BioNav: Effective Navigation on Query Results of Biomedical Databases

Abhijith Kashyap [1] Vagelis Hristidis [2]
Michalis Petropoulos [1] Sotiria Tavoulari[3]

[1]Dept. of Computer Science and Engineering
University at Buffalo, SUNY
[2]School of Computing and Information Sciences
Florida International University
[3]Department of Pharmacology
Yale University

September 8, 2008

MOTIVATION

- *Exploratory queries* are increasingly becoming a common phenomenon in life sciences
  - e.g., search for citations on a given keyword on PubMed
- These queries return too-many results, but only a small fraction is relevant
  - the user ends up examining all or most of the result tuples to find the interesting ones
- Can happen when the user is unsure about what is *relevant*
  - e.g., user is looking for articles on a broad topic: 'cancer'. . .
  - query returns over 2 million citations on PubMed

This phenomenon is commonly referred to as
'information-overload'

MOTIVATION

- *Exploratory queries* are increasingly becoming a common phenomenon in life sciences
  - e.g., search for citations on a given keyword on PubMed
- These queries return too-many results, but only a small fraction is relevant
  - the user ends up examining all or most of the result tuples to find the interesting ones
- Can happen when the user is unsure about what is *relevant*
  - e.g., user is looking for articles on a broad topic: 'cancer'. . .
  - query returns over 2 million citations on PubMed

This phenomenon is commonly referred to as
'information-overload'

MOTIVATION

- *Exploratory queries* are increasingly becoming a common phenomenon in life sciences
  - e.g., search for citations on a given keyword on PubMed
- These queries return too-many results, but only a small fraction is relevant
  - the user ends up examining all or most of the result tuples to find the interesting ones
- Can happen when the user is unsure about what is *relevant*
  - e.g., user is looking for articles on a broad topic: 'cancer'. . .
  - query returns over 2 million citations on PubMed

This phenomenon is commonly referred to as
'information-overload'

MOTIVATION

- *Exploratory queries* are increasingly becoming a common phenomenon in life sciences
  - e.g., search for citations on a given keyword on PubMed
- These queries return too-many results, but only a small fraction is relevant
  - the user ends up examining all or most of the result tuples to find the interesting ones
- Can happen when the user is unsure about what is *relevant*
  - e.g., user is looking for articles on a broad topic: 'cancer'. . .
  - query returns over 2 million citations on PubMed

This phenomenon is commonly referred to as
'information-overload'

Common approaches to avoid information-overload

- Ranking
- Categorization

Common approaches to avoid information-overload

- Ranking
- **Categorization**

Categorization in information systems

Assumptions:

- Tuples in the database are annotated with one or more categories or concepts
- The set of concepts are arranged in a concept hierarchy
  - **Example:** Each citation in PubMed is associated with several concepts from the MeSH (Medical Subject Headings) hierarchy, typically 12 to 20
- Users querying the database are familiar with the controlled vocabulary of the concept hierarchy

CATEGORIZATION IN INFORMATION SYSTEMS

Assumptions:

- Tuples in the database are annotated with one or more categories or concepts
- The set of concepts are arranged in a concept hierarchy
  - **Example:** Each citation in PubMed is associated with several concepts from the MeSH (Medical Subject Headings) hierarchy, typically 12 to 20
- Users querying the database are familiar with the controlled vocabulary of the concept hierarchy

Query Result Navigation: Naive Approach

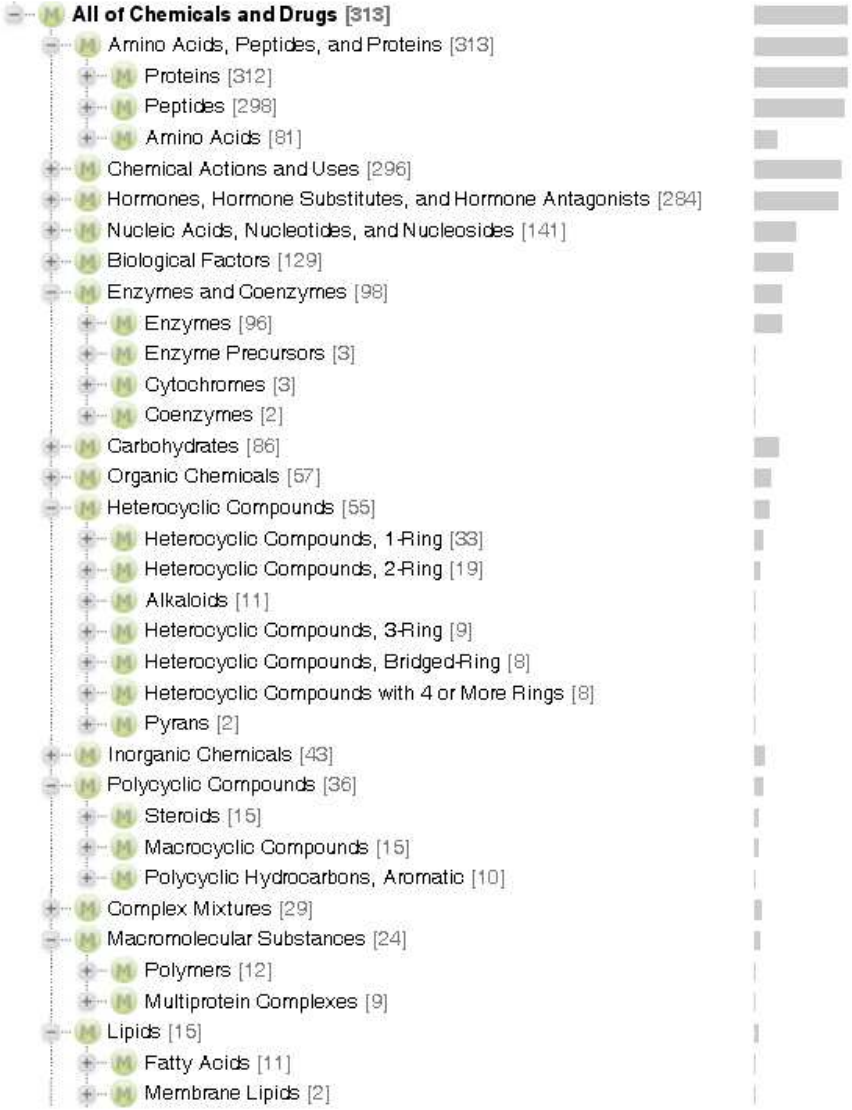
GoPubMed

Create the Navigation Tree as follows:

- Extract the set S of concepts annotating tuples in the query result set Q
- Construct the minimal sub-*concept hierarchy* tree T, that *covers* all concepts in S

