

Effective Navigation of Query Results Based on Concept Hierarchies

Abhijith Kashyap, Vagelis Hristidis, Michalis Petropoulos, and Sotiria Tavoulari

Abstract— Search queries on biomedical databases, such as PubMed, often return a large number of results, only a small subset of which is relevant to the user. Ranking and categorization, which can also be combined, have been proposed to alleviate this information overload problem. Results categorization for biomedical databases is the focus of this work. A natural way to organize biomedical citations is according to their MeSH annotations. MeSH is a comprehensive concept hierarchy used by PubMed. In this paper, we present the BioNav system, a novel search interface that enables the user to navigate large number of query results by organizing them using the MeSH concept hierarchy. First, the query results are organized into a navigation tree. At each node expansion step, BioNav reveals only a small subset of the concept nodes, selected such that the expected user navigation cost is minimized. In contrast, previous works expand the hierarchy in a predefined static manner, without navigation cost modeling. We show that the problem of selecting the best concepts to reveal at each node expansion is NP-complete and propose an efficient heuristic as well as a feasible optimal algorithm for relatively small trees. We show experimentally that BioNav outperforms state-of-the-art categorization systems by up to an order of magnitude, with respect to the user navigation cost. BioNav for the MEDLINE database is available at <http://db.cse.buffalo.edu/bionav>.

Index Terms—Interactive data exploration and discovery, Search process, Graphical user interfaces, Interaction styles.

1 INTRODUCTION

The last decade has been marked by unprecedented growth in both the production of biomedical data and the amount of published literature discussing it. The MEDLINE database, on which the PubMed search engine operates, contains over 18 million citations and is currently growing at the rate of 500,000 new citations each year [20]. Other biological sources, such as Entrez Gene [18] and OMIM [21], witness similar growth. As claimed in previous work [26], the ability to rapidly survey this literature constitutes a necessary step toward both the design and the interpretation of any large scale experiment. Biologists, chemists, medical and health scientists are used to searching their domain literature –such as PubMed– using a keyword search interface. Currently, in an exploratory scenario where the user tries to find citations relevant to her line of research and hence not known a priori, she submits an initially broad keyword-based query that typically returns a large number of results. Subsequently, the user iteratively refines the query, if she has an idea of how to, by adding more keywords, and re-submits it, until a relatively small number of results are returned. This refinement process is problematic because after a number

of iterations the user is not aware if she has over-specified the query, in which case relevant citations might be excluded from the final query result.

As an example, a query on PubMed for “cancer” returns more than 2 million citations. A more specific query, “breast cancer treatment”, returns 111,433 citations. Our running example query for “prothymosin”, a nucleoprotein gaining attention for its putative role in cancer development, returns 313 citations. The size of the query result makes it difficult for the user to find the citations that she is most interested in, and a large amount of effort is expended searching for these results. Many solutions have been proposed to address this problem – commonly referred to as *information overload* [1,2,3,9,16]. These approaches can be broadly classified into two classes: ranking and categorization - which can also be combined. Ranking presents the user with a list of results ordered by some metric of relevance [9] or by content similarity to a result or a set of results [16]. In categorization [1,2,3], query results are grouped based on hierarchies, keywords, tags or attribute values. User studies have demonstrated the usefulness of categorization in finding relevant results of exploratory queries [12]. While ranked results are useful when the ranking function is aligned with user preferences or the result list is small in size, categorization is generally employed by users when ranking fails or the query is too “broad” [12].

BioNav belongs primarily to the categorization class, which is especially suitable for this domain given the rich concept hierarchies (e.g., MeSH [19]) available for biomedical data. We augment our categorization techniques with simple ranking techniques. BioNav organizes the query results into a dynamic hierarchy, the *navigation tree*. Each concept (node) of the hierarchy has a descriptive label. The user then navigates this tree structure, in a top-

- A. Kashyap and M. Petropoulos are with the Department of Computer Science and Engineering, University at Buffalo, SUNY, Buffalo, NY 14260. E-mail: {rk39,mpetro}@cse.buffalo.edu.
- V. Hristidis is with the School of Computing and Information Sciences, Florida International University, Miami FL 33199. E-mail: vagelis@cis.fiu.edu.
- Sotiria Tavoulari is with the Department of Pharmacology, Yale University, New Haven, CT 06520. E-mail: sotiria.tavoulari@yale.edu.

Manuscript received 26 Jan. 2009; revised 11 July 2009; accepted 16 Sept. 2009; published online 12 Aug. 2010.
Recommended for acceptance by D. Cook.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2009-01-0038.
Digital Object Identifier no. 10.1109/TKDE.2010.135.

down fashion, exploring the concepts of interest while ignoring the rest.

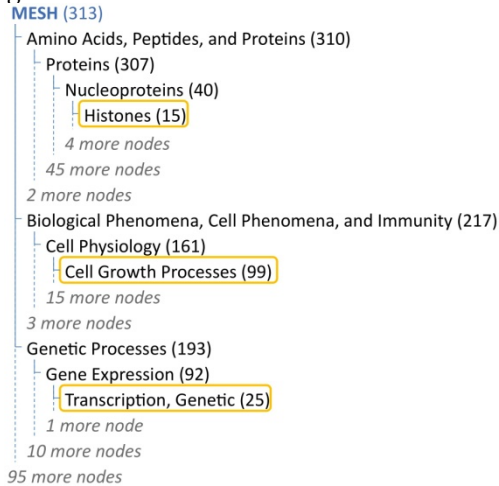


Fig. 1. Static Navigation on the MeSH Concept Hierarchy¹

An intuitive way to categorize the results of a query on PubMed is by using the MeSH static concept hierarchy [19], thus utilizing the initiative of the US National Library of Medicine (NLM) to build and maintain such a comprehensive structure. Each citation in MEDLINE is associated with several MeSH concepts in two ways: (i) by being explicitly annotated with them, and (ii) by mentioning those in their text (see Section 7 for details). Since these associations are provided by PubMed, a relatively straightforward interface to navigate the query result would first attach the citations to the corresponding MeSH concept nodes and then let the user navigate the navigation tree. Fig. 1 displays a snapshot of such an interface where shown next to each node label is the count of distinct citations in the subtree rooted at that node. A typical navigation starts by revealing the children of the root ranked by their citation count, and is continued by the user expanding on or more of them, revealing their ranked children and so on, until she clicks on a concept and inspects the citations attached to it. A similar interface and navigation method is used by e-commerce sites, such as Amazon and eBay. For this example interaction, we assume that some of the citations the user is interested in are available on the three indicated concepts corresponding to three independent lines of research related to prothymosin, and therefore the user is interested in navigating to these concepts. These include, “Histones”, which play a role in gene regulation and are essential for virus replication and tumor growth, “Cell Growth Processes” and “Transcription, Genetic”, a key process for synthesis and replication of RNA and thus plays an important role in the duplication of cancer cells.

Note that the user is not aware that the relevant results are available specifically on these nodes – she is only interested in narrowing down the results, using a familiar concept hierarchy, instead of examining all the results.

The above *static* –same for every query result– navigation method is problematic when the MeSH hierarchy (or

one with similar properties) is used for categorization for the following reasons:



Fig. 2. Dynamic navigation steps to reach the concept “Histones” for the query “prothymosin”

- The massive size of the MeSH hierarchy (over 48,000 concept nodes) makes it challenging for the users to effectively navigate to the desired concepts and browse the associated records. Even if we remove from the MeSH concept nodes with no citations attached to them, the 313 distinct query results for “prothymosin” are attached to 3,940 nodes, which is the actual size of the navigation tree in Fig. 1. Combined with the fact that the MeSH hierarchy is quite bushy on the upper levels, this means that the user has to inspect, for example, a total of 152 concept nodes before she reaches the indicated concept “Histones”; a number comparable to the distinct citation count in the query result. A common practice [28] for hierarchy navigation is to show only a subset of a node’s children, which would be appropriate if only few nodes contain many results. Unfortunately, this is not the case for the MeSH navigation tree; most of

¹ The complete tree can be seen at <http://db.cse.buffalo.edu/allnav>

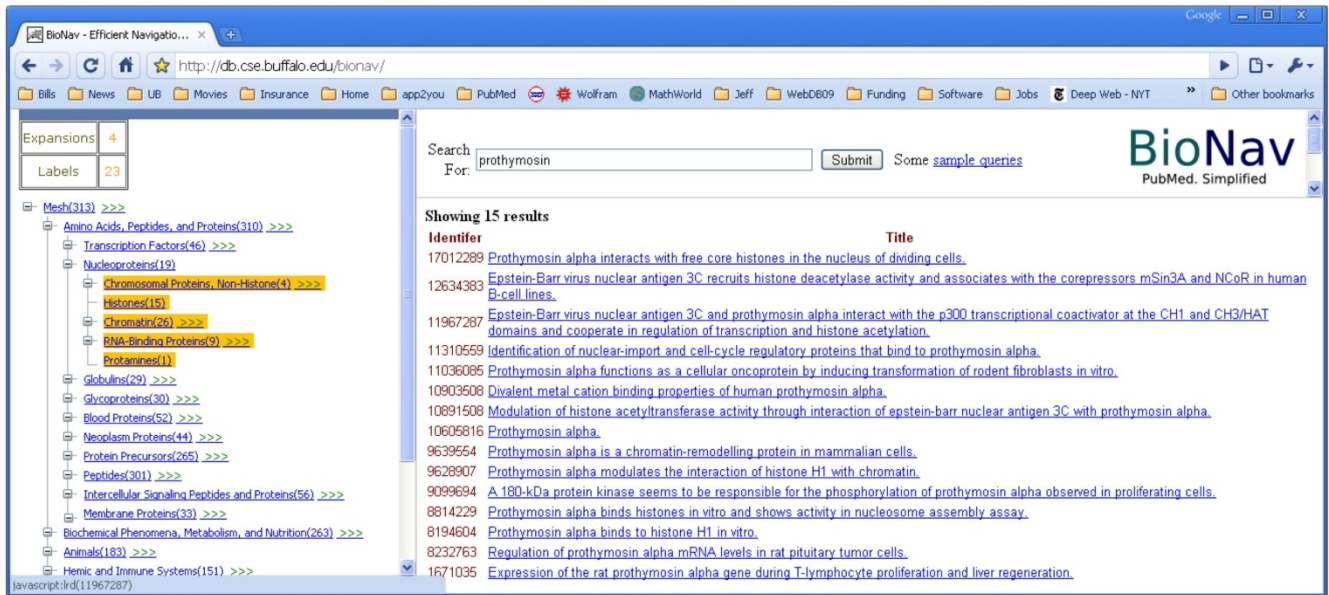


Fig. 3. BioNav Interface

the 98 children of the root in Fig. 1 have many results (the first three shown have 310, 217 and 193).

