, and so ties are common. Scores are incomparable across measures, and we use ranks for our comparison. See [15] for a description of our rank assignment procedure that meaningfully accommodates ties.

Many queries return thousands of results, and we cannot expect that the users will evaluate the quality of results exhaustively. We focus on a sub-set of results that is most informative about either the performance of a particular similarity measure, or about the relative performance of a pair of measures. For a pair of measures $\mathcal{M}_1$ and $\mathcal{M}_2$, we choose 10 results from each of the following categories:

- $top\mathcal{M}_1$: in top 10% of ranks for $\mathcal{M}_1$ but not for $\mathcal{M}_2$
- $top\mathcal{M}_2$: in top 10% of ranks for $\mathcal{M}_2$ but not for $\mathcal{M}_1$
- $bot\mathcal{M}_1$: in bottom 10% of ranks for $\mathcal{M}_1$ but not for $\mathcal{M}_2$
- $bot\mathcal{M}_2$: in bottom 10% of ranks for $\mathcal{M}_2$ but not for $\mathcal{M}_1$

TABLE III

AGREEMENT BETWEEN SIMILARITY MEASURES AND USER JUDGMENTS.

	TermSim	**CondSim**	**BalSim**	**MeanPath**	**MeanInfo**
Q_1	0.56	0.51	0.51	**0.71**	**0.65**
Q_2	0.49	0.50	0.50	0.52	0.49
Q_3	**0.67**	**0.63**	**0.63**	0.39	0.48
Q_4	**0.66**	**0.66**	**0.66**	0.40	0.48
Q_5	0.42	0.43	0.43	0.33	**0.67**
Q_6	0.48	0.51	**0.60**	0.48	0.50
Q_7	0.43	0.45	0.45	**0.63**	0.44
Q_8	0.47	0.47	0.47	0.31	0.57
Avg	0.52	0.52	0.53	0.47	0.54

Results are chosen to maximize rank distances. Finally, we generate pairs of results to be compared to each other by the user. We never compare $top\mathcal{M}_1$ to $top\mathcal{M}_2$, and $bottom\mathcal{M}_1$ to $bottom\mathcal{M}_2$. Comparing top against bottom for the same method helps us validate that method on its own. Comparing top of one method against bottom of another allows us to compare a pair of methods against each other.

The user is presented with two annotation sets, **Match 1** and **Match 2**, and rates each set on a three-point scale. The user also compares the matches with respect to how well they answer the query, on a three-point scale. Both scales include a "not sure" option. See [15] for a screenshot of our evaluation interface.

2) Results: Results in this section are based on 8 queries, each evaluated by a single user. We collected 670 individual judgments, and 335 pairwise judgments. In this section, we analyze the performance of each of our similarity measures individually, and then describe the relative performance of our measures, and compare them to the baselines. For results r_1 and r_2, user $\mathcal{U}$ issues a pair-wise relevance judgment $\mathcal{U} : r_1 = r_2$ if he considers results to be of equal quality, $\mathcal{U} : r_1 > r_2$ if r_1 is better, or $\mathcal{U} : r_1 < r_2$ if r_2 is better. (We exclude the cases where the user was unable to compare the results.) Likewise, a similarity measure $\mathcal{M}$ issues a judgment w.r.t. the relative quality of r_1 and r_2 by assigning ranks. Because users only judge a pair of results that are far apart in the ranking, the case $\mathcal{M} : r_1 = r_2$ never occurs.

A similarity measure may agree with the user's assessment, or it may disagree, in one of two ways: by reversing the rank order of r_1 and r_2, or by ranking r_1 and r_2 differently while the user considers them a tie. For ease of exposition, we incorporate all three outcomes: *agreement (A)*, *tie (T)* and *rank reversal error(E)*, into a single *agreement score*, defined as: $agreement(\mathcal{U}, \mathcal{M}, Q) = \frac{A + 0.5 * T}{A + T + E}$. The worst possible score is 0, the best possible is 1. Table III presents the agreement between the user and each similarity measure, for each query. See [15] for a break-down of results in each category.

Due to the scale of our study we are unable to draw statistically significant conclusions about the relative performance of the measures. However, we point out some trends that emerge based on the data in Table III, and which we plan to investigate further in the future; see Section V-C for a

TABLE IV

TermSim, *CondSim* AND *BalancedSim* COMPARED TO BASELINES.