Query Result Navigation: Naive Approach

GoPubMed



Example:
Section of Navigation Tree for query ’Prothymosin’ (313 results)

Query Result Navigation: Naive Approach

GoPubMed

Problems:

- Massive size of the Navigation Tree
  - MeSH has over 48000 concept nodes
  - 313 results span over 3000 of these concepts
- Large number of duplicate tuples
  - Each tuple is annotated with 12-20 MeSH concepts
  - Total tuple count is over 5000

Effort required to navigate the query results **increases!**

Query Result Navigation: Naive Approach

GoPubMed

Problems:

- Massive size of the Navigation Tree
  - MeSH has over 48000 concept nodes
  - 313 results span over 3000 of these concepts
- Large number of duplicate tuples
  - Each tuple is annotated with 12-20 MeSH concepts
  - Total tuple count is over 5000

Effort required to navigate the query results **increases!**

Query Result Navigation: Naive Approach

GoPubMed

Problems:

- Massive size of the Navigation Tree
  - MeSH has over 48000 concept nodes
  - 313 results span over 3000 of these concepts
- Large number of duplicate tuples
  - Each tuple is annotated with 12-20 MeSH concepts
  - Total tuple count is over 5000

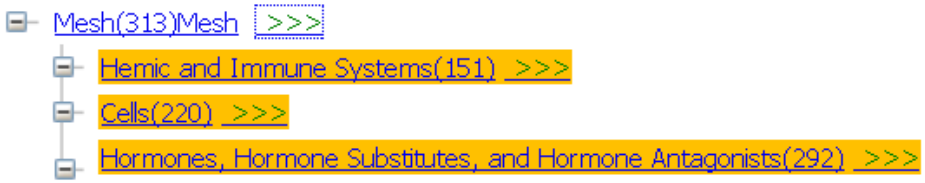
Effort required to navigate the query results **increases!**

Query Result Navigation: Dynamic Approach

BioNav

Example: Navigation steps for query 'Prothymosin'

Only a selective set of descendents is shown

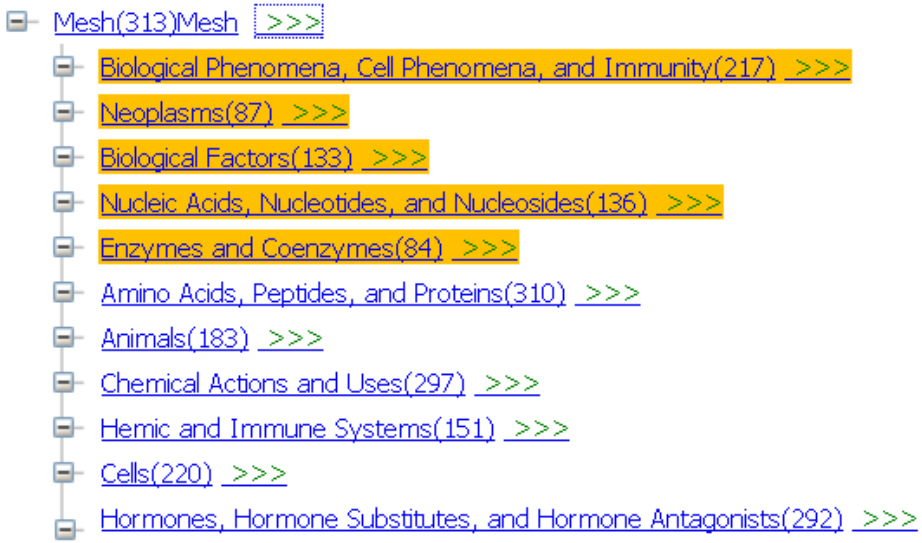


Query Result Navigation: Dynamic Approach

BioNav

Example: Navigation steps for query 'Prothymosin'

An **expand action** >>> on the root reveals next relevant set of descendants



Query Result Navigation: Dynamic Approach

BioNav

Example: Navigation steps for query 'Prothymosin'

User can choose to expand an internal node, to see nodes from the sub-tree rooted at the node

Query Result Navigation: Dynamic Approach

BioNav

BioNav Idea: At each navigation step,

- for a given node, instead of showing all children, reveal a selective set of descendants
- descendents are chosen so that the overall navigation cost is minimized, using a formal cost model

CONTRIBUTIONS

- Comprehensive framework for navigating large query results using extensive concept hierarchies
- A formal cost model for measuring the navigation cost incurred by the user
- Algorithms and heuristics for minimizing the expected navigation cost
- Experimental evaluation and system demo: http://db.cse.buffalo.edu/bionav/

Framework
Definitions

1. A **Concept Hierarchy** $H(V, E, r)$ is labeled tree of:
   - A set V of concept nodes
   - A set E of parent/child edges
   - A root r

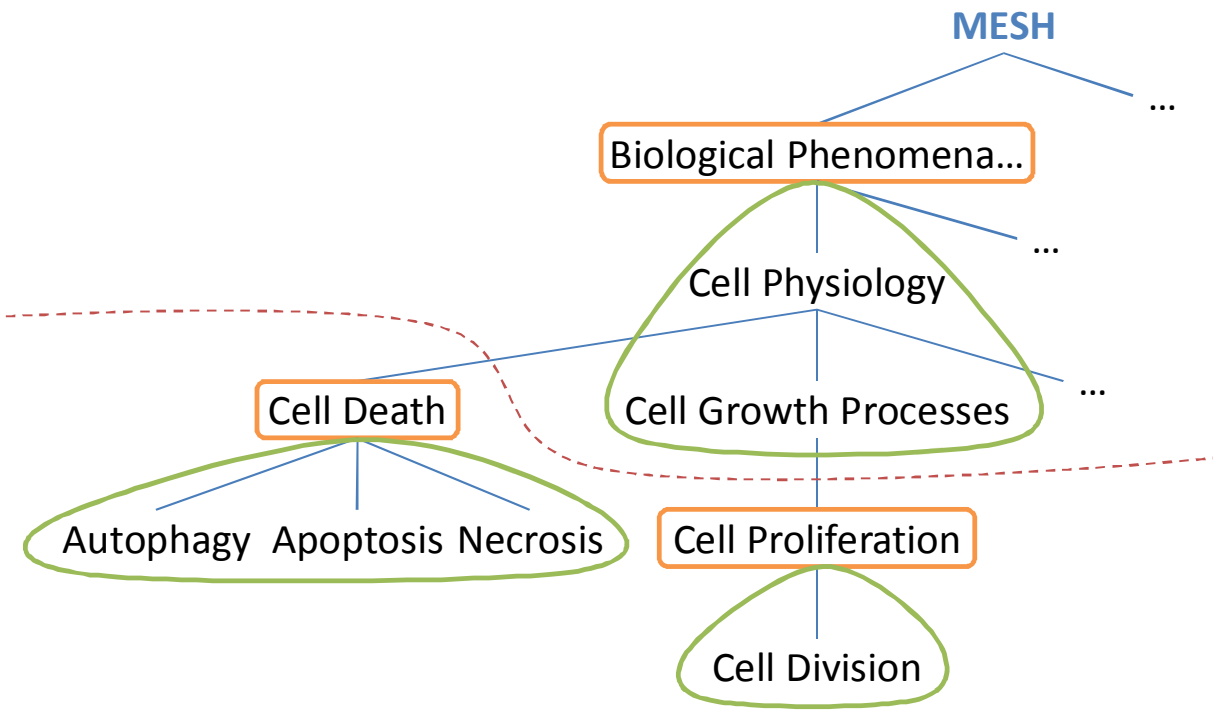
   According to the semantics of the MeSH concept hierarchy, a child is more specific than the parent
2. A **Navigation Tree** $T(V, E, r)$ is
   - created as a response to the user query
   - by attaching to each node of (MeSH) concept hierarchy, a list of its associated citations
   - and removing all nodes with no attached citations (while preserving parent/child relationship)