- A substantial number of *duplicate* citations are introduced in the navigation tree of Fig. 1, since each one of the 313 distinct citations is associated with several concepts. Specifically, the total count of citations in Fig. 1 is 30,895. Naturally, the user would like to know which concepts fragment the query result into subsets of citations with as few duplicate citations as possible across them. Currently, the only way to figure this out using the interface in Fig. 1 is to click on different concept nodes and inspect the attached citations. As an example, the query results for “prothymosin” are related to three independent lines of research, represented by the three indicated concepts in Fig. 1, which are hard to locate. Among the total 139 citations attached to the three indicated concept nodes, only 20 of them are duplicates.

BioNav introduces a *dynamic* navigation method that depends on the particular query result at hand and is demonstrated in Fig. 2. The query results are attached to the corresponding MeSH concept nodes as in Fig. 1, but then the navigation proceeds differently. The key action on the interface is the *expansion* of a node that selectively reveals a ranked list of descendant (not necessarily children) concepts, instead of simply showing all its children.

Fig. 2a, for example, shows the initial expansion of the root node where only 8 (highlighted) descendants are revealed compared to 98 children shown in Fig. 1. The concepts are ranked by their relevance to the user query and the number of them revealed depends on the characteristics of the query results. Next, assuming the user is interested in the “Amino Acids...” node and judging that the 310 attached citations is still a big number, she expands it by clicking on the “>>>” hyperlink next to it in Fig. 2b. The user inspects the 6 concepts revealed and decides that she is not interested in any of them. Hence, she expands the “Amino Acids...” node one more time in Fig. 2c, revealing 4 additional concepts. Note that “Nuc-

leoproteins” is an example of a descendant node being revealed, since its parent node “Proteins” is not revealed in Fig. 2c. In Fig. 2d, the user expands the “Nucleoproteins” node and reveals “Histones”, one of the three key concepts for the query. In the last step of the interaction, the user clicks on the “Histones” hyperlink and the 15 corresponding citations are displayed in a separate frame as shown in Fig. 3.

To reach “Histones” using the BioNav navigation method only 23 concepts are revealed, after 4 node expansions, compared to 152 concepts, also after 4 expansions, with the static navigation method of Fig. 1.

For each expansion, the displayed descendant concepts are chosen in a way that the expected navigation cost is minimized, based on an intuitive navigation cost model we present in Section 3. The cost model estimates the exploration probability for a node based on its selectivity, that is, the ratio of attached citations before and after the query. The navigation cost for a concept node is also proportional to the density of the navigation subtree rooted at this node in terms of citation count. Intuitively, the selection is done such that every expansion reduces maximally the expected remaining navigation cost. For example, the reason that “Proteins” is not displayed in Fig. 2 is that it is too general given the query results and the original distribution of citations in the PubMed database (details in Sections 3 and 4), and hence displaying it would lead to an expected increase in the user navigation cost, based on the user navigation cost model.

In addition to the static hierarchy navigation works mentioned above, there are works on dynamic categorization of query results (e.g., the Clusty search engine [29], or [2], [3]), which create unsupervised query-dependent results clusters, but do not study how the clusters should be navigated. BioNav is distinct since it offers dynamic navigation on a predefined hierarchy, as is the MeSH concept hierarchy. Another difference is that BioNav uses a navigation cost model to minimize the navigation cost.

We make the following contributions:

1. A comprehensive framework for navigating large query results from PubMed using MeSH, an extensive concept hierarchy used for indexing citations in MEDLINE (Section 2).
2. A formal cost model for measuring the navigation cost incurred by the user (Sections 3 and 4).
3. A complexity result proving that expanding the tree in a way that minimizes the user's navigation cost is an NP-complete problem (Section 5).
4. An efficient heuristic and a feasible optimal algorithm for minimizing the navigation cost (Section 5).
5. Experimental results validating the effectiveness of the BioNav system when compared to state-of-the-art categorization systems (Section 8).
6. An online version of the BioNav system is available at <http://db.cse.buffalo.edu/bionav>.

Although we specifically target the biomedical domain in this work, the approach can be directly applied to datasets where tuples are classified using terms from a concept hierarchy. The core of the first contribution has been presented in our preliminary short paper [13]. The BioNav system architecture and implementation is presented in Section 7. Related work is discussed in Section 9 and the paper concludes in Section 10.

2 FRAMEWORK AND BIO NAV OVERVIEW

The MeSH concept hierarchy is the starting point of the framework and is defined as follows.

Definition 1 (Concept Hierarchy). A Concept Hierarchy $H(V, E, r)$ is a labeled tree consisting of a set V of concept nodes, a set E of edges and is rooted at node r . Each node $n \in V$ has a label l and a unique identifier id .

According to the semantics of the MeSH concept hierarchy [19], the label of a child concept node is more specific than the one of its parent. This also holds for most concept hierarchies.

Once the user issues a keyword query, PubMed –BioNav uses the Entrez Programming Utilities (eUtils) [7]–returns a list of citations, each associated with several MeSH concepts. BioNav constructs an Initial Navigation Tree by attaching to each concept node of the MeSH concept hierarchy a list of its associated citations. Formally, an Initial Navigation Tree $T_1(V_1, E_1, r)$ is a concept hierarchy, where every node (concept) $n \in V_1$ is additionally labelled with a results (citations) list $L(n)$.

In a given initial navigation tree, several concept nodes might have an empty results list. Since MeSH is a rather large concept hierarchy, BioNav reduces the size of the initial navigation tree by removing the nodes with empty results lists, while preserving the ancestor/descendant relationships. Formally, the resulting structure is defined as follows.

Definition 2 (Navigation Tree). A Navigation Tree $T(V, E, r)$ is the maximum embedding of an initial navigation tree $T_1(V_1, E_1, r)$ such that no node $n \in V$ is labeled with an empty results list $L(n)$, excluding the root (in order to maintain the tree structure and avoid a forest).

An embedding $T(V, E, r)$ of a tree $T_1(V_1, E_1, r)$ is an injection

from V to V_1 such that every edge in E corresponds to a path (disjoint from all other such paths) in T_1 . An embedding T of a tree T_1 , where both trees are rooted at node r , is maximum if no other node n with a nonempty results list $L(n)$ can be added to V and T still be an embedding. The maximum embedding of the initial navigation tree is recursively computed in a single depth-first left-to-right traversal. If a non-leaf node n has an empty results list $L(n)$, then add all children of n to the parent of n and remove it. If n is a leaf, then remove it. Fig. 4a shows part of the navigation tree for the “prothymosin” query, where the results lists are omitted for clarity.

The above procedure reduces the size of the initial navigation tree, but the structure is still too big (3,940 nodes for “prothymosin”) to simply display it to the user and let her navigate it. BioNav minimizes her effort to reach the desired citations in the navigation tree by expanding in a way that minimizes the expected overall user navigation cost. Moreover, BioNav avoids information clutter by hiding unimportant concept nodes leading to interesting ones. This is achieved through a series of *expand* actions that reveal only a few descendants (not necessarily children) of the user selected node for further navigation.

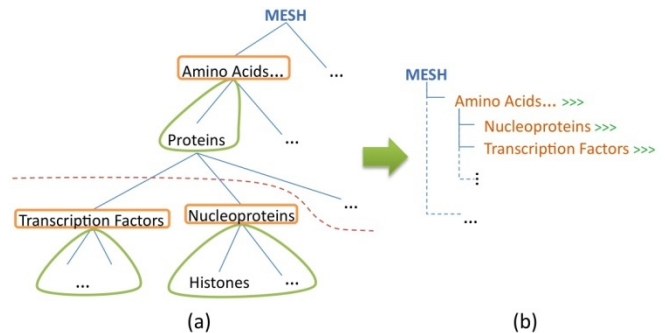


Fig. 4. (a) Navigation Tree, EdgeCut and Component Subtrees, (b) Visualization of the EdgeCut on the user interface.

We model a node expansion at a given navigation step as an *EdgeCut* in the navigation tree. In graph theory, an EdgeCut in a graph $G(V, E)$ is a set of edges $E_C \subseteq E$ such that the graph $G'(V, E \setminus E_C)$ has more components than G . For trees, any subset of the edges constitutes an EdgeCut, since the removal of any edge creates a new component.

In Fig. 4a, the dashed line illustrates the EdgeCut corresponding to the expansion of the node “Amino Acids...” and reveals the highlighted concepts of Fig. 4a. These revealed nodes are visualized on the interface as a tree shown in Fig. 4b. The EdgeCut consists of the edges (“Proteins”, “Transcription Factors”) and (“Proteins”, “Nucleoproteins”). Intuitively, an EdgeCut allows us to “skip” child nodes (“Proteins”), navigate directly to descendant nodes located deeper in the tree and show them as children of the node being expanded. Moreover, an EdgeCut can selectively reveal only a subset of a potentially large set of descendant nodes, as is the case in Fig. 2b where only 6 out of the 52 descendants of “Amino Acids...” are revealed.

Definition 3 (Valid EdgeCut). A valid EdgeCut of a tree

$T(V, E, r)$ is an EdgeCut $C \subseteq E$ such that no two edges in C appear in the same path from the root to some leaf node.