	MeanPath	***MeanInfo***
TermSim	46% / *28%*	31% / *36%*
CondSim	41% / *31%*	36% / *36%*
BalSim	42% / *29%*	36% / *35%*

discussion. None of the measures seem to agree with user's judgment for queries Q_2 and Q_8. These queries do not exhibit polyhierarchy features: each term maps to a single node in MeSH. Our measures appear to outperform the baselines for queries Q_3, Q_4, and Q_6. All these queries include at least one term that exhibits polyhierarchy features: either the term itself *maps to two or more nodes* and *induces subtrees of different shape*, or its descendant terms do. Baselines appear to outperform our measures for queries Q_1, Q_5, and Q_7. Query Q_1 exhibits no polyhierarchy features. For a two-term query Q_8, each term maps to two nodes in MeSH, but the subtrees are isomorphic, i.e., there is *structural redundancy* in this part of the hierarchy. Query Q_5 exhibits true polyhierarchy features, yet the information theoretic baseline seems to be more in line with the user's judgment for this query.

Table IV presents the relative performance of our measures against the baselines. We present averages across queries, but note that performance for individual queries is in line with the trends in Table III. Here, we are using judgments about pairs of results such that one of the results has a high rank w.r.t. one method and a low rank w.r.t. another. We present the average percentage of user judgments that were in line with the judgment made by the similarity measure. For example, in the entry for *TermSim* and *MeanPath* the user agreed with *TermSim* 46% of the time, and with *MeanPath* 28% of the time, and considered the remaining 26% of the cases as ties.

We also compared the relative performance of our measures for queries, for which there was a difference in performance. For Q_6, *BalancedSim* outperforms *CondSim*, which in turn outperforms *TermSim*. For Q_3, *TermSim* outperforms other measures. These findings are in line with results in Table III.

C. Assessment of Results

Several issues make ranking difficult in our context. First, all results are already matches, i.e., all are in some sense "good". So, ranking by ontology is a second-order ranking among documents that may not be all that different from each other in terms of real relevance. However, as we demonstrate in Section V-B.2, ontology-related score *is* correlated with quality as judged by the users in some cases. This occurs when terms appear in multiple tree locations and induce subtrees of different shape, a distinguishing feature in MeSH. Second, our user study is small, and so we cannot expect to demonstrate statistical significance. We plan to deploy the system and obtain more information by studying user feedback.

A user's perception of quality is informed by many aspects. Our work is motivated by the hypothesis that one of these aspects is captured by ontological relationships. This was supported by observations made by several users that they appreciated the presence of both general concepts, e.g., *Neurodegenerative Disease*, and related concepts that are more specific, e.g., *Alzheimer* and *Parkinson*.

Nonetheless, other aspects of user's quality perception may require a more sophisticated ontology than MeSH. Even when the ontology is helpful in principle, users may disagree with the classification, as observed by one user in our study. Semantic relationships, e.g., a connection between a protein and a disease, may be known to experts but are not present in MeSH, and are therefore unavailable for scoring. In future work, we plan to combine MeSH with other information sources that provide additional information about relationships between concepts. We also plan to incorporate weighting of terms, perhaps on a user by user basis, based on external information.

Due to the scale of our study, we do not establish which ranking is best for which kind of query, and when a query is amenable to ontology-aware ranking. We will investigate this in the future. For some queries our methods appear to do better, while for others the competing methods appear to do better. While no method dominates another for all queries, our methods seem to outperform the path-based method, while performing comparably with the information theoretic method.

VI. Related Work

Ranking that takes into account hierarchical structure of the domain has been considered in the literature. Ganesan et al. [6] develop several families of similarity measures that relate sets or multisets of hierarchically classified items, such as two customers who buy one or several instances of the same product, or who buy several products in the same hierarchy. This work assumes that items in the sets are confined to being leaves of the hierarchy, and that the hierarchy is a strict tree. In our work we are comparing sets of terms in a scoped polyhierarchy, and we do not restrict the terms to being leaves.

Rada and Bicknell [13] consider the problem of ranking MEDLINE documents using the MeSH polyhierarchy, the same problem as we consider in our work. The authors propose to model the distance between the query and the document as the mean path-length between all pairs of document and query terms. This measure is one of several distance-based measures that have been proposed in the literature; see also [11].

In an alternative approach, several information-theoretic measures have been proposed that quantify the semantic relatedness between concepts in hierarchical domains [12], [14]. These measures are similar to the distance-based methods in that they typically relate two concepts via a common ancestor. However, rather than simply counting the length of the path to the ancestor, the information content of the ancestor (the size of its subtree) is factored into the measure. The intuition is that a common ancestor that is very general is not as informative as one that is more specific.

In our work we propose several alternative ways to relate a document to a query, by measuring the overlap among common descendants (rather than ancestors) of all nodes labeled with two concepts. To the best of our knowledge, our work is the first to explicitly model semantic relatedness in a *scoped* polyhierarchy in which a term may appear in many parts of the hierarchy with subtly different meanings in each context. The question of how contributions of different terms, or different meanings of the same term, are reconciled in the final score is central to our approach. We explicitly model and explore alternative semantics of combining the contributions of individual pairs of terms to the over-all similarity score. Despite the extra computation needed for measures based on sets of descendants rather than ancestors, we demonstrate experimentally that interactive response times are still possible even when processing tens of thousands of documents.