Framework
Definitions

- User navigates the Navigation Tree by a series of 'expand' actions on concept nodes
- Each expand action generates an EdgeCut on the residual navigation tree rooted at the given node

FRAMEWORK

EXAMPLE: (NAVIGATION TREE, EDGECUT AND COMPONENT SUBTREES)



A valid EdgeCut divides the tree
into a number of Component Subtrees

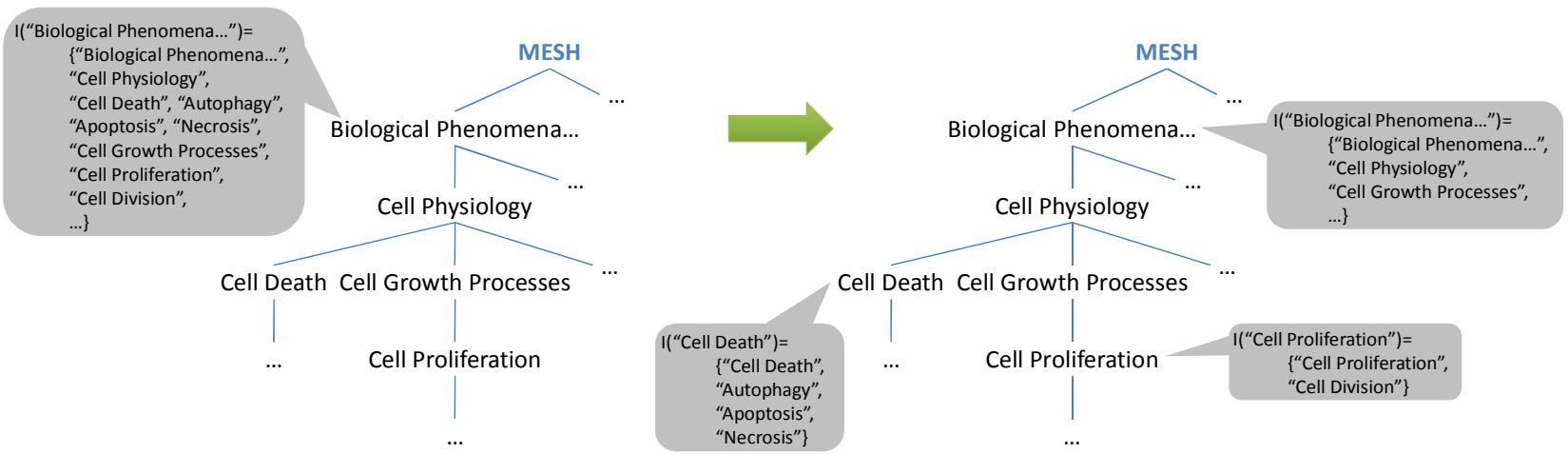
FRAMEWORK
DEFINITIONS (CONTD):

Not all EdgeCuts are valid, the 'expand' action generates only valid EdgeCuts

3. **Valid EdgeCut:** An EdgeCut C is valid is no two edges in C appear in a path from the root to a leaf node

FRAMEWORK
DEFINITIONS (CONTD):

4. An **Active Tree** $T_A(V, E, r)$ is a Navigation Tree where each node $n \in V$ is annotated with the nodeset consisting of nodes in the *component subtree* rooted at n



Visualization of the Active Tree as presented to the user

Navigation Model

TOP-DOWN User explores the Active Tree until she finds *every* relevant tuple in the query result

- In response to a query, BioNav presents the initial active tree to the user
- The user navigates the tree by performing one of the following actions:
  - **EXPAND**
  - **SHOWRESULTS**
  - **IGNORE**

Navigation Model

Model of exploration of node C in TOP-DOWN scenario:

Algorithm 1 Explore C

1: **if** n is not a leaf node, then choose one of the following **then**
2: SHOWRESULTS(n)
3: IGNORE(n)
4: $S \leftarrow$ EXPAND(n)
5: **for** each $n_i \in S$ **do**
6: EXPLORE(n_i)
7: **end for**
8: **else**
9: CHOOSE one of the following:
10: a) Examine all tuples in (C)
11: b) IGNORE C
12: **end if**

Cost Model

- Define *cost* as the total number of items, both tuples and concept labels, examined by the user
- Minimizing the *cost* also minimizes the information-overload a user encounters
- The choices for a *given* user for a given query is not known *apriori*
  - but structure of the active tree and the distribution of results on the tree are known
- Use this knowledge to estimate the *cost* for the *average* case

Cost Model

- Define *cost* as the total number of items, both tuples and concept labels, examined by the user
- Minimizing the *cost* also minimizes the information-overload a user encounters
- The choices for a *given* user for a given query is not known *apriori*
  - but structure of the active tree and the distribution of results on the tree are known
- Use this knowledge to estimate the *cost* for the *average* case

Cost Model

- Define *cost* as the total number of items, both tuples and concept labels, examined by the user
- Minimizing the *cost* also minimizes the information-overload a user encounters
- The choices for a *given* user for a given query is not known *apriori*
  - but structure of the active tree and the distribution of results on the tree are known
- Use this knowledge to estimate the *cost* for the *average* case