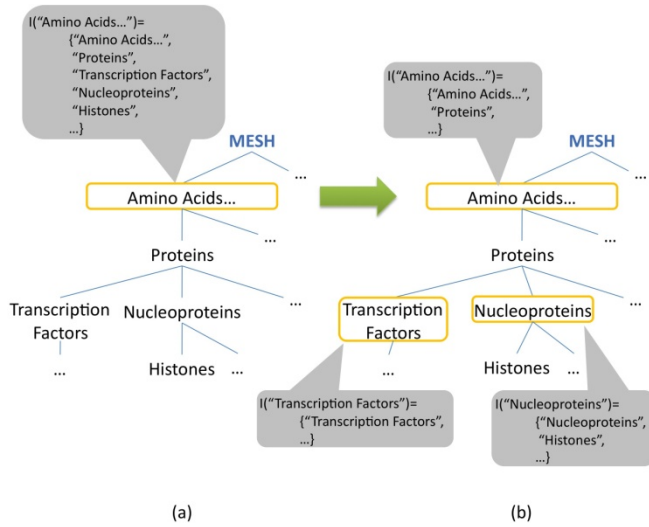


Fig. 5. The Active Tree Before and After the EdgeCut in Fig. 4

We only consider valid EdgeCuts in the rest of the paper, because invalid ones lead to unintuitive navigations.

Component Subtrees. An EdgeCut causes the creation of two types of *component subtrees*, *upper* and *lower*. Given an EdgeCut C of a tree $T(V, E, r)$, a *lower* component subtree $I(y_i)$ rooted at y_i is created by each node $y_i \in V$, such that $(x, y_i) \in C$ for some node x . In Fig. 2c, the expansion of “Amino Acids...” creates four lower component subtrees, two of which are shown in Fig. 4a, rooted at “Transcription Factors” and “Nucleoproteins”. Moreover, for a given EdgeCut C , a single *upper* component subtree is created consisting of the nodes not in any *lower* component subtree, and is always rooted at the root of the tree being expanded. In Fig. 4a, the upper component subtree comprises of the nodes “Amino Acids...” and “Proteins”.

The state of the navigation tree after an EdgeCut, and the component subtrees created, is captured by the *Active Tree* defined below.

Definition 4 (Active Tree). An Active Tree $T_A(V, E, r)$ is a Navigation Tree where each node $n \in V$ is annotated with a node set $I(n)$ consisting of the nodes in the component subtree rooted at n . If a node n is not a root of a component subtree, then $I(n) = \{n\}$. The non-singleton I sets are disjoint.

Before any EdgeCut, a navigation tree is trivially converted to an active tree by annotating the root node with an I set that includes all tree nodes. The rest of the nodes n_i are annotated with the node set $I(n_i) = \{n_i\}$. Fig. 5a shows (part of) the active tree capturing the state of the navigation tree before the EdgeCut in Fig. 4a (singleton I sets, such as “Histones”, are not shown).

An EdgeCut (expansion) is an operation on the active tree, performed on the I set of a given node, and updates the sets $I(n_i)$ of the roots n_i of the upper and lower subtrees created by the EdgeCut based on the nodes included in these subtrees. It is denoted by $\text{EdgeCut}: I(n) \rightarrow S \subseteq I(n)$ and returns the set S of roots of the upper and lower subtrees that it creates. Fig. 5b shows the effect of the Ed-

geCut operation in Fig. 4a on the active tree in Fig. 5a. The active tree is closed under the EdgeCut operation.

Note that the set $I(n)$ of a node n is overloaded to also denote the “invisible” component subtree of the active tree that is rooted at n and only consists of the nodes in $I(n)$. For instance, the invisible subtree $I(\text{“Amino Acids...”})$ in Fig. 5b is the one indicated as the upper component subtree in Fig. 4a.

BioNav visualizes the active tree to the user by showing only the nodes that do not appear in any non-singleton I set organized as follows.

Definition 5 (Active Tree Visualization). The visualization of an active tree $T_A(V, E, r)$ is the embedded tree $T_A'(V', E', r)$. V' are the nodes not in any non-singleton $I(n)$, for all $n \in V$. Shown next to every node $n \in V'$ is the number of distinct citations attached to nodes in $I(n)$, given by $|L(I(n))| = |\cup_{n_i \in I(n)} L(n_i)|$. If n has a non-singleton $I(n)$, then an expand hyperlink is shown next to it.

The visualization of the active tree after the EdgeCut in Fig. 4a is shown in Fig. 4b and is a subset of the nodes revealed in Fig. 2c. The citation count $|L(I(n))|$ for “Nucleoproteins” in Fig. 2c is 40 denoting the unique citations attached to it and its (invisible) component subtree. It is reduced to 19 in Fig. 2d, since its component subtree is getting smaller as descendant concept nodes are revealed.

An EdgeCut and the visualization of the resulting active tree are capable of reducing the navigation tree both height- and width-wise. The embedded tree in Fig. 2c, compared to the navigation tree in Fig. 1, is narrower and shorter. Note that we do not make any assumptions about the user’s preference over the tuples in the result and every citation in the result can be reached by a sequence of navigation actions, that is, there is no information loss in navigating the query results using our framework.

Using the “>>>” hyperlinks, the user can trigger subsequent EdgeCut operations on component subtrees in a recursive fashion. Although we expect the user to trigger EdgeCut operations predominantly on the lower component subtrees, an EdgeCut is possible on the upper subtree as well. For example, an EdgeCut operation on the upper component subtree of Fig. 4a would reveal the “Proteins” concept as parent of the previously revealed concept “Nucleoproteins”.

3 NAVIGATION AND COST MODEL

The navigation model of BioNav is formally defined in this section. Then the navigation cost model is presented, which is used to devise and evaluate our algorithms.

Navigation Model. After the user issues a keyword query, BioNav initiates a navigation by constructing the initial active tree (which has a single component tree rooted at the MeSH root) and displaying its root to the user. Subsequently, the user navigates the tree by performing one of the following actions on a given component subtree $I(n)$ rooted at concept node n :

1. **EXPAND $I(n)$:** The user clicks on the “>>>” hyperlink next to node n and causes an $\text{EdgeCut}(I(n))$ operation to be performed on it, thus revealing a new set of

- concept nodes from the set $I(n)$.
2. **SHOWRESULTS** $I(n)$: By performing this action, the user sees the results list $L(I(n))$ of citations attached to the component subtree $I(n)$.
 3. **IGNORE** $I(n)$: The user examines the label of concept node n , ignores it as unimportant and moves on to the next revealed concept.
 4. **BACKTRACK**: The user decides to undo the last EdgeCut operation.

This navigation process continues until the user finds *all* the citations she is interested in.

```

EXPLORE( $I(n)$ )
if  $n$  is the root
   $S \leftarrow$  EXPAND  $I(n)$  // that is  $S \leftarrow$  EdgeCut( $I(n)$ )
  For each  $n_i$  in  $S$ 
    EXPLORE( $I(n_i)$ )
else, if  $n$  is not a leaf-node, choose one of the following:
  1. SHOWRESULTS  $I(n)$ 
  2. IGNORE  $I(n)$ 
  3.  $S \leftarrow$  EXPAND  $I(n)$ 
    For each  $n_i$  in  $S$ 
      EXPLORE( $I(n_i)$ )
else, choose one of the following: //  $n$  is a leaf node
  1. SHOWRESULTS  $I(n)$ 
  2. IGNORE  $I(n)$ 

```

Fig. 6. TOPDOWN Navigation Model

In order to define a cost model, we focus on a simplification of the general navigation model, which we call TOPDOWN, where only EXPAND, SHOWRESULTS and IGNORE are the available operations, that is, the user follows a top-down only navigation starting from the root. TOPDOWN is common in practice. When the user encounters a leaf node in TOPDOWN, the only option is SHOWRESULTS. The TOPDOWN navigation model is formally presented in Fig. 6 and is a recursive procedure that is initially called on the root of the active tree.

TOPDOWN Cost Model. The cost model, which is inspired by a previous work [2], takes into consideration the number of concept nodes revealed by an EXPAND action, the number of EXPAND actions that the user performs and the number of citations displayed for a SHOWRESULTS action. In particular, the cost model assigns (i) cost of 1 to each newly revealed concept node that the user examines after an EXPAND action, (ii) cost of 1 to each EXPAND action the user executes, and (iii) cost of 1 to each citation displayed after a SHOWRESULTS action.

For example, in the navigation of Fig. 2 above, the cost for reaching the “Histones” concept and inspecting its attached citations is 42. That is, 4 EXPAND actions that reveal a total of 23 concept nodes, and a SHOWRESULTS action on the “Histones” concept that lists 15 citations. The user examines all concept nodes and all citations in order to select the ones of interest.

Since the exact sequence of actions of a user cannot be known *a priori*, we estimate the cost based on the following two probabilities:

- EXPLORE probability $P_E(I(n))$ is the probability that

the user is interested in the component subtree $I(n)$ and will hence explore it. The IGNORE probability is $1 - P_E(I(n))$.

- EXPAND probability $P_C(I(n))$ is the probability that the user executes an EXPAND action on component subtree $I(n)$ given that she has chosen to explore $I(n)$. The SHOWRESULTS probability for $I(n)$ is $1 - P_C(I(n))$.

In Section 4, we show how we estimate probabilities $P_E(I(n))$ and $P_C(I(n))$. The cost of exploring component subtree $I(n)$, rooted at node n , is:

$$cost(I(n)) = P_E^N(I(n)) \cdot \left((1 - P_C(I(n))) \cdot |L(I(n))| + P_C(I(n)) \cdot \left(B + |S| + \sum_{s \in S} cost(I_C(s)) \right) \right)$$

where $P_E^N(I(n))$ is the normalized $P_E(I(n))$, such that the sum of P_E^N 's of the component subtrees after an EdgeCut equals 1. P_E^N of the original tree is 1. The intuition for this normalization is that the probability that the user wants to explore a node n should not depend on the specific expansions sequence that revealed n .

The first operand of the addition inside the big parenthesis is the cost of executing SHOWRESULTS on n . The second operand is the cost of executing an EXPAND action on n . The constant B is the cost of executing the EXPAND action, and S is the set of concept nodes revealed by the action, or otherwise the roots of component subtrees returned by the EdgeCut operation. $I_C(s)$ is the updated I set of a node $s \in S$ after the EXPAND action on $I(n)$ has been performed.