Hadjieleftheriou et al [7] develop indexing structures and processing algorithms for computing the similarity of weighted sets. A query Q and a document D are compared based on corpus-derived weights of substrings of length q, termed q-grams, of which Q and D are comprised. The similarity between a query and a document is computed based on the combined weight of the q-grams that are common to D and Q, normalized by the weights of D and Q. This approach is similar to ours in that we also consider the set of elements, in our case MeSH terms, that are common to D and Q for the computation of similarity. However, unlike in [7], the elements we consider come from a hierarchy, and incorporating the structure of the hierarchy into the similarity score is central to our approach. We do not currently associate weights with query terms; incorporating term weights is an interesting direction for future research.

Ontology matching uses a wide range of similarity measures to compare two or more ontologies. Ontology matching techniques in which comparison is based on taxonomic structure, bear some similarity to our approach. A particular class of similarity measures represents ontologies as labeled graphs and compares nodes in these graphs using lexical and structural features [4]. Pairwise node similarities are then aggregated into collection-wide measures. In our work we focus on structural similarity between sets of ontology terms, and consider them in the context of scoped polyhierarchies that do not naturally lend themselves to a graph-based representation.

A variety of web-based systems for bibliographic search in life sciences have been developed, see [10] for a review. The system that is closest to our approach, GoPubMed [5], uses three ontologies - the Gene Ontology, MeSH and Uniprot, to organize PubMed query results. Results are presented in a faceted hierarchy that includes ontology terms, authors, journals, and publication dates. When multiple MeSH terms appear in the query or annotate query results, the system allows the user to navigate by each of these terms. Unlike in our work, no attempt is made to reconcile the contributions of multiple MeSH terms into a single score.

Efficient computation of skyline results has been receiving significant attention in the database community. We build on the classic divide-and-conquer algorithm by Bentley [2], and adapt it to our application scenario and performance needs by incorporating score upper-bounds. Tan et al. [16] develop progressive skyline computation methods, while Jin et al. [8] propose an efficient algorithm for the mining of thick skylines in large databases. Our scenario differs from prior work in that coordinates of skyline points may be costly to compute, motivating us to use score upper-bounds.

VII. Conclusions

MeSH is a sophisticated, curated real-world ontology with about 25,000 terms. It has the interesting property that terms can appear in multiple parts of the hierarchy. Each time a term appears, its meaning is scoped, i.e., the meaning of the term depends on its position in the hierarchy. This observation challenges most past work which has been developed assuming that a term has a unique node in the generalization hierarchy.

We have attempted to capture the semantics of a term by looking at all of the term's descendants, across the whole hierarchy. We developed three similarity measures that relate sets of terms based on the degree of overlap between the sets of their descendants. We have demonstrated that each of these measures can be computed in interactive time for the complete MeSH ontology, at the scale of the complete PubMed corpus. We have also shown how computing score upper-bounds can be used to reduce the cost of identifying the best-matching documents, or of computing the skyline of the dataset with respect to score and publication date. We evaluated our similarity measures with a user study.

Acknowledgments

This research was supported by National Institute of Health grant 5 U54 CA121852-05.

References

[1] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. 1999.

[2] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23, 1980.

[3] S. Börzsönyi, D. Kossman, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[4] J. David and J. Euzenat. Comparison between ontology distances (preliminary results). In *International Semantic Web Conference*, 2008.

[5] A. Doms and M. Schroeder. GoPubMed: Exploring PubMed with the GeneOntology. 33, 2005.

[6] P. Ganesan, H. Garcia-Molina, and J. Widom. Exploring hierarchical domain structure to compute similarity. *ACM TOIS*, 21(1), 2003.

[7] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava. Fast indexes and algorithms for set similarity selection queries. In *ICDE*, 2008.

[8] W. Jin, J. Han, and M. Ester. Mining thick skylines over large databases. In *PKDD*, 2004.

[9] J. Kaiser. *Systematic indexing*. London, Pitman, 1911.

[10] J.-J. Kim and D. Rebholz-Schuhmann. Categorization of services for seeking information in biomedical literature: a typology for improvement of practice. 9(6), 2008.

[11] J. Lee and M. Kim. Information retrieval based on a conceptual distance in is-a hierarchy. *J Doc*, 49, 1993.

[12] D. Lin. An information-theoretic definition of similarity. In *ICML*, 1998.

[13] R. Rada and E. Bicknell. Ranking documents with a thesaurus. *JASIS*, 40(5), 1989.

[14] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, 1995.

[15] J. Stoyanovich, W. Mee, and K. A. Ross. Semantic ranking and result visualization for life sciences publications. *Columbia University Technical Report cucs-028-09*, 2009.

[16] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.