Cost Model

- Define *cost* as the total number of items, both tuples and concept labels, examined by the user
- Minimizing the *cost* also minimizes the information-overload a user encounters
- The choices for a *given* user for a given query is not known *apriori*
  - but structure of the active tree and the distribution of results on the tree are known
- Use this knowledge to estimate the *cost* for the *average* case

Cost Model

- Define *cost* as the total number of items, both tuples and concept labels, examined by the user
- Minimizing the *cost* also minimizes the information-overload a user encounters
- The choices for a *given* user for a given query is not known *apriori*
  - but structure of the active tree and the distribution of results on the tree are known
- Use this knowledge to estimate the *cost* for the *average* case

Cost Model
Intuition

Aim: Minimize the overall navigation cost

- There is a trade-off between the number of navigation actions (expand actions and viewing labels) and viewing results. Factors affecting cost:
  - Showing a large number of results up-front increases cost
  - A large number of navigation actions also increase the cost
  - The active tree has a large number of duplicates, which add to cost

Assumption:

- Tuples in the query results are not ranked every tuple is assumed have equal relevance

Cost Model
Probabilities

- The user choices in navigation model are non-deterministic and not equally likely
- However, a cost estimate is needed (to compute optimal navigation path) even before the user starts navigation

Cost Model
Probabilities

- The user choices in navigation model are non-deterministic and not equally likely
- However, a cost estimate is needed (to compute optimal navigation path) even before the user starts navigation

Cost Model
Probabilities

To estimate the cost, we introduce two probabilities to capture the user's intensions:

- **EXPLORE Probability** $P_E(T)$: Probability that the user is interested in the component sub-tree and hence will explore it
  - $1 - P_E(T)$ is the probability that the user would ignore
- **EXPAND Probability** $P_C(T)$: Probability that the user executes a EXPAND component sub-tree and hence will explore it
  - $1 - P_C(T)$ is the probability that the user would choose to see all the tuples of T

Cost Model
Probabilities

To estimate the cost, we introduce two probabilities to capture the user's intensions:

- **EXPLORE Probability** $P_E(T)$: Probability that the user is interested in the component sub-tree and hence will explore it
  - $1 - P_E(T)$ is the probability that the user would ignore
- **EXPAND Probability** $P_C(T)$: Probability that the user executes a EXPAND component sub-tree and hence will explore it
  - $1 - P_C(T)$ is the probability that the user would choose to see all the tuples of T

Cost Model
Probabilities

To estimate the cost, we introduce two probabilities to capture the user's intensions:

- **EXPLORE Probability** $P_E(T)$: Probability that the user is interested in the component sub-tree and hence will explore it
  - $1 - P_E(T)$ is the probability that the user would ignore
- **EXPAND Probability** $P_C(T)$: Probability that the user executes a EXPAND component sub-tree and hence will explore it
  - $1 - P_C(T)$ is the probability that the user would choose to see all the tuples of T

Cost Model

Cost Formula

If the user explores a concept node n, she has two choices:

- SHOWRESULT(n): with cost $(1 - P_C(n)) \times numRes(n)$
- EXPAND(n): cost has 2 components
  - Expand action. Cost : 1
  - Viewing the revealed labels $|S|$
  - EXPLOREing the component-subtrees $\sum_{s \in S} cost(s)$

Total cost of exploring a node n is:

$cost_{EXPLORE}(n) =$

$$(1 - P_C(n)) \times numRes(n) + P_C(n) \left(1 + |S| + \sum_{s \in S} cost(s)\right)$$

- $numRes(n)$ is the number of distinct tuples in the component subtree rooted at n
- S the set of component trees generated by an EdgeCut

Cost Model

Cost Formula (contd)

- For a given node n, a user can either EXPLORE or IGNORE a node
- Ignored nodes do not add to cost, that is, $cost_{IGNORE}(n) = 0$

$$cost_{TOTAL} =$$

$$((1 - P_E(n)) \times cost_{IGNORE}(n)) + (P_E(n) \times cost_{EXPLORE}(n)) =$$

$$P_E(n) \times \left((1 - P_C(n)) \times numRes(n) + P_C(n) \left(1 + |S| + \sum_{s \in S} cost(s) \right) \right)$$

Cost Model

Estimating EXPLORE Probability P_E

- A concept node has higher EXPLORE probability P_E if it has a large number of tuples attached to it
- unless, the concept is *too* generic and non-discriminatory and appears in a large number of tuples in the database, e.g., 'cancer' or 'water'

Therefore:

- P_E is proportional to the number of tuples attached to a node, for the given query
- inversely proportional to the total number of tuples associated with the concept in the database

Cost Model

Estimating EXPLORE Probability P_E

$$P_{EXPLORE}(n) \propto \left(\frac{numRes_{query}(n)}{numRes_{db}(n)}\right)$$

Normalized over all nodes in the active tree:

$$P_{EXPLORE}(n) = \left(\frac{numRes_{query}(n)}{numRes_{db}(n)}\right) / \sum_{n_i \in N_tree} \left(\frac{numRes_{query}(n_i)}{numRes_{db}(n_i)}\right)$$

Cost Model

Estimating EXPAND Probability P_C

Intuition:

- Expanding a component-subtree with a 'large' number of tuples decreases cost
- *whereas*, for sub-trees with 'small' number of nodes, expanding increases cost
- For subtrees with moderate number of tuples:
  - If the results are widely distributed, expanding may reduce cost by narrowing down the nodes in the sub-tree

Cost Model

Estimating EXPAND Probability P_C

- If $numRes(n_{T_n}) > thres_{upper}$ we set P_C to 1, that is, always favor EXPAND
- If $numRes(n_{T_n}) < thres_{lower}$ we set P_C to 0, that is, always favor SHOWRESULTS
- For the remaining cases, we use (normalized) $entropy(n_{T_n})$ to estimate P_C

$$entropy(n_{T_n}) = \frac{\sum_{n \in T_n} \frac{numRes(n)}{numRes(T_n)} \log \frac{numRes(n)}{numRes(T_n)}}{-\log \frac{1}{numRes(T_n)}}$$

- $numRes(n)$ is the number of results in node n
- $numRes(T_n)$ is the number of *distinct* results in sub-tree T