Recall that $|L(I(n))|$ in the cost formula is the number of distinct citations attached to $I(n)$. Intuitively, creating a component subtree with large number of duplicates reduces the navigation cost if the SHOWRESULTS probability for that subtree is high. Moreover, the number of duplicates across component subtrees should be minimal; otherwise the user will pay the cost of inspecting a citation multiple times.

Finally, note that by changing B , the cost assigned to executing an EXPAND action we affect the number of revealed concepts after each EXPAND. In particular, increasing this cost leads to more concepts revealed for each EXPAND action. This cost can be thought of as a *cognitive measure* of a user's expectation of the system behavior as she navigates the query navigation tree. A small expand cost would decrease the number of concept nodes revealed during each EXPAND action, whereas the user can process more. It would also increase the number of EXPAND actions thus frustrating the user. In Section 8, we experiment with various values of B .

4 ESTIMATION OF NAVIGATION PROBABILITIES

We assume that each citation is *equally likely* to be of interest to the user. If more information about the “goodness” of the citations were available, our approach could be straightforwardly adapted using appropriate weighting

for $L(I(n))$.

Estimating EXPLORE Probability P_E . Since all citations in the query result are assumed to be of equal importance, concept n is of higher interest if $L(n)$ is large. On the other hand, a concept that is associated with a very large number of citations $L_T(n)$ of MEDLINE, independently of the query, is probably not discriminatory or important. The latter is inspired by the inverse document frequency measure in Information Retrieval. Hence, $P_E(n)$ for a node n is proportional to $|L(n)|/|L_T(n)|$. We normalize $P_E(n)$ by dividing by the sum of all P_E 's in the navigation tree T , that is:

$$P_E(n) = \left(\frac{\frac{|L(n)|}{|L_T(n)|}}{\sum_{n_i \in T} \frac{|L(n_i)|}{|L_T(n_i)|}} \right)$$

For a component tree $I(n)$ rooted at node n :

$$P_E(I(n)) = \sum_{n_i \in I(n)} P_E(n_i)$$

Given the above formula, $P_E(I(n)) = 1$ for the initial active tree. The above P_E formulas, together with the cost model in Section 3, largely determine the characteristics of the component subtrees BioNav creates during an EXPAND action. In particular, the upper component subtree typically groups together (i) concepts with low P_E and a large number of attached citations, and (ii) concepts with high P_E and a small number of attached citations. The first group is dismissed as uninteresting and the second could lead to a large number of concepts being revealed. Intuitively, the two groups of concepts average each other out according to the $P_E(I(n))$ formula. The lower component subtrees typically group concepts with P_E and number of attached citations in-between the two extremes in a way that minimizes the average navigation cost.

Estimating EXPAND Probability P_C . $P_C(I(n))$ is 0, if n is a leaf concept node or has a singleton $I(n)$ set, since there is no other choice for the user. For internal nodes in the active tree with a non-singleton $I(n)$ set that have a large $L(I(n))$, a typical user will want to further narrow down when faced with the prospect of seeing too many citations, that is, $P_C(I(n))$ is 1, if $L(I(n))$ is greater than an *upper threshold*. $P_C(I(n))$ is 0, if $L(I(n))$ is smaller than an *lower threshold*. Currently, BioNav operates with 50 and 10 being the upper and lower threshold respectively.

In the remaining cases, a user might want to narrow down the search of $I(n)$ by executing an EXPAND action, if the citations under n are widely distributed among the subconcepts in $I(n)$. An objective measure for such a wide distribution (disorder) is *information entropy*. If the entropy of the subtree $I(n)$ is high, then the user would benefit from an EXPAND action. Hence, $P_C(I(n))$ is defined as:

$$P_C(I(n)) = E(I(n)) = \frac{-\sum_{n_i \in I(n)} \frac{|L(n_i)|}{|L(I(n))|} \log \frac{|L(n_i)|}{|L(I(n))|}}{-\log \frac{1}{|I(n)|}}$$

The sum can become greater than 1 because of the existence of duplicates. Hence, we normalize the entropy of

$I(n)$ by dividing with the maximum entropy, where citations are uniformly distributed to all nodes in $I(n)$ and there are no duplicates.

P_C determines the impact of duplicates in a component subtree after a node expansion. If $P_C(I(n))$ is low, that is, the SHOWRESULTS probability is high, then the number of duplicates in $I(n)$ plays a bigger role in the way a component subtree is expanded.

5 COMPLEXITY RESULTS

To prove that the problem of selecting the optimal valid EdgeCut for a given tree is NP-hard, where "optimal" means minimize the user navigation cost according to the navigation model of Section 3, we prove that the problem is NP-complete for a simplified navigation model, which we refer to as TOPDOWN-EXHAUSTIVE and is a special case of the TOPDOWN model shown in Fig. 6.

In TOPDOWN-EXHAUSTIVE, BioNav performs an EXPAND action on the root of the initial active tree, and then the user selects randomly the root of one of the component subtrees created and performs a SHOWRESULTS action. The cost of TOPDOWN-EXHAUSTIVE navigation is the cost to read the root label of all component subtrees revealed by the EdgeCut plus the cost of SHOWRESULTS for the selected component subtree.

Intuition on the complexity of computing optimal valid EdgeCut: The "optimal" valid EdgeCut is the EdgeCut that will lead to the minimum expected navigation cost, that is, the minimum average cost. In order to minimize the expected cost of TOPDOWN-EXHAUSTIVE navigation, we need to minimize the cost of EXPAND and of SHOWRESULTS. The cost of EXPAND is simply the number k of component subtrees produced by the EdgeCut. The average cost of SHOWRESULTS over all component subtrees equals the sum of unique elements (citations) in every subtree over k . This sum would be $|L(T)|$ where T is the navigation tree if there were no duplicates among the subtrees. However, due to the existence of duplicates (the same citation can be annotated with multiple MeSH concepts) this sum depends on the EdgeCut. Hence, the duplicates are the reason that the problem is NP-complete for TOPDOWN-EXHAUSTIVE, because we need to maximize the number of duplicates within the created subtrees, and at the same time create a relatively small number of component subtrees. Note that even for a given k , the problem of selecting the best EdgeCut is NP-hard as we show in Theorem 1.

Theorem 1. *Finding the optimal valid EdgeCut in TOPDOWN-EXHAUSTIVE is NP-complete.*

Proof. The decision problem corresponding to the problem of computing the optimal EdgeCut is as follows:

TOPDOWN-EXHAUSTIVE Decision (TED) Problem: Given a navigation tree T , where each node n contains a list $L(n)$ of elements from universe U (U are all the citations in the query result), that is, $L(n) \subseteq U$, there exists an EdgeCut \mathcal{C} of T that creates k subtrees (including the upper subtree) with d duplicate elements within the created subtrees. That is, if S_1, \dots, S_k are the subtrees and each S_i

contains $b(S_i)$ duplicates, i.e., elements that appear somewhere in S_1, \dots, S_{i-1} (if an element appears 3 times, then it counts as 2 duplicates), then $\sum_{i=1..k} b(S_i) = d$.

Note that the cost of a TOPDOWN-EXHAUSTIVE navigation is computed as follows, if we solve the TED problem for every combination of k and d . If T has W unique results, then a subtree of the EdgeCut will have on average $(W + d)/k$ results. Hence the whole navigation cost is $k + (W + d)/k$, where k is the cost of reading the labels of the k subtrees.

TED is in NP since a solution can be verified in polynomial time. To prove that it is NP-complete, we will reduce the MAXIMUM EDGE SUBGRAPH (MES) problem, which is NP-complete **Error! Reference source not found.**, to TED.

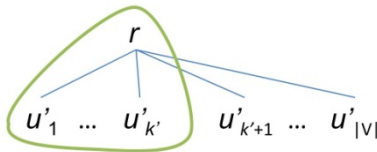
MAXIMUM EDGE SUBGRAPH (MES) Problem: Given graph $G(V, E)$, a weight function $w: E \rightarrow N$ (N are the natural numbers) and positive integers d and k' , there is a subset $V' \subseteq V$ with $|V'| = k'$ such that the sum of the edge weights of the edges between the nodes in V' is d , that is, $\sum_{(u,v) \in E \cap (V' \times V')} w(u, v) = d$.

Mapping of MES to TED: For each node $u \in V$, we create a node u' in T that is a child of the root of T . That is, the root r of T is empty ($L(r) = \emptyset$) and it has $|V|$ children.

The universe U is defined as follows: for each pair of edges $(u, v) \in E$ with weight $w(u, v)$, we add elements $B_{uv}^1, \dots, B_{uv}^{w(u,v)}$ in U .

Each of the nodes of T is populated with elements from U as follows: For each edge $(u, v) \in E$, we add to nodes u' and v' of T the elements $B_{uv}^1, \dots, B_{uv}^{w(u,v)}$. The intuition is that we map an edge weight in MES to the number of duplicates between two nodes in TED. We set $k = |V| - k' + 1$. In the figure below, the EdgeCut splits the tree into k subtrees.

Note that the above reduction is linear on the maximum edge weight in G , which generally is less than $|V|$, hence the reduction is polynomial on $|V|$ and $|E|$. Now, a solution to MES is mapped to a solution to TED, since selecting k' nodes in MES corresponds to expanding the tree into k subtrees in TED. The nodes of V corresponding to the nodes in the upper subtree of the EdgeCut (the one including the root) are the solution to MES. This set of nodes has maximum sum of edge weights in MES and maximum number of duplicates in TED.