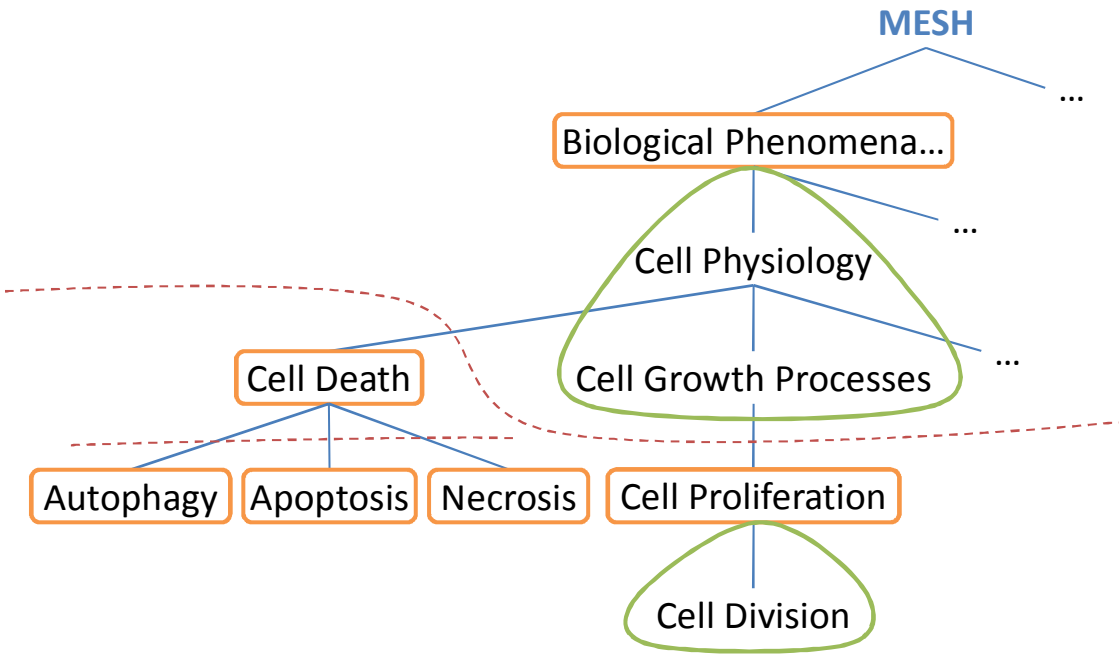
Algorithms for EdgeCut

Naive

- Enumerate all possible sequence of EdgeCuts over the initial active tree
- Compute the cost as given by the cost formula, and take the minimum
- Complexity: $O(2^{2|E|})$

Algorithms for EdgeCut

Naive



Example: Section of Active Tree with two subsequent cuts and the corresponding component sub-trees

Algorithms for EdgeCut

Optimal

- Enumerate only *valid* EdgeCuts
- Use dynamic programming to reduce computation cost

- Complexity: $O(|V| \times 2^{|E|})$

Still too slow to be used as real-time algorithm!

Algorithms for EdgeCut

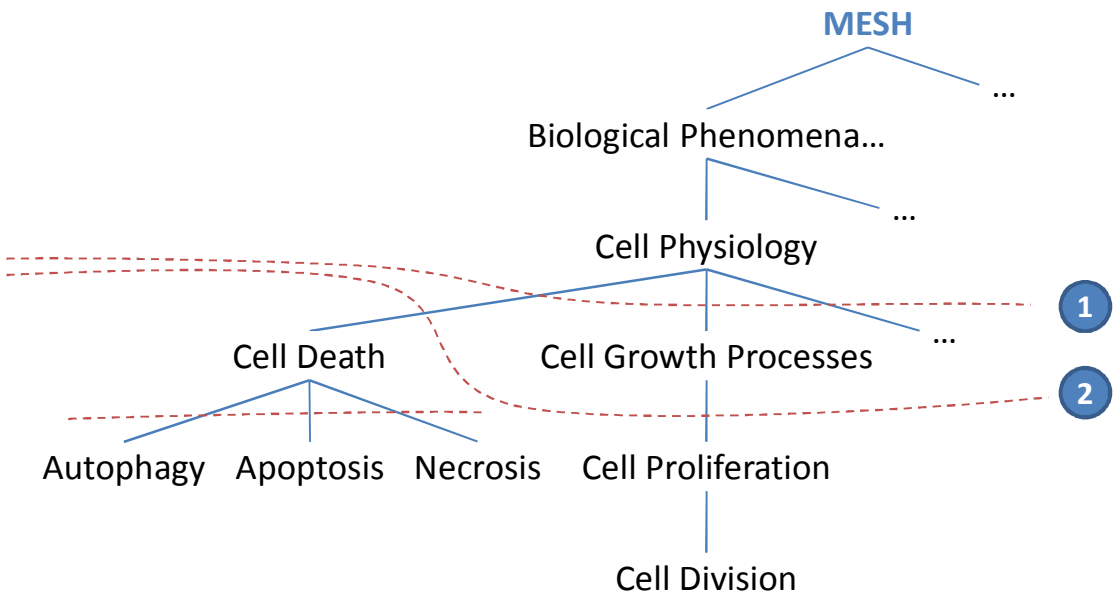
Optimal

- Enumerate only *valid* EdgeCuts
- Use dynamic programming to reduce computation cost
- Complexity: $O(|V| \times 2^{|E|})$

Still too slow to be used as real-time algorithm!

Algorithms for EdgeCut

Optimal



Example: Section of Active Tree with two possible cuts

Algorithms for EdgeCut

Heuristic

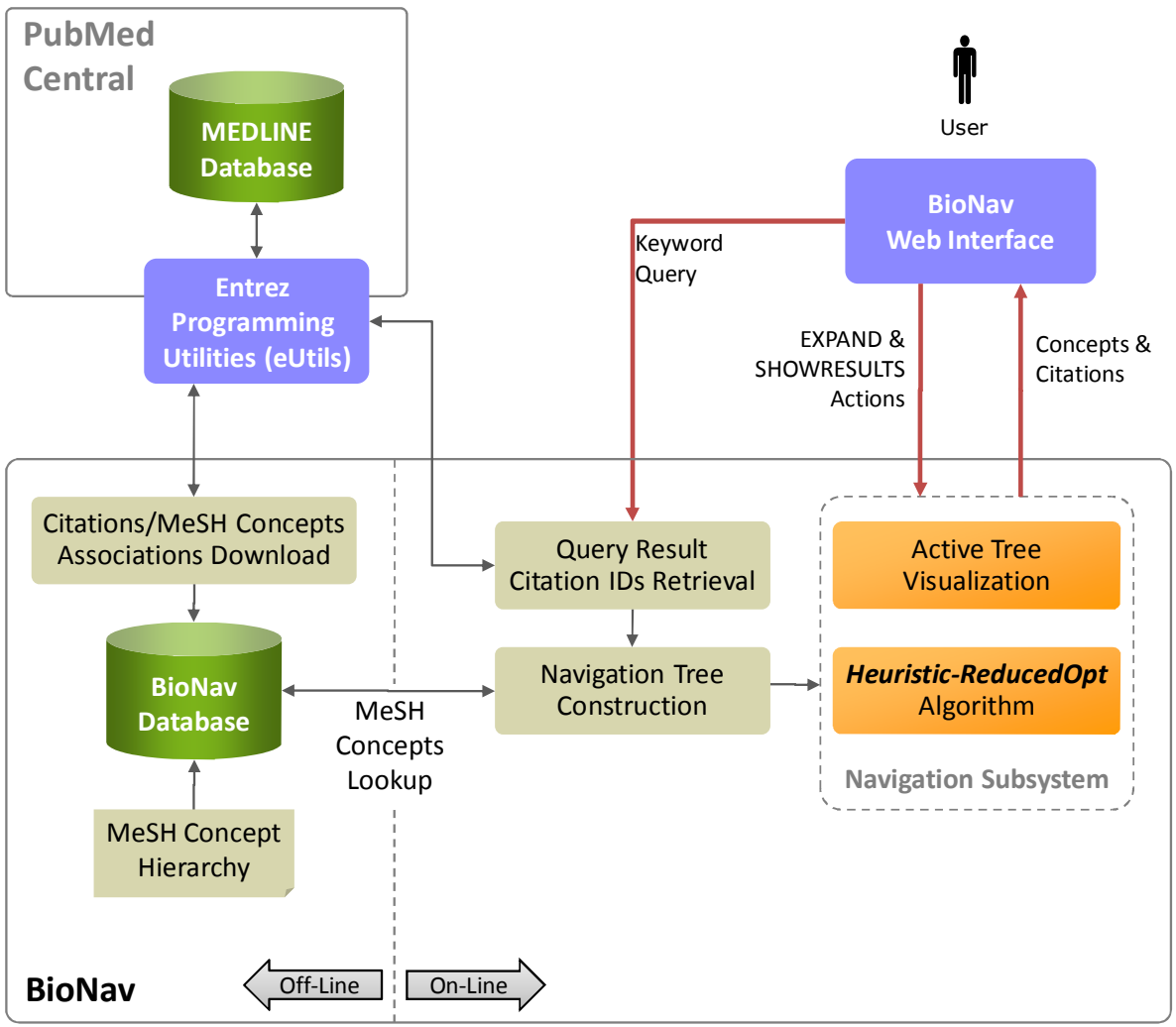
- Idea: Reduce the size of the active tree and run the optimal algorithm
- Ensure: The reduced tree 'approximates' the active tree as much as possible

Method:

- We use the equi-partitioning algorithm proposed by [Misra77] to partition the *active tree*
- Partitions are created such that each partitioned sub-tree has approximately same SHOWRESULTS cost, that is, same number of results
- A representing node is created for each partition and added to the reduced tree
- while maintaining the parent/child relationship

[Misra77] J. Misra and M. Kundu: A linear tree partitioning algorithm, SIAM 77

ARCHITECTURE



Experimental Evaluation

Setup

Experiments to evaluate:

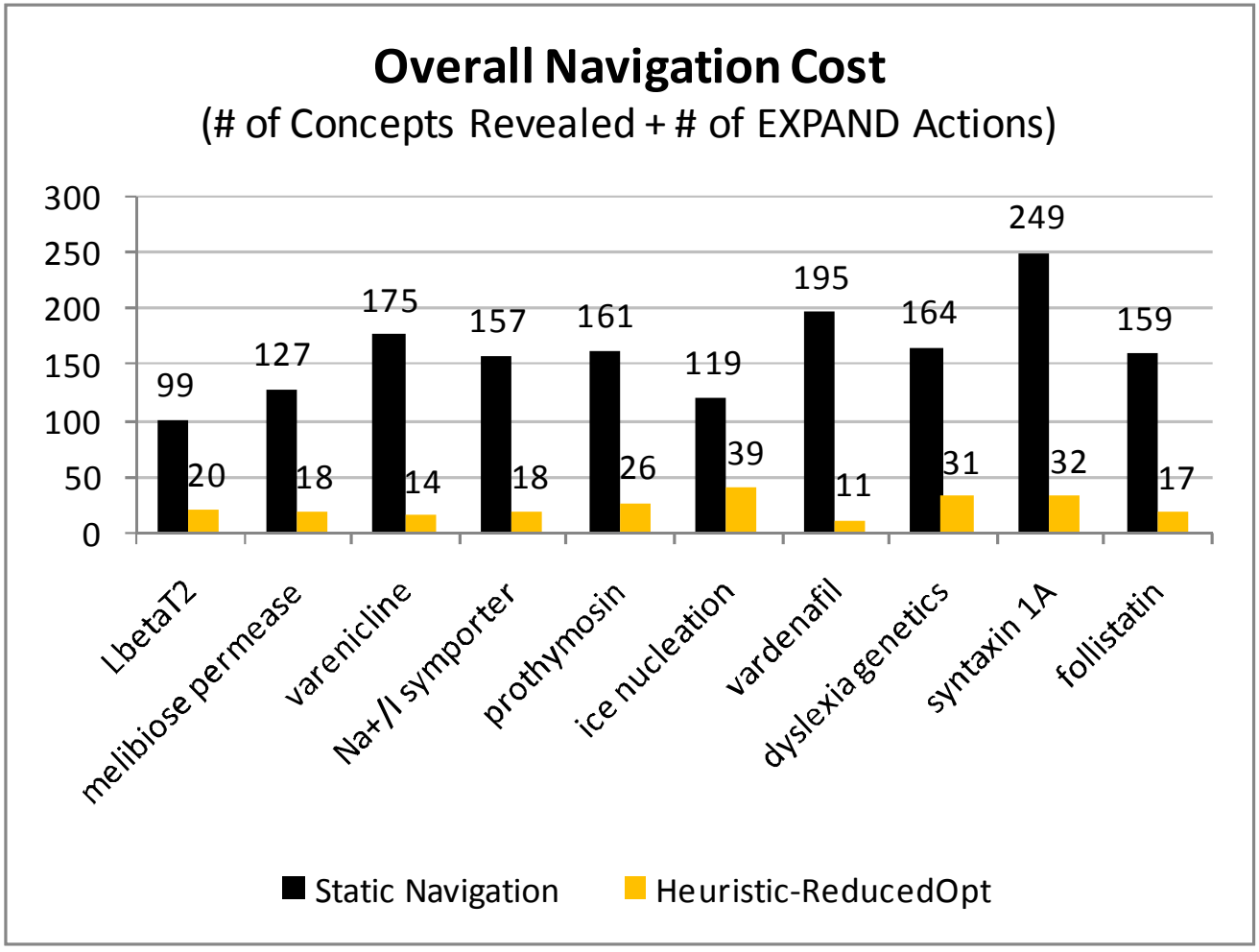
- Effect on navigation cost
- Performance of the system