Note: We assume that a node in TED can have the same element L multiple times. We could raise this assumption and just replace this node with a subtree that contains the element L multiple times. We did not do so to simplify the presentation of the proof. \square

6 ALGORITHMS FOR BEST EDGE CUT

Given the cost equation in Section 3, we can compute the

optimal cost by recursively enumerating all possible sequences of valid EdgeCuts, starting from the root and reaching every concept in the navigation tree, computing the cost for each step and taking the minimum. However, this algorithm is also prohibitively expensive. Instead we propose an alternative algorithm *Opt-EdgeCut* that makes use of the *dynamic programming* technique to reduce the computation cost. As shown in Section 6.1 below, *Opt-EdgeCut* is still exponential and is just used to evaluate the quality of the heuristic we present in Section 6.2 (*Heuristic-ReducedOpt*). In Section 6.3, we consider an alternate navigation strategy (*TopKLevelWise*), which in several variations is used in existing systems, such as eBay and Amazon, and allows users to navigate query results using extensive concept hierarchies. In *TopKLevelWise*, a fixed-size subset of children is revealed during each EXPAND action on a concept node, where the subset is selected based on a fixed cost metric. We compare two variations of *TopKLevelWise* with *Heuristic-ReducedOpt* in Section 6.2 and show that the navigation cost incurred using our approach can be an order of magnitude lower than either of these approaches.

Algorithm *Opt-EdgeCut*

Input: The navigation tree T

Output: The best EdgeCut

```

1 Traversing  $T$  in post-order, let  $n$  be the current node
2 while  $n \neq \text{root}$  do
3   if  $n$  is a leaf node then
4      $\text{mincost}(n, \emptyset) \leftarrow P_E(n) * L(n)$ 
5      $\text{optcut}(n, \emptyset) \leftarrow \{\emptyset\}$ 
6   else
7      $\mathbb{C}(n) \leftarrow$  enumerate all possible EdgeCuts
8     for the tree rooted at  $n$ 
9      $\mathbb{I}(n) \leftarrow$  enumerate all possible subtrees
10    for the tree rooted at  $n$ 
11    foreach  $I(n) \in \mathbb{I}(n)$  do
12      compute  $P_E(I(n))$  and  $P_C(I(n))$ 
13      foreach  $C \in \mathbb{C}(n)$  do
14        if  $C$  is a valid EdgeCut for  $I(n)$  then
15           $\text{cost}(I(n), C) \leftarrow$ 
16             $P_E(I(n)) \cdot \left( \frac{(1 - P_C(I(n))) \cdot L(I(n))}{+ P_C(I(n)) \cdot (B + |S| + \sum_{s \in S} \text{mincost}(I_C(s)))} \right)$ 
17          else
18             $\text{cost}(I(n), C) = \infty$ 
19           $\text{mincost}(n, I(n)) \leftarrow \min_{C_i \in \mathbb{C}(n)} \text{cost}(I(n), C_i)$ 
20           $\text{optcut}(n, I(n)) \leftarrow C_i$ 
21 return  $\text{optcut}(\text{root}, E)$  //  $E$  is the set of all tree edges
```

6.1 Optimal Algorithm for Best EdgeCut

The *Opt-EdgeCut* algorithm to compute the minimum expected navigation cost (and the EdgeCut that achieves it) traverses the navigation tree in post-order and computes the navigation cost bottom-up starting from the leaves. For each node n , the algorithm enumerates and stores the list $\mathbb{C}(n)$ of all possible EdgeCuts for the subtree rooted at n , and the list $\mathbb{I}(n)$ of all possible $I(n)$ sets that node n can be annotated with. The inclusion-exclusion principle [4] is used when enumerating $\mathbb{C}(n)$ and $\mathbb{I}(n)$, which leads to an ordering that maximizes reuse in the

dynamic programming algorithm. The algorithm then computes the minimum cost for each subtree in $\mathbb{I}(n)$ given the EdgeCuts in $\mathbb{C}(n)$ and the already computed minimum costs for the descendants of n . The complexity of *Opt-EdgeCut* is $O(|V| \cdot 2^{|E|})$.

Algorithm *Heuristic-ReducedOpt*

Input: Component subtree $I(n)$, number z of partitions

Output: The best EdgeCut

```

1   $z' \leftarrow z$ 
2  repeat
3     $k \leftarrow \sum_{n \in T} L(n) \cdot P_E(n) / z'$ 
4     $Partitions \leftarrow k$ -partition( $I(n), k$ )
      // call  $k$ -partition algorithm [14]
5     $z' \leftarrow z' - 1$ 
6  until  $|Partitions| \leq z$ 
7  construct reduced subtree  $I'(n)$  from  $Partitions$ 
8   $EdgeCut' \leftarrow Opt-EdgeCut(I'(n))$ 
9   $EdgeCut \leftarrow$  corresponding of  $EdgeCut'$  for  $I(n)$ 
10 return  $EdgeCut$ 

```

6.2 *Heuristic-ReducedOpt* Algorithm

The algorithm to compute the optimal navigation, *Opt-EdgeCut*, is exponential and hence infeasible for the navigation trees of most queries. We propose a heuristic to select a good EdgeCut for a node expansion. Note that the input argument to the heuristic is a component tree $I(n)$ and not the whole active tree T as in *Opt-EdgeCut*. The reason is that once *Opt-EdgeCut* is executed for T , the costs (and optimal EdgeCuts) for all possible $I(n)$'s are also computed and hence there is no need to call the algorithm again for subsequent expansions.

For a given component subtree $I(n)$, *Opt-EdgeCut* enumerates a large number of EdgeCuts on $I(n)$ and repeats this recursively on its subtrees. We propose to run *Opt-EdgeCut* on a reduced version $I'(n)$ of $I(n)$. The reduced tree $I'(n)$ has to be small enough so that *Opt-EdgeCut* can run on it in “real-time”. We select the size z of $I'(n)$ according to the processing power of our system. We set $z = 15$ in our experiments. Also, $I'(n)$ should approximate $I(n)$ as closely as possible. $I'(n)$ is the tree of “supernodes” created by partitioning $I(n)$. Each supernode in $I'(n)$ corresponds to a partition of tree $I(n)$. Then, *Opt-EdgeCut* is executed on $I'(n)$.

The algorithm we use to partition the tree is based on the k -partition algorithm [14] that processes the tree in a bottom-up fashion. For each tree node n , the algorithm removes the “heaviest” children of n one-by-one until the weight of n falls below k . For each of the removed children, it creates a partition. The result is a tree-partitioning with the minimum cardinality. The complexity of the k -partition algorithm is $O(|V| \cdot \log|V|)$.

We adopt the k -partition algorithm to our needs as follows. For each node in $I(n)$, we assign weight equal to $|L(n)| \cdot P_E(n)$, which is an estimation of its navigation cost. We run the k -partition algorithm by setting k , the weight threshold, to $\sum_{n_i \in I(n)} L(n_i) \cdot P_E(n_i) / z$, where z is the number of desired partitions. However, this might result in more than z partitions, due to some non-full partitions. Therefore we repeatedly run k -partition algorithm on $I(n)$, gradually increasing k (by decreasing z) until up

to z partitions are obtained. Note that z is the maximum tree size on which *Opt-EdgeCut* can operate in “real-time”.

6.3 The *TopKLevelWise* Method

In *TopKLevelWise*, the navigation model has the following key difference to our expansion model: the component subtree generated by an EXPAND on a node n are all rooted at one of the children of n . The size of the EdgeCut is limited by a parameter K , and the component subtree are chosen using a simple cost metric – the number of distinct results in a given component subtree. We consider two variations of *TopKLevelWise*. The first, which we call *static*, is employed by GoPubMed [28] and Amazon and uses $K = \text{infinity}$, that is, it selects the entire set of children to be included in the EdgeCut. The second, *Top10LevelWise*, is used by e-commerce websites such as eBay. Here, a set of $K = 10$ children, with the highest number of results, are displayed. We compare these two strategies to *Heuristic-ReducedOpt* in Section 8 and show that our approach outperforms both of them.

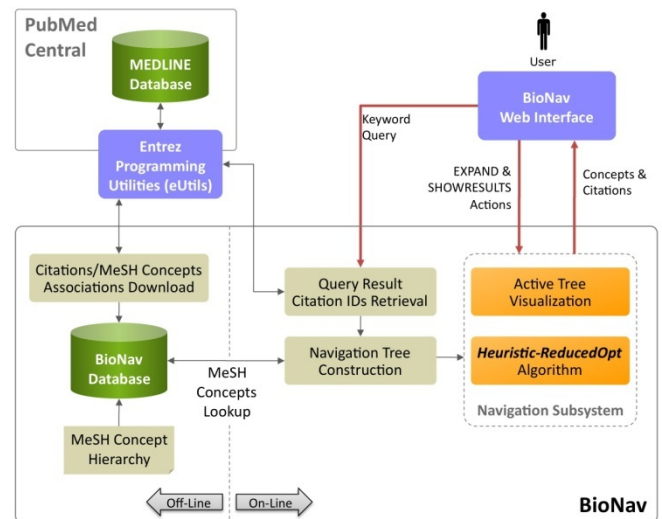


Fig. 7. BioNav System Architecture

7 SYSTEM ARCHITECTURE AND IMPLEMENTATION

The BioNav system architecture is shown in Fig. 7 and consists of two parts. The off-line components populate the BioNav database with the MeSH concept hierarchy and the associations of the MEDLINE citations with MeSH concepts, while the on-line components support BioNav’s web interface and the EXPAND-SHOWRESULTS actions of the user.

Off-Line Pre-Processing. The BioNav database is first populated with the MeSH hierarchy, which is available online [19] and has more than 48,000 concept nodes.