Setup:

- Total of ten queries considered for evaluation
- Queries and target concepts were sourced from expert-users from the biomedical domain
- Cover a range of use-cases including:
  - Queries with highly specific keywords with a relevant specific concept
  - Non-specific queries with a relevant specific concept

Experimental Evaluation

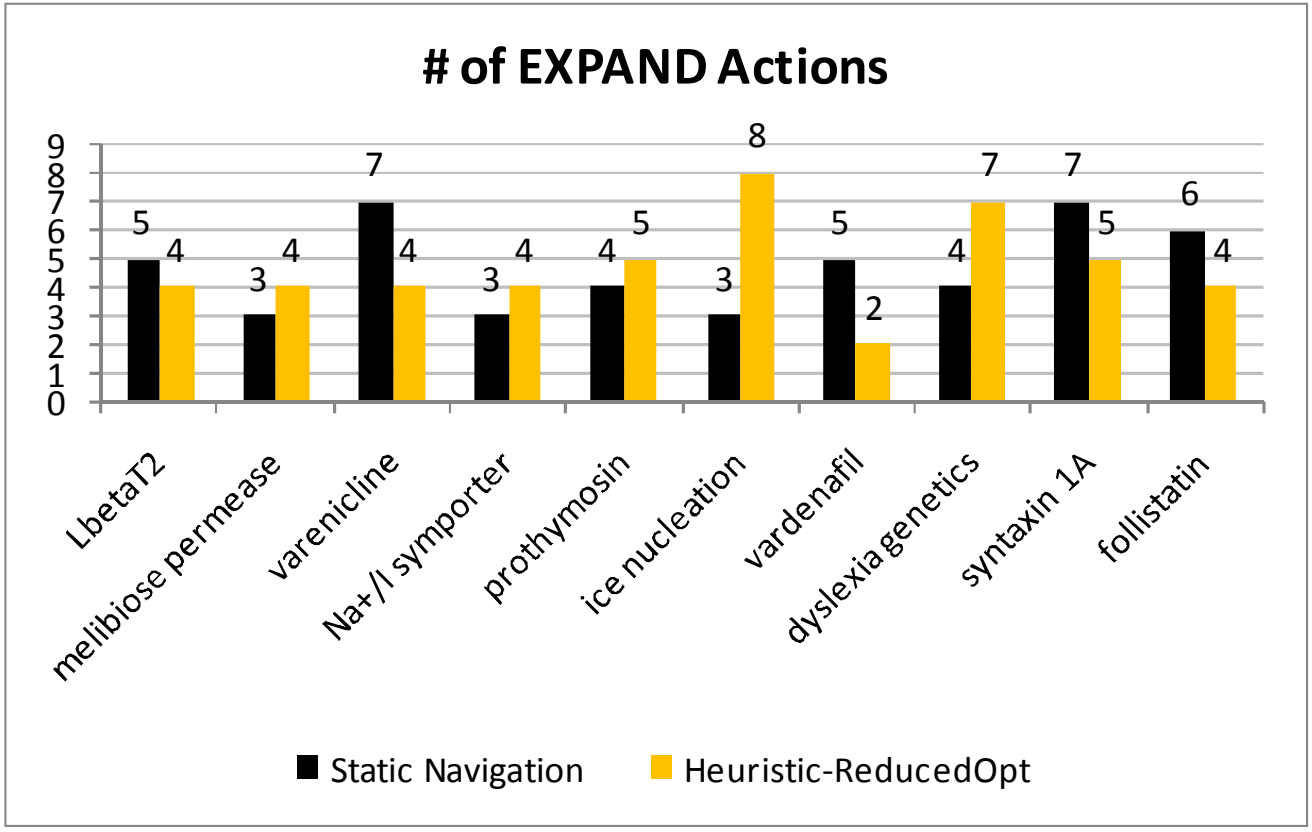
Improvement in Overall Navigation Cost



Navigation Cost of BioNav vs. Static Navigation

Experimental Evaluation

Improvement in navigation actions



Expand actions of BioNav vs. Static Navigation

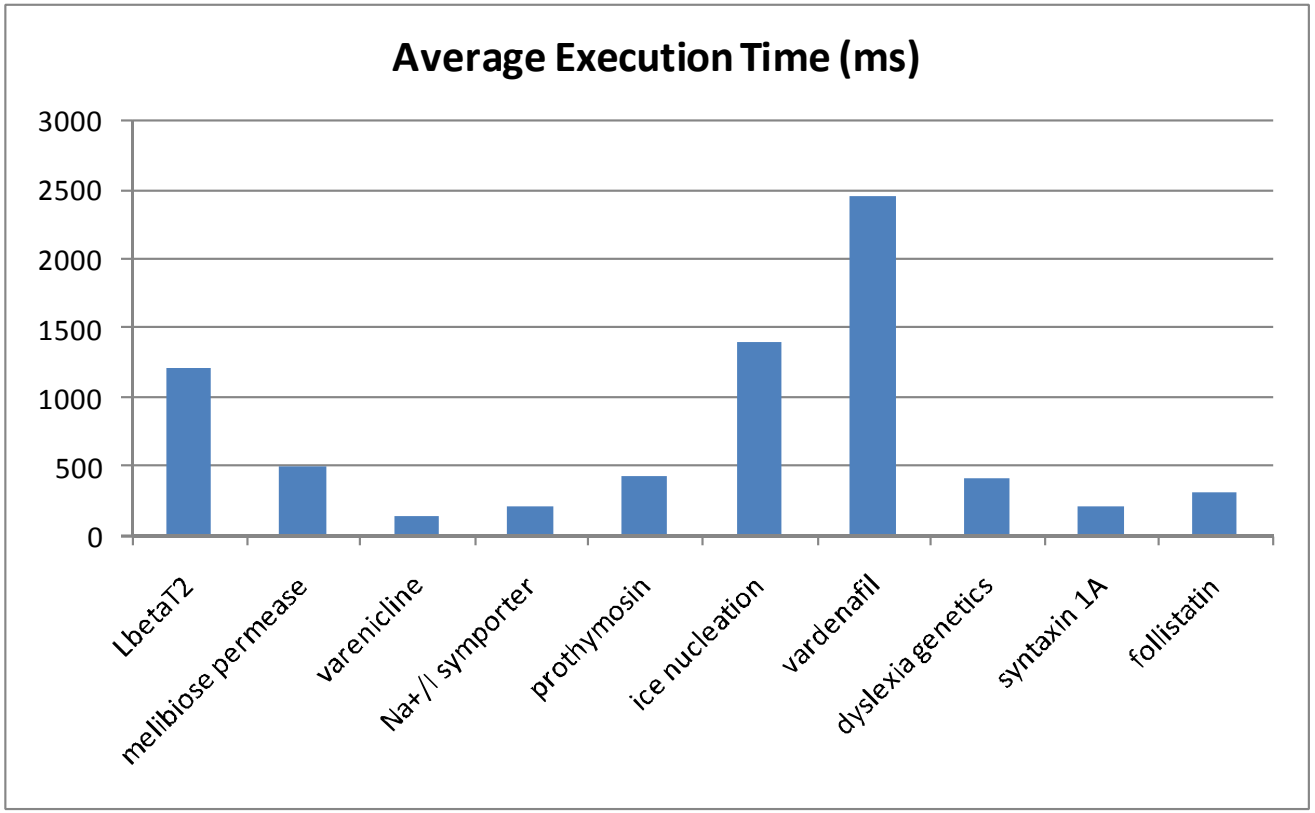
Experimental Evaluation

Query Workload

Keyword(s)	# of Citations in Query Result	Navigation Tree Size	Maximum Tree Width	Maximum Tree Height	Tree Citations w/ Duplicates	Target Concept	MeSH Level of Target Concept	|L(n)| of Target Concept	|L^T(n)| of Target Concept
LbetaT2	116	1947	1009	10	14927	Mice, Transgenic	5	11	90804
melibiose permea	160	1324	722	8	14419	Substrate Specificity	3	31	79470
varenicline	162	1830	962	6	11370	Nicotinic Agonists	7	81	18277