Then, the BioNav database is populated with the associations of the MEDLINE citations to MeSH concepts. These associations are not directly provided by the Entrez Programming Utilities (eUtils), so we had to implement the following method to infer these associations. For each concept in the MeSH hierarchy, we issued a query on PubMed using the concept as the keyword. For each citation ID in the query result, we added to a table in the

TABLE 1. QUERY WORKLOAD

#	Keyword(s)	# of Citations in Query Result	Navigation Tree Size	Max Tree Width/Height	Tree Citations w/ Duplicates	Target Concept	MeSH Level of Target Concept	L(n) of Target Concept	LT(n) of Target Concept
Biochemistry									
Q1	lbeta2	116	1947	1009/10	14927	Mice, Transgenic	5	11	90804
Q2	melibiose permease	160	1324	722/8	14419	Substrate Specificity	3	31	79470
Q3	Na+/I symporter	163	2596	1367/6	17146	Perchloric Acid	3	7	4250
Q4	ibogaine	287	3020	1656/11	28148	Serotonin	5	43	101567
Q5	prothymosin	313	3941	2113/10	30897	Histones	4	15	22741
Q6	ice nucleation	474	3181	1776/9	27440	Plants, Genetically Modified	3	2	12330
Q7	dyslexia genetics	517	3056	1691/9	45079	Polymorphism, Single Nucleotide	4	18	18843
Q8	syntaxin 1A	1115	6589	3764/10	105503	GABA Plasma Membrane Transport Proteins	7	11	650
Q9	folistatin	1183	6446	3656/10	102946	Follicle Stimulating Hormone	6	157	34540
Q10	norepinephrine transporter	1681	6482	3816/11	124199	Protein Kinase C	7	18	46928
Medicine									
Q11	varenicline	162	1830	962/6	11370	Nicotinic Agonists	7	81	18277
Q12	ildenafil	486	3424	2014/8	40987	Phosphodiesterase Inhibitors	5	401	69984
Q13	duloxetine	695	3884	2323/10	57979	Fibromyalgia	3	28	4683
Q14	ebola virus	1062	5187	2992/11	83602	Ebola Vaccines	5	25	27
Q15	asperger's syndrome	1126	3884	2323/9	57979	Early Diagnosis	2	28	4683
Q16	nocturia	1297	4646	2660/11	77083	Nocturnal Enuresis	5	39	1397
Q17	oxaluria	1727	5097	2913/10	85536	Celiac Disease	4	2	12871
Q18	blepharospasm	1329	5603	2145/9	72419	Blepharospasm	3	984	1313
Q19	cadmium poisoning	1882	6217	3628/11	79808	Infertility, Male	4	2	18839
Q20	tourette syndrome	3029	5196	1977/9	76835	Tourette Syndrome	5	36	2289

BioNav database the tuple $\langle \text{concept}, \text{citationID} \rangle$. Alternatively, we could determine the associations by using the MeSH concepts that each citation is annotated with in the MEDLINE database. This information is available through eUtils. In this case though, the navigation trees of BioNav would not be very informative, since each citation is annotated with 20 concepts on average in MEDLINE, while the PubMed indexing associates each citation with approximately 90 concepts on average (and include the 20 from MEDLINE.)

Given the number of concepts in the MeSH hierarchy, the number of citations in MEDLINE (~18 million), and the PubMed eUtils restrictions on the number of queries that can be executed within a certain period of time, it took almost 20 days to collect all the $\langle \text{concept}, \text{citationID} \rangle$ tuples. In the end, there were almost 747 million such tuples. To improve the selection queries on this table, we de-normalized it by concatenating all concepts associated with each citation into a comma-separated list, that is:

$\langle \text{citationID}, (\text{concept1}, \text{concept2}, \dots) \rangle$

In this work, we assume the dataset D to be fixed. However, in practice, D changes frequently as new citations are added and existing citations are updated to include new terms from the MeSH hierarchy. In this case, we assume that D is refreshed periodically by an offline process that issues queries to PubMed using the concept keyword and updates the concept counts and rows of retrieved citations. A newly added citation may not appear immediately in the query result, but we assume that such delays are acceptable to users.

When executing queries using concepts as keywords, we also store the number of citations $L_T(n)$ in the query result needed for the computation of P_E in Section 4.

On-Line Operation. Upon receiving a keyword query from the user, BioNav executes the same query against the MEDLINE database and retrieves only the IDs (PubMed Identifiers) of the citations in the query result. This is done using the ESearch utility of the Entrez Programming Utilities (eUtils) [7]. eUtils are a collection of web interfaces to PubMed for issuing a query and down-

loading the results with various levels of detail and in a variety of formats. Next, the navigation tree is constructed by retrieving the MeSH concepts associated with each citation in the query result from the BioNav database. This is possible since MeSH concepts have tree identifiers encoding their location in the MeSH hierarchy, which are also retrieved from the BioNav database. This process is done once for each user query. The navigation tree is trivially converted to an active tree (see Section 2) and passed on the *Navigation Subsystem* that supports the user's actions on the BioNav web interface.

Initially, the navigation subsystem just visualizes the active tree on the web interface, that is, it simply shows its root node. Subsequently, the user requests an EXPAND action on the root. Then, the navigation subsystem executes the *Heuristic-ReducedOpt* algorithm on the tree $I(r)$ of the root r , and the resulting active tree is visualized on the web interface. When the user makes a SHOWRESULTS request, BioNav uses the Entrez ESummary utility to download high level information of the citations to be shown, such title and authors.

8 EXPERIMENTAL EVALUATION

We evaluated the BioNav system in terms of both average navigation cost and expansion time performance. Other traditional measures of quality, such as precision and recall, are not applicable to our scenario as the objective is to minimize the tree navigation cost and not to classify.

In Section 8.1, we show that the BioNav navigation method, which is evaluated using the *Heuristic-ReducedOpt* algorithm, leads to considerably smaller navigation cost for a set of real queries on the MEDLINE database and navigations on the MeSH hierarchy. In Section 8.1, we compare the optimal algorithm (*Opt-EdgeCut*) with *Heuristic-ReducedOpt* and show that the heuristic is a good approximation of the optimal. These experiments were executed on a *reduced* navigation tree (~20 nodes), constructed from the original query navigation tree for each query, since *Opt-EdgeCut* is prohibitively expensive for most navigation trees. Finally, Section 8.3 shows that the execution time of *Heuristic-ReducedOpt* is small

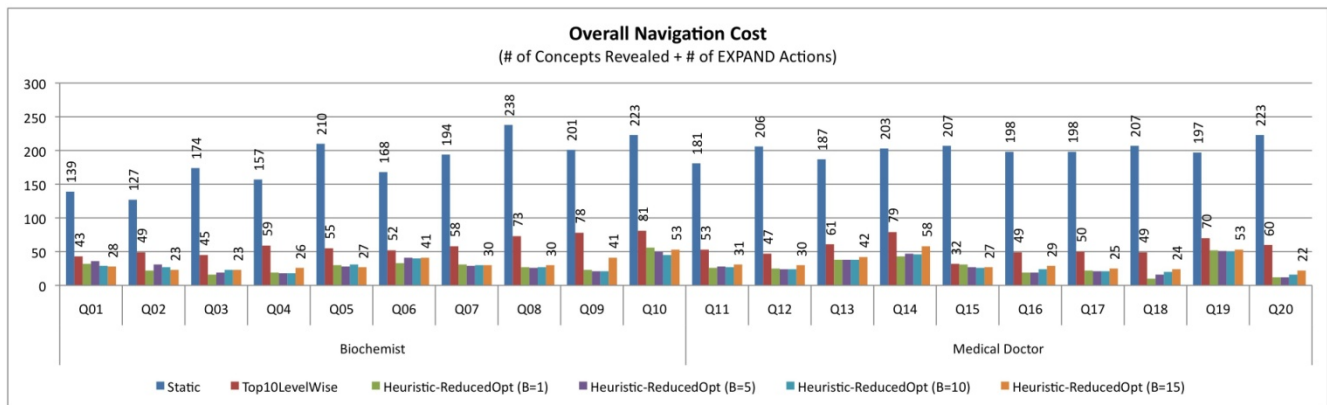


Fig. 8. Overall Navigation Cost Comparison for Biochemistry and Medicine

enough to facilitate interactive-time user navigation.

The experiments were executed on a Dell Optiplex machine with 3Ghz CPU and 2 GB of main memory, running Windows XP Professional. All algorithms were implemented in Java and Oracle 10g was used as the database.

8.1 Navigation Cost Evaluation

To evaluate the navigation cost benefit of BioNav, we asked two researchers, who use PubMed regularly, to create a set of 10 queries each. The first researcher was a biochemist and the second a medical doctor. We asked them to consider queries that cover topics within their fields and are of exploratory nature, that is, queries that return more than just a few citations. For each query, we also asked them to designate a *target* MeSH concept in the corresponding navigation tree that they would subjectively consider as most interesting. The two sets of queries we received consist our workload and is show in Table 1. Apart from the queries (“Keywords” column), listed are statistics on the initial navigation trees, the target concepts and information regarding their location depth in the MeSH hierarchy, the number of citations $|L(n)|$ attached to them for the given query, and the total number of citations $|L_T(n)|$ attached to them in MEDLINE.

“Follistatin” and “LbetaT2” are terms that mainly interest biochemists studying reproductive endocrinology and gynecology. The “dyslexia genetics” query accumulates results related to genes associated with dyslexia. “Melibiose permease” and “Na⁺/I⁻ symporter” are transport proteins related to bacterial growth and thyroid function respectively. On the other hand, “vardenafil” (Levitra), used for the treatment of erectile dysfunction, and “varenicline” (Chantix), used for quitting smoking, are two new drugs that interest many medical doctors.

Interestingly, some queries correlate with quite a few fields of research and others concentrate in more specific topics. For example, the literature for “prothymosin”, although not particularly broad in number of citations in the query result (313), is associated with several topics such as cancer, cell proliferation, apoptosis, chromatin remodeling, transcriptional regulation and immunity. In contrast, “vardenafil” retrieves a higher number of citations (486) but the literature is mostly targeted to erectile dysfunction and hypertension. This fact is reflected on the navigation tree characteristics for the two queries, also shown in Table 1. The navigation tree for “prothymosin”

is bigger than the one for “vardenafil” in every respect, that is, tree size, maximum width and height.

In this experiment we assume that the user follows a top-down navigation where she always chooses the right node to expand in order to finally reveal the target concept. We compare the navigation cost of BioNav, where EXPAND is implemented using the *Heuristic-ReducedOpt* algorithm (with $z = 10$), to the two navigation strategies, *Static* and *Top10LevelWise*, described in Section 6.3.

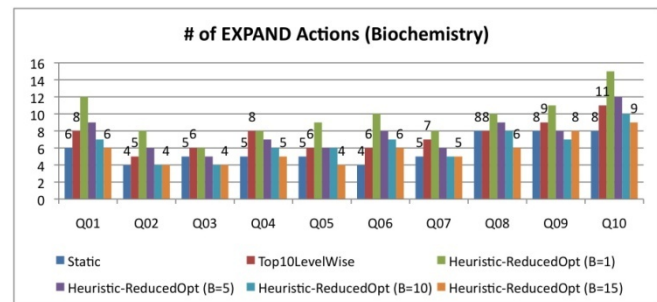


Fig. 9. Number of Expand Actions Comparison

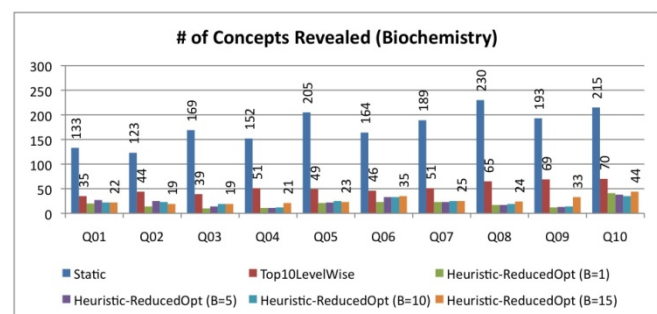


Fig. 10. Number of Concepts Revealed Comparison

Fig. 8 compares the navigation cost for these three methods. We observe that BioNav often improves the navigation cost by an order of magnitude, over Static navigation. The average improvement of BioNav, over static navigation, is 82%, for $B = 15$. The improvement is high regardless of the navigation tree characteristics (87% for “prothymosin” (Q5), 85% for “vardenafil” (Q12)), and regardless of the number of citations in the query result (80% for “LbetaT2” (Q1), 90% for “tourette syndrome” (Q20)). The smallest improvement (71%) was observed for

“ebola virus” (Q14). The reason is that its target concept (Ebola Vaccines) is located far away, in terms of navigation tree distance, from other query results. Most query results are distributed under a MeSH concept called “Viruses”, while the target concept is located under a sibling concept called “Complex Mixtures”. Hence, it takes several EXPAND actions until BioNav reveals the latter. Query “ice nucleation” (Q6) also exhibits small improvement (75%), but for a different reason. Its target concept (Plants, Genetically Modified) has an extremely low $|L(n)| = 2$. Hence, its P_E is quite low and so it takes several EXPAND actions until it is revealed.

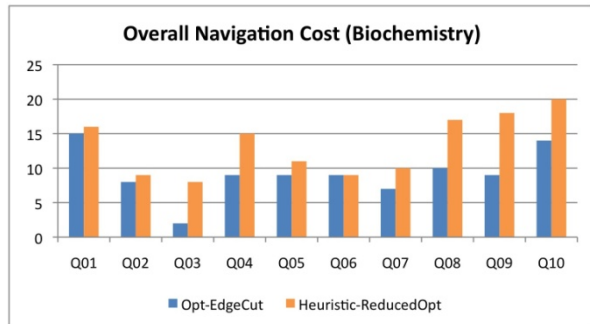


Fig. 11. Overall Navigation Cost Comparison

Consistent, but more modest, improvement in navigation cost is achieved by BioNav over *Top10LevelWise*. The average improvement is 41%, with a minimum of 16% for query “asperger’s syndrome” (Q15) and a maximum of 63% for “tourette syndrome” (Q20). Since *Top10LevelWise* explores the navigation tree level-wise, a concept that is high up in the hierarchy, such as the target concept of “asperger’s syndrome”, can be reached as fast by *Top10LevelWise* as it does by BioNav. On the other hand, a concept that is deep inside the navigation tree but with high P_E , such as the target concept of “tourette syndrome”, is reached much faster by BioNav.

Fig. 9 shows the number of EXPAND actions for the three methods for the biochemistry query set only. Note that these numbers are relatively close, which means that the dramatic differences in Fig. 8 are due to the fact that BioNav selectively reveals few descendant nodes for each EXPAND, instead of a possibly large number of child nodes. The worst case is the “ice nucleation” (Q6), where BioNav requires 6 EXPAND actions, compared to 4 of static navigation, since the target concept is quite high in the MeSH hierarchy, and at the same time has a low P_E , as discussed above. A similar increase in the number of EXPAND actions is observed for query “ebola virus” (not shown in Fig. 9) for the reason discussed above. Fig. 10 shows the number of revealed concepts for each method and demonstrates the superiority of our approach.

8.2 Opt-EdgeCut Comparison

To compare the optimal algorithm *Opt-EdgeCut* and *Heuristic-ReducedOpt*, we use the same query workload as in Section 8.1. As mentioned earlier, it is infeasible to execute *Opt-EdgeCut* on the navigation tree obtained for any query in Table 1. Therefore, we base our comparison on a

reduced navigation tree $I'(n)$ obtained by applying the procedure *GenReducedTree* to an initial navigation tree $I(n)$. The procedure *GenReducedTree* ensures that a reduced navigation tree has (1) at least one concept node with the same label as the target concept of the queries in Table 1, and (2) up to a maximum number $maxN$ of concept nodes. In this experiment, we set $maxN$ to 25.

Procedure *GenReducedTree*

Input: Initial Navigation Tree $I(n)$, the target concept c , and the desired number $maxN$ of nodes in the reduced tree

Output: A reduced tree with at most $maxN$ nodes, including c

```

1  collect all nodes of  $I(n)$  in list  $L$ 
2  create list  $L'$  to store the nodes of the reduced tree
3  add to  $L'$  a concept node in  $L$  with the same label as  $c$  and
   all its ancestors
4  while ( $sizeof(L') \leq maxN$ ) repeat
5     select a node  $c'$  uniformly at random from  $L$ 
6     add  $c'$  and all its ancestors to  $L'$ , excluding duplicates
7  create a tree  $I'(n)$  from the nodes in  $L'$ , preserving the
   parent-child relationship
8  return  $I'(n)$ 

```

Fig. 11 compares the proportional navigation cost of *Opt-EdgeCut* over *Heuristic-ReducedOpt* for the biochemistry query set only. *Opt-EdgeCut* performs better than *Heuristic-ReducedOpt* for all queries. However, the improvement varies over a wide range (6% for “LbetaT2” (Q1), to 75% for “Na+/I symporter” (Q3)). This is because partitioning in *Heuristic-ReducedOpt* hides away the target nodes inside one of the partitions during an EXPAND action, effectively excluding their participation in an EdgeCut. Thus, more EXPAND actions are needed to reach the target concept, which increases the cost. The opposite is true for query “ice nucleation” (Q6). The target concept is relatively high up in the hierarchy and the partition algorithm creates a partition for the target concept during the first expansion. Thus the same number of expansions is needed to reach it, resulting in the same overall cost.

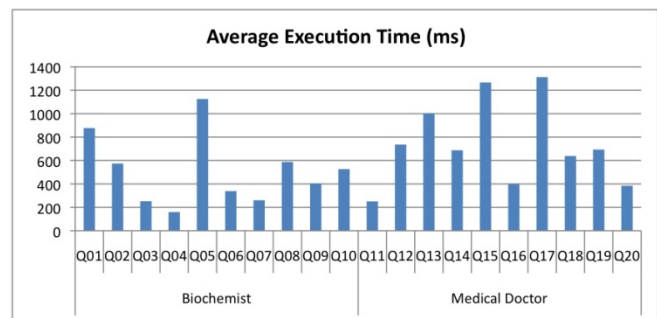


Fig. 12. Heuristic-ReducedOpt EXPAND Performance

8.3 Performance Evaluation

Fig. 12 shows the average time of *Heuristic-ReducedOpt* to execute an EXPAND action for each query in Table 1. The average was taken over the number of EXPAND actions partially shown in Fig. 9. For an input tree $I(n)$, *Heuristic-ReducedOpt* first creates a reduced tree $I'(n)$, and then runs the *Opt-EdgeCut* algorithm on it. The execution time is dominated by *Opt-EdgeCut* as it is an exponential algo-

rithm and depends on the size of the input tree. As stated earlier, we restrict the size of the reduced tree $I'(n)$ to 10 nodes and the EXPAND cost B is set to 15. However, $I'(n)$ can have a smaller size (see Section 6.2), in which case *Opt-EdgeCut* executes faster but with reduced accuracy.

For example, the reduced tree $I'(n)$ for “oxaluria” (Q17), in both EXPAND actions, had sizes 10 and 9 respectively, which explains the highest average execution time, and also among the highest improvements in Fig. 8. On the other hand, for “Na⁺/I symporter” (Q3), the first three EXPAND actions resulted in an $I'(n)$ of sizes 8, 8 and 7, respectively. Hence, the average execution time in Fig. 12 is lower, as is the improvement in navigation cost.

9 RELATED WORK

Several systems have been developed to facilitate keyword search on PubMed using the MeSH concept hierarchy. PubMed itself allows the user to search for citations based on MeSH annotations. A keyword query “histones[MeSH Terms]” will retrieve all citations annotated with the MeSH term “histones” in the MeSH hierarchy. The user can also limit her search to a MeSH term by using additional filters, e.g., “[majr]” to filter out all citations in the query result that don’t have the term as their major term. These filters can be combined by using the Boolean connectives AND, OR and NOT. This interface poses significant challenges, even to experienced users, since the annotation process is manual and thus prone to errors. The closest to BioNav is GoPubMed [5,28], which implements a static navigation method on the results of PubMed. GoPubMed lists a predefined list of high-level MeSH concepts, such as “Chemicals and Drugs”, “Biological Sciences” and so on, and for each one of them displays the top-10 concepts. After a node expansion, its children are revealed and ranked by the number of their attached citations, whereas BioNav reveals a selective and dynamic list of descendant (not always children) nodes ranked by their estimated relevance to the user’s query. Further, BioNav uses a cost model to decide which concepts to display at each step.