Na+/I symporter	163	2596	1367	6	17146	Perchloric Acid	3	7	4250
prothymosin	313	3941	2113	10	30897	Histones	4	15	22741
ice nucleation	474	3181	1776	9	27440	Plants, Genetically Modified	3	2	12330
vardenafil	486	3424	2014	8	40987	Phosphodiesterase Inhibitors	5	401	69984
dyslexia genetics	517	3056	1691	9	45079	Polymorphism, Single Nucleotide	4	18	18843
syntaxin 1A	1115	6589	3764	10	105503	GABA Plasma Membrane Transport Protei	7	11	650
follistatin	1183	6446	3656	10	102946	Follicle Stimulating Hormone	6	157	34540

Experimental Evaluation

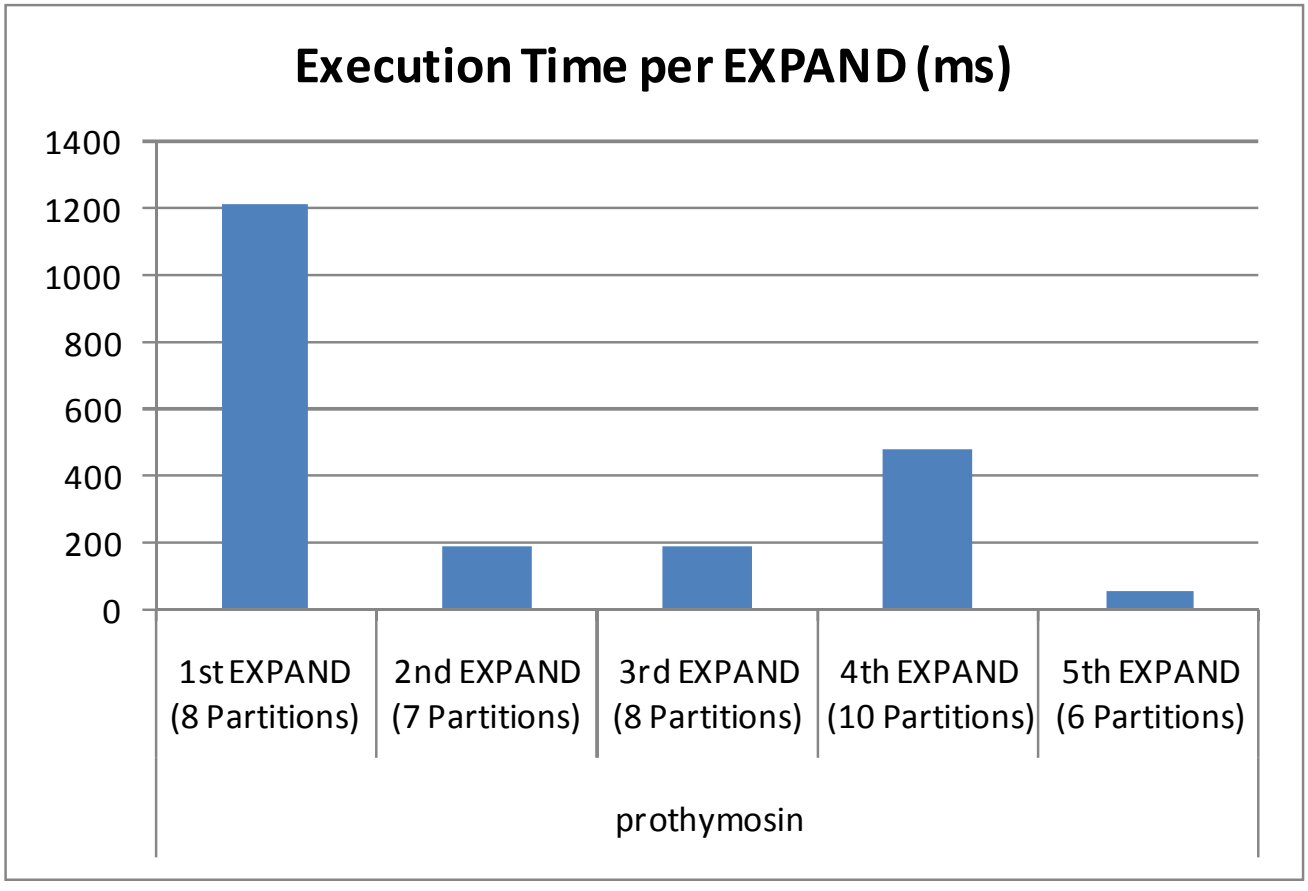
Performance Experiments



Average execution time of navigation

Experimental Evaluation

Performance Experiments



Execution per-EXPAND action for query 'prothymosin'

BioNav

Future work

- Fully integrate categorization and ranking methods
- Include user preferences in the cost model
- Explore query-history as a source of user-preference
- Leverage user preferences to suggest better query keywords
- Explore alternate cost model based on work on graph summarization