Other systems that tackle PubMed search using the MeSH concept hierarchy include PubMed PubReMiner [25] and XplorMed [22,30]. Both of them are query refinement tools and do not implement a particular navigation method. In particular, PubMed PubReMiner outputs a long list of all MeSH concepts associated with each query along with their citation count. The user can select one or more of them and refine her query. XplorMed performs statistical analysis of the words in the abstracts of the citations in the query result and proposes query refinements/extensions to the user in a multi-step process. Ali Baba [23] displays the results on a graph where edges denote associations between the result nodes, which are typically genes and proteins. iHOP [10,11] shows to the user the genes associated to a query gene, where the association is measured through co-occurrence in a sentence. LSLink [15] uses the physical links between objects in the query result to find *meaningful* associations between pairs of terms in different controlled vocabularies annotating

objects in multiple datasources. These associations allow users to discover novel and interesting relationships between pairs of concepts and potentially explore objects that are not retrieved by the initial query.

Two academic proposals [2,3] dynamically categorize SQL query results by inferring a hierarchy based on the characteristics of the result tuples. Their domain is the tuple attributes and their problem is how to organize them hierarchically in order to minimize the navigation cost. They also decide the value ranges for each attribute, for both categorical and numerical ones, and how to rank them. One of the systems [3] takes into consideration the user’s preferences during the inference for a more personalized experience. Once the hierarchy is inferred, they follow a static navigation method. BioNav is distinct since it offers dynamic navigation on a predefined hierarchy. Hence, BioNav is complementary to these systems, since it can be used to optimize the navigation, after these systems construct the initial navigation tree.

Clustering systems [27,29,31] create unsupervised query-dependent clusters. PubMatrix [24] takes as input two sets of keywords terms, in addition to query keyword, and generates a co-occurrence frequency matrix of each pair of terms from the two lists, in the query result. The user can then browse this matrix and perform independent searches on pairs of terms. The Clusty [29] search engine clusters keyword-based query results on the web and operates on top of other search engines. HighWire Press [27] uses Clusty’s algorithms to cluster query results in the biomedical domain. [6] clusters PubMed documents by the drug they refer to based on the UMLS [17] drugs classification. Once the clusters are created, a static navigation method is followed. BioNav could be adapted to work on top of the (typically shallow) hierarchy created by clustering systems.

10 CONCLUSION

Information overload is a common phenomenon encountered by users searching biomedical databases such as PubMed. We address this problem by organizing the query results according to their associations to concepts of the MeSH concept hierarchy and propose a dynamic navigation method on the resulting navigation tree. Each node expansion on the navigation tree, reveals a small set of nodes, selected from among its descendants, and the nodes are selected such that the information overload observed by the user is minimized. We formally stated the underlying framework and the navigation and cost models used for evaluation of our approach. We prove that the problem of selecting the set of nodes that minimize the navigation cost is NP-complete, we propose an efficient heuristic, and we validate it for diverse sets of queries and navigation trees. The implementation of BioNav is available at <http://db.cse.buffalo.edu/bionav>.

ACKNOWLEDGMENT

Vagelis Hristidis is partially supported by NSF grant IIS-0811922 and DHS grant 2009-ST-062-000016.

REFERENCES

- [1] J.S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis, "Automated Ranking of Database Query Results," *Proc. First Biennial Conf. Innovative Data Systems Research*, 2003.
- [2] K. Chakrabarti, S. Chaudhuri, and S.W. Hwang, "Automatic Categorization of Query Results," *Proc. ACM SIGMOD*, pp. 755-766, 2004.
- [3] Z. Chen and T. Li, "Addressing Diverse User Preferences in SQL-Query-Result Navigation," *Proc. ACM SIGMOD*, pp. 641-652, 2007.
- [4] L. Comtet, *Advanced Combinatorics: The Art of Finite and Infinite Expansions*, pp. 176-177, Reidel, 1974.
- [5] R. Delfs, A. Doms, A. Kozenkov, and M. Schroeder, "GoPubMed: Ontology-Based Literature Search Applied to Gene Ontology and PubMed," *Proc. German Conf. Bioinformatics*, pp. 169-178, 2004.
- [6] D. Demner-Fushman and J. Lin, "Answer Extraction, Semantic Clustering, and Extractive Summarization for Clinical Question Answering," *Proc. Int'l Conf. Computational Linguistics and Ann. Meeting of the Assoc. for Computational Linguistics*, pp. 841-848, 2006.
- [7] Entrez Programming Utilities, http://www.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html, 2008.
- [8] U. Feige, D. Peleg, and G. Kortsarz, "The Dense k-Subgraph Problem," *Algorithmica*, vol. 29, pp. 410-421, 2001.
- [9] V. Hristidis and Y. Papakonstantinou, "DISCOVER: Keyword Search in Relational Databases," *Proc. Int'l Conf. Very Large Databases (VLDB)*, 2002.
- [10] R. Hoffman and A. Valencia, "A Gene Network for Navigating the Literature," *Nature Genetics*, vol. 36, no. 7, p. 664, 2004.
- [11] iHOP – Information Hyperlinked over Protein, <http://www.ihop-net.org/UniPub/iHOP/>, 2008.
- [12] M. Kaki, "Findex : search results categories help where document ranking fails" *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems*, pp. 131-140, 2005.
- [13] A. Kashyap, V. Hristidis, M. Petropoulos, and S. Tavoulari, "BioNav: Effective Navigation on Query Results of Biomedical Databases," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, (short paper), pp. 1287-1290, 2009.
- [14] S. Kundu and J. Misra, "A Linear Tree Partitioning Algorithm," *SIAM J. Computing*, vol. 6, no. 1, pp. 151-154, 1977.
- [15] W. Lee, L. Raschid, H. Sayyadi, and P. Srinivasan, "Exploiting Ontology Structure and Patterns in Annotation to Mine Significant Associations between Pairs of Controlled Vocabulary Terms," *Proc. Fifth Int'l Workshop Data Integration in the Life Sciences (DILS)*, pp. 44-60, 2008.
- [16] J. Lin and W.J. Wilbur, "Pubmed Related Articles: A Probabilistic Topic Based Model for Content Similarity," *BMC Bioinformatics*, vol. 8, p. 423, 2007.
- [17] D. Lindberg, B. Humphreys, and A. McCray, "The Unified Medical Language System," *Methods of Information in Medicine*, vol. 32, no. 4, pp. 281-291, 1993.
- [18] D. Maglott, J. Ostell, K.D. Pruitt, and T. Tatusova, "Entrez Gene: Gene-Centered Information at NCBI," *Nucleic Acids Research*, vol. 33, pp. D54-D58, Jan. 2005.
- [19] Medical Subject Headings (MeSH), <http://www.nlm.nih.gov/mesh>, 2010.
- [20] J.A. Mitchell, A.R. Aronson, and J.G. Mork, "Gene Indexing: Characterization and Analysis of NLM's GeneRIFs," *Proc. AMIA Ann. Symp.*, pp. 460-464, Nov.
- [21] OMIM – Online Mendelian Inheritance in Man, <http://www.ncbi.nlm.nih.gov/Omim/>, 2008.
- [22] C. Perez-Iratxeta, P. Bork, and M.A. Andrade, "Exploring MEDLINE Abstracts with XplorMed," *Drugs of Today*, vol. 38, pp. 381-389, 2002.
- [23] C. Plake, T. Schiemann, M. Pankalla, J. Hakenberg, and U. Leser, "Ali Baba: PubMed as a Graph," *Bioinformatics*, vol. 22, no. 19, pp. 2444-2445, 2006.
- [24] PubMatrix: A Tool for Multiplex Literature Mining, <http://pubmatrix.grc.nia.nih.gov/>, 2003.
- [25] PubMed PubReMiner: A Tool for PubMed Query Building and Literature Mining, <http://bioinfo.amc.uva.nl/human-genetics/pubreminer/>, 2008.
- [26] H. Shatkay and R. Feldman, "Mining the Biomedical Literature in the Genomic Era: An Overview," *J. Computational Biology*, vol. 10, no. 6, pp. 821-855, 2003..
- [27] Stanford Univ. – HighWire Press, <http://highwire.stanford.edu> 2008.
- [28] Transinsight GmbH – GoPubMed, <http://gopubmed.org> 2008.
- [29] Vivisimo, Inc. – Clusty, <http://clusty.com/>, 2008
- [30] XplorMed: eXploring Medline abstracts, <http://www.ogic.ca/projects/xplormed/>, 2008.
- [31] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," *Proc. ACM SIGMOD*, pp. 103-114, 1996.



Abhijith Kashyap received his BE degree in Computer Engineering in 2002 from National Institute of Technology of Karnataka, India, and the MS degree in Computer Science in 2008 from SUNY Buffalo. He is a PhD student in the Computer Science and Engineering Department at SUNY Buffalo. His research interests include query interfaces, information discovery in heterogenous datasets, exploration of large databases, and preservation archives. He has worked at Oracle Corp. and at Yodlee Inc. (04-06), which develops personal finance applications.



Vagelis Hristidis received the BS degree in Electrical and Computer Engineering from the National Technical University of Athens and the MS and PhD degrees in Computer Science from the University of California, San Diego, in 2000 and 2004, respectively. Since then, he is an Assistant Professor in the School of Computing and Information Sciences at Florida International University, in Miami. His main research work addresses the problem of bridging the gap between databases and information retrieval. His research has been funded by the National Science Foundation, the Department of Homeland Security and the Kauffmann Foundation.



Michalis Petropoulos received his Diploma in Electronic and Computer Engineering in 1998 from the Technical University of Crete, Greece; and the MS and PhD degrees in Computer Science from the University of California, San Diego in 200 and 2005, respectively. He is an Assistant Professor in the Computer Science and Engineering Department at SUNY Buffalo since 2006. His research interests are in the intersection of the web and databases and include interfaces to databases, web data services, XML, and data integration. In 2000-01, he worked at Enosys Software Inc., which developed a commercial XML-based data integration platform.



Sotiria Tavoulari received her B.Sc. degree in Chemistry from the University of Ioannina, Greece in 1998, and the M.Sc. and Ph.D. degree in Biotechnology and Biochemistry in 2001 and 2005 respectively. She is a Post Doctoral Research Associate at Yale University. Her current research interests include structure-function relationships and post-transcriptional regulation of proteins responsible for neurotransmitter transport. Her post-doctoral research has been awarded with a Mentor Based Fellowship from "Autism Speaks".

▷ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib