Answering Top-*k* Representative Queries on Graph Databases

Sayan Ranu
Dept. of CSE
IIT Madras
Chennai, India.
sayan@cse.iitm.ac.in

Minh Hoang
Dept. of Computer Science
University of California
Santa Barbara, CA, USA.
mhoang@cs.ucsb.edu

Ambuj Singh
Dept. of Computer Science
University of California
Santa Barbara, CA, USA.
ambuj@cs.ucsb.edu

ABSTRACT

Given a function that classifies a data object as relevant or irrelevant, we consider the task of selecting k objects that best represent all relevant objects in the underlying database. This problem occurs naturally when analysts want to familiarize themselves with the relevant objects in a database using a small set of k exemplars. In this paper, we solve the problem of *top-k representative queries* on graph databases. While graph databases model a wide range of scientific data, solving the problem in the context of graphs presents us with unique challenges due to the inherent complexity of matching structures. Furthermore, top-k representative queries map to the classic Set Cover problem, making it NP-hard. To overcome these challenges, we develop a greedy approximation with theoretical guarantees on the quality of the answer set, noting that a better approximation is not feasible in polynomial time. To further optimize the quadratic computational cost of the greedy algorithm, we propose an index structure called *NB-Index* to index the θ-neighborhoods of the database graphs by employing a novel combination of Lipschitz embedding and agglomerative clustering. Extensive experiments on real graph datasets validate the efficiency and effectiveness of the proposed techniques that achieve up to two orders of magnitude speed-up over state-of-the-art algorithms.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance

Keywords

Graph Search, Representative power, top-k

1. INTRODUCTION

Top-k queries play a critical role in various domains such as e-commerce, social networks, and multimedia and scientific databases. In traditional top-k formulations [22], given a function to quantify

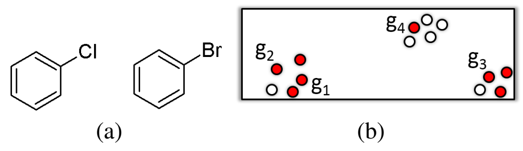


Figure 1: (a) An example of redundant information in traditional top-k answer sets. (b) A sample metric space where two objects are similar if they are located close to each other. Red objects denote those that are relevant.

object relevance, the goal is to identify the k most relevant objects. However, if multiple objects in the answer set are highly similar to each other, the information content embedded in these objects is significantly diminished. For example, in drug design, chemists often want to extract from a molecular database only a small subset of molecules that exhibit high binding affinity toward certain protein targets while preserving other desirable properties such as low toxicity. More importantly, this subset should be compact and informationally dense to be manageable for a human-centric analysis and inexpensive assay. Traditional top-k framework would return the k best molecules according to a scoring function on the desired properties of the molecules. A hypothetical answer set under this framework is shown in Fig. 1(a), which contains two molecules: chloro-benzene and bromo-benzene. Even if both molecules are scored high by the top-k query function, a chemist is likely aware of the fact that replacing the chlorine with any other halogen would result in a molecule with similar properties. Therefore, to maximize the utility of the answer set, it is desirable to identify molecules that are high-scoring, structurally diverse, and representative of the different structural groupings in the database.

The possibility of information redundancy in traditional top-k formulations has ignited much interest in diversification of search results [2, 3, 6, 9, 15, 24, 25, 27]. While different models of diversity exist, at their core, all models penalize the relevance of an object if it is not diverse enough with regard to the objects that have already been added to the answer set. Focusing on diversity however, is not enough to maximize the information content. Consider the metric space in Fig. 1(b), where red-filled objects denote those that are relevant. Assume g_1 has already been added to the answer set and g_3 and g_4 are equi-distant to g_1. In a diversity-aware model, both g_3 and g_4 have the same score since they are equally relevant and diverse. However, it is clear that g_3 is representative of a cluster of other relevant objects, whereas g_4 is a relevant outlier. In other words, a diverse cluster center such as g_3 encodes more information than a diverse outlier like g_4 and should therefore be preferred. Thus, to combat information redundancy, we address the following

Table 1: Example applications with the combination of feature vectors and graphs. The properties of each graph g_i in the database is characterized by a feature vector $\overrightarrow{g_i}$. A graph is relevant if $q(\overrightarrow{g_i})$ is higher than some relevance threshold.

Property	**Example 1: Molecular Library**	**Example 2: Information Cascades**	**Example 3: Bug Analysis**	**Example 4: Social Networks**
Graph object g_i	A molecule in the database	An information cascade structure	A function call graph	d-hop neighborhood of an expert in a social network
Feature vector $\vec{g_i} = (v_{i,1}, v_{i,2}, \cdots, v_{i,m})$	Binding compatibility against m different proteins	Set of topics covered in the cascade	Count frequency over m days	Expertise areas of the social or collaboration group g_i
Target	Molecules with the most desirable properties	Most relevant cascades on a given topic set T	Frequently occurring bugs	Most knowledgeable groups of users on expertise areas E
Query function $q(\vec{g_i})$	$q(\vec{g_i}) = \sum_{j=1}^{m} v_{i,j}$	$q(g_i, T) = \frac{\lvert \vec{g_i} \cap T \rvert}{\lvert \vec{g_i} \cup T \rvert}$	$q(\vec{g_i}) = w^T \vec{g_i},$	$q(\vec{g_i}, E) = \lvert \vec{g_i} \cap E \rvert$

problem: *Given a budget k, which are the k relevant objects that best represent the underlying database?* More specifically, let $\mathbb{D}$ be a database, and each object $o \in \mathbb{D}$ is tagged with a feature vector $\overrightarrow{o}$. Users provide a query function $q : \overrightarrow{o} \rightarrow \{-1, 1\}$ to categorize o as relevant (1) or non-relevant (-1). In addition, an object o is *representative* of another object $o' \in \mathbb{D}$ if $d(o, o') \leq \theta$, for some distance function d and a user-provided distance threshold θ. Our goal is now to identify the k objects that are relevant and most representative of the remaining relevant objects in the database.

In this paper, we focus on the setting where each object is a graph. Such graphs can be used to model a wide range of scientific data such as chemical compounds [20, 23], system call graphs [8], communication graphs [17], social networks [7] and gene interaction networks [21], as demonstrated in Table 1. Example 1 in Table 1 formalizes the problem for molecular libraries discussed earlier. Example 2 identifies the spectrum of information cascade patterns under a set of user-provided topics. A traditional top-k query is prone to identifying cascades from a single community of highly active users. For example, cascades arising out of populous countries such as USA, China or India are likely to eclipse remaining communities. This effect can be negated by being aware of the representative power of each cascade and favor those that are active and remain to be represented. Example 3 computes a summary of the most diverse set of bugs that occurred recently in software bug analysis. A traditional top-k query here is likely to identify function call graphs that share the same core bug-inducing subgraph. By rewarding representativeness and diversity, we can identify the entire spectrum of subgraphs that induce bugs. Finally, in social or collaboration networks such as DBLP [1] (example 4), each node (or user) can be described by a collection of tags to represent his/her interests or expertise areas. Given a set of expertise areas as query, we can identify the most knowledgeable groups, where each group is based on the neighborhood of a node. A traditional top-k query in this case is likely to return groups with large overlaps among them. In contrast, a top-k representative query finds neighborhoods that are relevant as well as non-overlapping. In a nutshell, the combination of feature vectors with graphs, as well as the need to maximize information content within a budget k, positions the proposed problem in a unique space, which has not been studied before. Our main contributions are as follows:

- We propose a flexible and intuitive model for diversity to reward the representative power of the answer set. The flexibility is achieved by allowing users to define relevance at query-time and controlling the answer set size through a budget. While we focus on graphs, the proposed algorithm is generalizable to all metric spaces.
- We prove that top-k representative query is NP-hard and propose a technique to compute a constant factor approximation of the optimal answer set. We also show that no other polynomial time algorithm can provide a better approximation unless $P = NP$.
- We develop an index structure called *NB-Index* to index the θ-neighborhoods of graphs. Our proposed index structure employs a novel combination of Lipschitz embedding [5] and agglomerative clustering to expedite the answer set computation. In addition to fast answering of top-k representative queries, NB-Index also allows *interactive refinement* of θ to reach the optimal zoom level, while incurring minimal overhead cost.
- Empirical results on real graph datasets demonstrate higher information density in answer sets and a superior performance by up to 2 orders of magnitude speedup over state-of-the-art techniques.

2. PROBLEM FORMULATION

We assume a graph database $\mathbb{D} = \{g_1, \cdots, g_n\}$, where each graph g_i is tagged with a feature vector $\overrightarrow{g_i} = [g_{i,1}, \cdots, g_{i,m}]$ to characterize its properties. Based on a user-provided query function $q : \overrightarrow{g} \rightarrow \{-1, 1\}$, which operates on the feature vector $\overrightarrow{g}$, g is classified as either relevant (1) or irrelevant (-1). Our goal is to maximize the *representative power* of the answer set within a budget k.

DEFINITION 1. TOP-k REPRESENTATIVE QUERIES: *Given a query function $q(\overrightarrow{g})$, distance threshold θ and a budget k, compute the set of graphs $\mathbb{A}$, such that*

$$\mathbb{A} = \arg\max_{\mathbb{S}} \{\pi_\theta(\mathbb{S}) \mid \mathbb{S} \subseteq \mathbb{L}_q,\ |\mathbb{S}| = k\} \tag{1}$$

where $\mathbb{L}_q = \{g \in \mathbb{D} \mid q(\overrightarrow{g}) = 1\}$ is the set of relevant graphs with respect to q, and $\pi_\theta(\mathbb{S})$ is the representative power of $\mathbb{S}$.

To quantify the representative power of a graph or a set of graphs, we first define the θ*-neighborhood* of a graph.

DEFINITION 2. θ-NEIGHBORHOOD. *The θ-neighborhood of a graph g, denoted as $N_\theta(g)$, contains all relevant database graphs within a distance threshold θ from g.*

$$N_\theta(g) = \{g' \in \mathbb{L}_q \mid d(g, g') \leq \theta\} \tag{2}$$

where $d(g, g')$ is the classical graph edit distance [12, 28] and θ is a user provided threshold. In other words, g is a structural representative of all graphs in its θ-neighborhood $N_\theta(g)$. Consequently, it is not desirable to have two graphs with a large overlap between their θ-neighborhoods in the answer set. By combining these intuitions, the representative power $\pi_\theta(\mathbb{S})$ of a set of graphs $\mathbb{S}$ is defined as the proportion of relevant graphs represented by $\mathbb{S}$.

$$\pi_\theta(\mathbb{S}) = \frac{\left|\bigcup_{g \in \mathbb{S}} N_\theta(g)\right|}{|\mathbb{L}_q|} \tag{3}$$

The proposed model captures as much of the various relevant structural groupings as possible within the provided budget. As a natural consequence of maximizing the representative power, the

Table 2: Summary of the notations used.

Notation	Explanation
$q(\cdot)$	User-provided relevance function that maps each object to {-1,1}.
$\mathbb{L}_q$	The set of relevant graphs with respect to relevance function $q(\cdot)$.
θ	User-provided distance threshold to quantify if two graphs are similar.
$N(g)$	All graphs that are within distance θ of g, and therefore *represented* by g.
$\pi(\mathbb{S})$	The representative power of a set of graphs as defined in Eq. 3
$d(g, g')$	The edit distance between graphs g and g'.
$d_v(g, g')$	The vantage distance between graphs g and g' as defined in Def. 4

answer set also favors structural diversity. For brevity's sake, hereon, we denote the representative power $\pi_\theta(\{g\})$ of a graph g as $\pi_\theta(g)$. Additionally, the notations $\pi(g)$ and $N(g)$ are used to denote the representative power and θ-neighborhood, respectively, for arbitrary θ's. Our notation is summarized in Table 2.

3. EXISTING DIVERSITY MODELS

As discussed in Sec. 1, a number of models exist to favor diversity in the top-k answer set. In this section, we discuss two recent models that are closest to ours.

3.1 DisC

DisC [9] recognized the need to go beyond diversity and focus on the representative power of the answer set. Given the set of relevant objects $\mathbb{S}$, DisC attempts to find the smallest set of objects that represent all relevant objects. Mathematically, DisC computes an answer set $\mathbb{A}$, such that $\forall o_1 \in \mathbb{S}, \exists o_2 \in \mathbb{A}$, where $d(o_1, o_2) \leq \theta$ and $\forall o_2,\ o_3 \in \mathbb{A},\ o_2 \neq o_3,\ d(o_2, o_3) > \theta$. There are two main differences between our approach and DisC.

1. Static set of relevant objects: DisC assumes that the set of relevant objects in a given database is static. At times, the set of relevant objects are query-dependent even though the underlying database may be static. For example, in a database of information cascade graphs, one might be interested in only those that discuss a certain topic, such as sports. Owing to this assumption, the index structure proposed in DisC needs to be rebuilt whenever the relevant objects change. In this paper, we focus on a dynamic setting where the relevant objects are defined at query time by the user through a query function $q(\cdot)$.

2. Coverage of answer set: DisC requires the coverage of all relevant objects while we quantify the coverage of an answer set and seek to maximize it. DisC's requirement to represent all relevant objects can make the answer set too large. Such scenarios occur when a dataset contains some objects that are relevant but not clustered together with other relevant objects. Graph g_4 in Fig. 1(b) is an example of such objects. These relevant outliers have low representative power. Consequently, their presence in the answer set dilutes the *compression ratio* $\frac{|\mathbb{S}|}{|\mathbb{A}|}$, which is the average number of relevant objects represented by each object in the answer set.

To study this phenomenon empirically, we examined the answer set produced by DisC in a graph database representing the DUD molecular repository [10]. We use the classical graph edit distance [12, 28] for this experiment under a distance threshold $\theta = 10$ to define if a graph g_1 represents graph g_2. First, we identify the set of relevant molecules that are active against the enzyme Acetylcholine ezterase (AChE), a key component for curing Alzheimer's disease [4]. Next, we plot the growth rate in the answer set size as the number of relevant objects is varied. Fig. 2(a) presents the results. The growth rate is almost linear and on average, the answer set is one third of the number of relevant molecules. Clearly, a higher compression ratio is desirable and this is compromised due to the

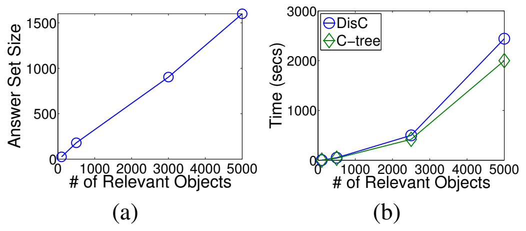


Figure 2: (a) The growth rate of DisC answer set size against the number of relevant objects. (b) The growth rate in the running time of the DisC model in the presence of graphs.

presence of relevant outliers. More critically however, there is no control over the answer set size.

3.2 Tuning diversity models to reward representativeness

One of the most generic diversity models is proposed in [19]. Let us refer to this technique as *DIV*. Given a database of objects $\mathbb{D}$, the goal is to identify the subset $\mathbb{S} \subseteq \mathbb{D}$ of k objects, such that $\sum_{g \in \mathbb{S}} score(g)$ is maximized and $\forall g_1, g_2 \in \mathbb{S},\ d(g_1, g_2) > \theta$. $score(g)$ denotes the value of an object (or graph) g. In our problem, the goal is to maximize representativeness. Thus, by assigning $\pi(g)$ as the score of graph g, we can attempt to maximize the representativeness of the answer set. However, in this model, $score(g)$ of a graph g is assumed to be static and independent of other objects in the answer set. In contrast, in our model, $\pi(g)$ is dependent on other graphs that are already in the answer set. More formally, for DIV, when $\pi(g)$ is used as the score, even in the presence of the constraint $\forall g_1, g_2 \in \mathbb{S},\ d(g_1, g_2) > \theta$, the maximization function $score(\mathbb{S}) = \sum_{\forall g \in \mathbb{S}} score(g) = \sum_{\forall g \in \mathbb{S}} \pi(g) \neq \pi(\mathbb{S})$. Consequently, the model cannot be used to solve our problem. To ensure score independence, we need to enforce the stricter constraint of $\forall g_1, g_2 \in \mathbb{S},\ d(g_1, g_2) > 2\theta$. This constraint, however, poses the risk of ruling out highly representative graphs from inclusion in the answer set and therefore will not maximize $\pi(\mathbb{S})$.

From the indexing perspective, three different schemes are proposed in DIV. However, all of them rely on an upper bound, which assumes scores of graphs in the answer set to be mutually independent. As a result, the proposed indexing techniques cannot be used in our problem. Beyond the independence assumption, DIV constructs a *diversity-graph* where each node is a graph in the database and two nodes are connected if $d(g_1, g_2) \leq \theta$. Since θ is provided at query-time in our model, the diversity-graphs need to be computed online, which is a computational bottleneck.

4. PROPERTIES OF TOP-K REPRESENTATIVE QUERIES

In this section, we analyze the complexity of the proposed problem and investigate the theoretical guarantees that can be achieved.

4.1 NP-hardness

THEOREM 1. *Top-k representative query is NP-hard.*

PROOF: We reduce the *Set Cover problem* to the problem of answering top-k representative queries. The Set Cover problem is defined as follows: Given a universe of elements $U = \{e_1, e_2, \cdots, e_n\}$ and a collection of subsets $S = \{\ S_1,\ S_2,\ \cdots,\ S_m\}$, the Set Cover problem seeks to determine whether there exists a collection $S' \subseteq S$ of k subsets, such that $\bigcup_{S_i \in S'} S_i = U$.

Given an arbitrary instance of the Set Cover problem, we construct a graph database containing three sets of graphs: D_1, D_2, and D_3 all of which are relevant. D_1 contains graph s_i corresponding to each set $S_i \in S$, and D_2 contains graph u_j corresponding to each element $e_j \in U$. Graph $u_j \in N(s_i)$ if and only if $e_j \in S_i$. From symmetry of graph edit distance, the θ-neighborhoods of all graphs in D_2 can be computed analogously. Now, let $x = max\{\pi(u) | u \in D_2\}$. D_3 contains $|S|$ disjoint groups of graphs, such that each group G_i contains x graphs and all graphs in G_i are in the θ-neighborhood of graph $s_i \in D_1$. In other words, D_3 contains $x|S|$ graphs and it ensures that any graph in D_1 has a higher representative power than graphs in D_2 or D_3. We now perform top-k representative query on this database.

It is easy to see that there is a set cover of size k if and only if there exists an answer set $\mathbb{A}$ where $\pi(\mathbb{A}) = \frac{|D_2|+k(x+1)}{|D_1|+|D_2|+|D_3|}$. From our construction, no graph from either D_2 or D_3 will be selected in $\mathbb{A}$. Additionally, the number of graphs added to $\pi(\mathbb{A})$ from D_3 is a constant number kx. As a result, $\pi(\mathbb{A})$ is maximized when, in addition to the k answer set graphs and their corresponding kx neighbors from D_3, all graphs in D_2 are in $\pi(\mathbb{A})$. On the other hand, if subsets $S_{i_1}, S_{i_2}, \cdots, S_{i_k}$ form a set cover, then including the corresponding graphs in answer set $\mathbb{A}$ results in $\pi(\mathbb{A}) = \frac{|D_2|+k(x+1)}{|D_1|+|D_2|+|D_3|}$. □

4.2 Submodularity

A function $f(.)$ is submodular if the marginal gain from adding an element to a set S is at least as high as the marginal gain from adding it to a superset of S. Mathematically, it satisfies:

$$f(S \cup \{o\}) - f(S) \geq f(T \cup \{o\}) - f(T) \quad (4)$$

for all elements o and all pairs of sets $S \subseteq T$. For submodular and monotone functions, the greedy algorithm of iteratively adding the element with the maximum marginal gain approximates the optimal solution within a factor of $(1 - \frac{1}{e})$ [18]. We omit the proof of monotonicity for Eqn. 3 due to space limitations.

THEOREM 2. *Eqn. 3 is submodular.*

PROOF BY CONTRADICTION: Assume,

$$\pi(T \cup \{g\}) - \pi(T) > \pi(S \cup \{g\}) - \pi(S) \quad (5)$$

where S and T are sets of graphs, such that $S \subseteq T$, and $g \in \mathbb{D}$ is the graph being added. From Eqn. 5, we can say:

$$N(g) \backslash N(T) > N(g) \backslash N(S)$$
$$\text{or, } S \nsubseteq T \quad (6)$$

which contradicts the assumption that $S \subseteq T$. □
Thus,

$$\pi(\mathbb{A}_{greedy}) \geq (1 - \frac{1}{e})\pi(\mathbb{A}^*) \quad (7)$$

where $\mathbb{A}_{greedy}$ is the answer set computed using a greedy algorithm and $\mathbb{A}^*$ is the optimal answer set. Since the scoring function is normalized, i.e., $\pi(\emptyset) = 0$, no other polynomial time algorithm can provide a better approximation guarantee than $(1 - \frac{1}{e})$ unless $P = NP$, as proved in [11].

5. THE SIMPLE GREEDY APPROACH

Alg. 1 presents the pseudocode for the baseline greedy approach. From Theorem 2, Alg. 1 guarantees an approximation of $(1 - \frac{1}{e})$ or better. Unfortunately, in the presence of graphs, this approach is not scalable. The bottleneck lies in the neighborhood update step

Algorithm 1 Baseline-greedy($q(\cdot), \theta, k$)

```
1: Compute $\mathbb{L}_q$
2: $\mathbb{A} \leftarrow \emptyset$
3: while $|\mathbb{A}| < k$ do
4:     $g^* \leftarrow \arg\max_g \{\pi(\mathbb{A} \cup g) - \pi(\mathbb{A}) \mid g \in \mathbb{L}_q\}$
5:     $\mathbb{A} \leftarrow \mathbb{A} \cup g$
6:     for $\forall g \in \mathbb{L}_q \backslash \mathbb{A}$ do
7:         $N(g) \leftarrow N(g) \backslash N(g^*)$
8: return $\mathbb{A}$
```

(lines 6-7), which is performed at each iteration and requires $O(n^2)$ edit distance computations in the worst case. Computing the edit distance between two graphs is NP-hard [28]. As a result, Alg. 1 is not scalable to large databases. To mitigate this bottleneck, we need to index the neighborhood update step. More precisely, given the θ-neighborhoods of two graphs, we need to compute their overlap without incurring a quadratic computation cost with respect to the neighborhood sizes.

Indexing graph edit distance has been studied in the context of top-k nearest neighbor queries [12, 28]. The demands of indexing θ-neighborhoods, however, are different.

1. Overlap quantification: Instead of identifying the k nearest neighbors to a query graph, our goal is to compute the extent of the overlap between θ-neighborhoods of two graphs without incurring a quadratic computational cost. This ensures an efficient update of their representative power at each iteration.

2. Order independence: In our problem, ordering neighbors based on distance is not necessary; just detecting containment within the θ-neighborhood is sufficient. In top-k similarity queries, the ordering is central to solving the problem.

To establish the scalability challenge in the presence of graphs, we study the running time growth rate of Alg. 1 against database size. Furthermore, we also investigate the impact of two index structures that are both built for indexing nearest neighbor queries: the state-of-the-art graph indexing technique C-tree [12] and the index structure proposed by DisC, which is an adaptation of M-tree [29]. Fig. 2(b) shows the results. Regardless of the underlying indexing technique, it takes more than 35 minutes to compute the answer even on a small dataset containing only 5000 objects. This result highlights the need to go beyond traditional graph indexing techniques, which optimize nearest neighbor queries. To scale our problem, we need to index θ-neighborhoods.

6. INDEXING θ-NEIGHBORHOODS

Equipped with the insights derived from Sec. 5, we proceed toward analyzing the properties of graph edit distance, and then design an index structure called *NB-Index*.

6.1 Triangular inequality based pruning

The graph edit distance satisfies triangular inequality when the individual vertex and edge distances are metric. Mathematically, $d(g_1, g_2) + d(g_2, g_3) \geq d(g_1, g_3)$ for any three graphs g_1, g_2, g_3. Based on this property, we can easily derive the following theorem.

THEOREM 3. *For any two graphs g_1 and g_2, if $d(g_1, g_2) > 2\theta$, $N(g_1) \cap N(g_2) = \emptyset$.*

Theorem 3 highlights the property that, once the graph providing the highest marginal gain is added to the answer set, only the θ-neighborhoods of those relevant graphs within a distance of 2θ from that new graph will need to be updated. While this property does reduce the computational cost, scalability cannot be guaranteed. First, its efficiency depends on the value of θ. The smaller the value of θ is, the more efficient this property becomes. More importantly, to have a significant impact on the scalability, a technique must be designed to efficiently update the θ-neighborhoods of graphs which are within the 2θ boundary. Thus, to further improve scalability, we employ a Lipschitz embedding [5] of the metric space and design the concept of *Vantage Ordering*.

6.2 Vantage Ordering

Vantage Orderings (VO) perform a Lipschitz embedding of the metric space to speed up the indexing of the θ-neighborhoods.

DEFINITION 3. VANTAGE POINT AND VANTAGE ORDERING: *Given a metric space $\mathcal{M} : \{\mathbb{D}, d : (\mathbb{D},\mathbb{D}) \rightarrow \mathbb{R}_+\}$, let $v \in \mathbb{D}$ be a randomly selected graph. $\forall g \in \mathbb{D}$, $\mathcal{M}$ is embedded into a 1-dimensional feature space $\mathcal{F}_v$, where the feature vector of a graph g is $[d(v,g)]$. The 1-dimensional ordering of graphs in $\mathcal{F}_v$ is termed the Vantage Ordering of $\mathcal{M}$ with respect to the vantage point v.*

As defined above, VO captures a feature space representation of $\mathcal{M}$ from the viewpoint of the vantage point (VP) v. To disambiguate the resultant feature space of Lipschitz embedding from the feature vector representation of a graph on which the query function operates, hereon, we use the term vantage space to denote $\mathcal{F}_v$. Now, to quantify the distance between two graphs from a VP's viewpoint in $\mathcal{F}_v$, we introduce the concept of *Vantage Distance*.

DEFINITION 4. VANTAGE DISTANCE $d_v(g,g')$: *Given a vantage point v, the vantage distance $d_v(g,g')$ between any two graphs $g,\ g' \in \mathbb{D}$ is $|d(v,g) - d(v,g')|$.*

Now, we observe the following.

THEOREM 4. *Given a VP v, if $d_v(g,g') > \theta$, $g' \notin N(g)$.*

PROOF: From triangular inequality, $d(g,g') \geq |d(v,g) - d(v,g')| = d_v(g,g') > \theta$. Thus, $g' \notin N(g)$. □

Theorem 4 provides a mechanism to bound the maximum vantage distance between any two graphs g, g', where $g' \in N(g)$. It is straightforward to see that the information retained in the vantage space can be boosted by employing a set of VPs $\mathbb{V}$ rather than a single one. We denote this vantage space as $\mathcal{F}_{\mathbb{V}}$. In $\mathcal{F}_{\mathbb{V}}$, we can deduce the following theorem.

THEOREM 5. *Let $\hat{N}(g)$=$\{g'|d_v(g,g')\leq \theta\ \forall v\in \mathbb{V}\}$. $\hat{N}(g) \supseteq N(g)$.*

Theorem 5 allows us to compute an upper bound on the actual θ-neighborhoods based on the VOs. The VOs can be pre-computed as part of indexing procedure, and thus, computing $\hat{N}(g)$ only requires $|\mathbb{V}|$ linear scans. Furthermore, since the entire computation is performed on the vantage space, the expensive NP-hard computations of edit distances are required only on graphs $g' \in \hat{N}(g)$.

6.2.1 Choosing VPs

The performance gain achieved due to VPs is directly proportional to the *False Positive Rate (FPR)*, which is the probability of a graph being in $\hat{N}(g)$ but not in $N(g)$. FPR is quantified as follows under the simplifying assumption that the distances $d(g,g')$ and $d(g,g'')$ are independent.

$$FPR = P(g' \in \hat{N}(g) \backslash N(g)) = P(d(g,g') > \theta) \prod_{v\in\mathbb{V}} P(d_v(g,g') \leq \theta) \quad (8)$$

It is clear from Theorem 5 that the higher the number of VPs, the better the tightness of $\hat{N}(g)$. On the other hand, both the computational cost of $\hat{N}(g)$ and the storage footprint of the VOs increase with $|\mathbb{V}|$. It is therefore critical to have a deep understanding of the relationship between the FPR and $|\mathbb{V}|$. Towards that goal, given the number of VPs $|\mathbb{V}|$, we derive theoretical bounds on the FPR. The FPR is dependent on the underlying distribution of the graph distances. We thus choose the two most commonly encountered distributions, *Gaussian* and *Uniform*, and derive the bounds below.

1. Normally distributed metric space: Let us assume that the distance between any two randomly chosen graphs is distributed normally where $d(g,g') \in \mathcal{N}(\mu,\sigma^2)$ and $m\theta$ is the diameter of the metric space $\mathcal{M}$. Under this assumption,

$$\begin{aligned} P(d(g,g') > \theta) &= P\left(\frac{d(g,g')-\mu}{\sigma} > \frac{\theta-\mu}{\sigma}\right) \\ &= P\left(Z > \frac{\theta-\mu}{\sigma}\right) \\ &= 1-\phi\left(\frac{\theta-\mu}{\sigma}\right) \end{aligned} \quad (9)$$

where $Z = \mathcal{N}(0,1)$ is the standard normal variable and $\phi(x)$ is the cumulative distribution function of Z.

The next step in computing the FPR is to compute $P(d_v(g,g') \leq \theta)\ \forall v \in \mathbb{V}$. With respect to a vantage point v, if $d(g,v) = x$, then $P(d_v(g,g') \leq \theta) = P(x-\theta \leq d(g',v) \leq x+\theta)$. Therefore,

$$\begin{aligned} &P(d_v(g,g') \leq \theta) \\ &= \int_0^{m\theta} P\left(d(g,v)=x\right) P\left(x-\theta \leq d(g',v) \leq x+\theta\right)\,\mathrm{d}x \\ &= \int_0^{m\theta} P\left(d(g,v)=x\right) P\left(\frac{x-\theta-\mu}{\sigma} \leq Z \leq \frac{x+\theta-\mu}{\sigma}\right)\,\mathrm{d}x \\ &\leq \int_0^{m\theta} P\left(d(g,v)=x\right) P\left(\frac{\mu-\theta-\mu}{\sigma} \leq Z \leq \frac{\mu+\theta-\mu}{\sigma}\right)\,\mathrm{d}x \\ &= \left(\phi\left(\frac{\theta}{\sigma}\right)-\phi\left(\frac{-\theta}{\sigma}\right)\right)\int_0^{m\theta} P\left(d(g,v)=x\right)\,\mathrm{d}x \\ &= 2\phi\left(\frac{\theta}{\sigma}\right)-1 \end{aligned} \quad (10)$$

Combining Eq. 9 and Eq. 10, the FPR for $|\mathbb{V}|$ vantage points can be expressed as:

$$FPR \leq \left(1-\phi\left(\frac{\theta-\mu}{\sigma}\right)\right)\left(2\phi\left(\frac{\theta}{\sigma}\right)-1\right)^{|\mathbb{V}|} \quad (11)$$

2. Uniformly distributed metric space: Let us assume that $d(g,g') \in \mathcal{U}(0,m\theta)$ where m is the diameter of the metric space. Now, since the VPs are selected randomly, for any graph g, each dimension $d(v,g)$ is distributed uniformly. The FPR can therefore be quantified as:

$$\begin{aligned} FPR &= \frac{m\theta-\theta}{m\theta}\prod_{v\in\mathbb{V}}\frac{\theta}{m\theta} \\ &= \frac{m-1}{m}\frac{1}{m^{|\mathbb{V}|}} \end{aligned} \quad (12)$$

6.3 Updating representative power based on clusters

Sec. 6.2 provides an upper bound on the θ-neighborhood of a graph based on a vantage space analysis. In this section, we analyze the structural space directly and develop techniques to update the representative power of a graph in the presence of clusters. The technique developed in this section allows *batch updates*: a single calculation to compute upper bounds on the representative power of a cluster of similar graphs. Here, a cluster is a set of structurally

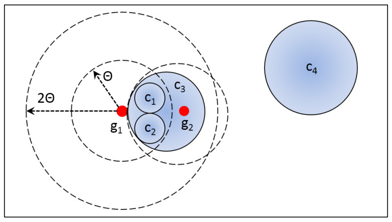


Figure 3: A representation of the structural space in the presence of clusters.

similar graphs. Fig. 3 shows a sample structural space of graphs with four clusters, c_1-c_4, depicted as shaded circles. Clusters c_1 and c_2 are contained within c_3. Consider two relevant graphs g_1 and g_2, where g_2 is contained within cluster c_3. The two dashed rings centered on g_1 with radii of θ and 2θ represent $N(g_1)$ and the space of graphs whose θ-neighborhoods overlap with $N(g_1)$. g_2 represents one such graph.

Now, assume that g_1 has been added to the answer set in the last iteration, and the θ-neighborhoods of the rest of the relevant graphs need to be updated. Since c_4 is more than 2θ away from g_1, neighborhoods of all graphs in c_4 are unaffected and do not need to be updated. In contrast, the neighborhoods of all graphs in clusters c_1, c_2 and c_3 are affected and need updating. Moreover, both c_1 and c_2 are fully contained within the θ-neighborhood of g_1. Thus, for any graph in c_1, their updated θ-neighborhoods must not contain any of the graphs in c_1. The same rule applies for c_2 as well. Whereas, this rule does not work for c_3 because it is not fully contained within the θ-neighborhood of g_1. However, graphs in c_3, such as g_2, could be affected. More specifically, $N(g_2)$, depicted using the dashed ring centered on g_2, needs to be reduced by its overlap with $N(g_1)$, depicted as the ring of radius θ centered on g_1. Furthermore, since c_1 and c_2 are completely contained within this overlap region, $N(g_2)$ must at least be reduced by the combined “area” of c_1 and c_2.

To formalize the above insights, we first introduce the definitions of *cluster radius* and *cluster diameter* with respect to a metric space. Since a cluster is essentially a set of graphs, we use set notations in the following definitions and proofs. Let $centroid(c)$ denote the centroid graph of a cluster c.

DEFINITION 5. CLUSTER RADIUS AND DIAMETER: *The radius of a cluster is the maximum of all the pairwise distances between the centroid and other constituent graphs in the cluster. Similarly, the diameter is the largest of all the pairwise distances.*

$$radius(c) = max\,\{d(centroid(c), g)\ \forall g \in c\}$$
$$diameter(c) = max\,\{d(g, g')\ \forall g,\ g' \in c\} \leq 2 \times radius(c)$$

Now, given a cluster c, we can derive the upper bound $d_{ub}(g, c)$ and lower bound $d_{lb}(g, c)$ for the distance between a graph g and $\forall g' \in c$ using triangular inequality.

$$d_{ub}(g, c) = d(g, centroid(c)) + radius(c)$$
$$d_{lb}(g, c) = max\,\{0, d(g, centroid(c)) - radius(c)\}$$

Now, let g represent the latest graph added to the answer set, c be a cluster, and $\pi(g')^*$ and $\pi(g')$ be the representative power of graph g' after and before g is added to the answer set respectively. Additionally, let $c_q = c \cap \mathbb{L}_q$ be the subset of relevant graphs in c. Using the distance bounds stated above, we have the following theorems:

THEOREM 6. *If $d_{lb}(g, c) > 2\theta$, then $\forall g' \in c$, $\pi(g')^* = \pi(g')$.*

PROOF: We omit the proof for its simplicity. □

THEOREM 7. *If $d_{ub}(g, c) \leq \theta$ and $diameter(c) \leq \theta$, then $\forall g' \in c$, $\pi(g')^* \leq \pi(g') - \frac{|c_q|}{|\mathbb{L}_q|}$.*

PROOF: Since $diameter(c) \leq \theta$, for any graph $g' \in c$,

$$N(g') \supseteq c.$$

Since $d_{ub}(g, c) \leq \theta$, $N(g) \supseteq c$. Thus,

$$N(g) \cap N(g') \supseteq c.$$
$$\text{or, } \pi(g')^* \leq \pi(g') - \frac{|c_q|}{|\mathbb{L}_q|} \qquad \square$$

THEOREM 8. *Let c be a cluster and $\mathbb{C} = \{c' \mid c' \subseteq c, c'$ satisfies Theorem 7$\}$ be the set of sub-clusters of c that satisfy Theorem 7. Additionally, all clusters in $\mathbb{C}$ are disjoint, i.e., $\forall c_i,\ c_j \in \mathbb{C}, c_i \cap c_j = \emptyset$. If $\theta \leq d_{ub}(g, c) \leq 2\theta$ and $diameter(c) \leq \theta$, then $\forall g' \in c$, $\pi(g')^* \leq \pi(g') - \sum_{c' \in \mathbb{C}} \frac{|c'_q|}{|\mathbb{L}_q|}$*

PROOF: Since $diameter(c) \leq \theta$, for any graph $g' \in c$,

$$N(g') \supseteq c.$$

Now, $N(g) \supseteq \bigcup_{c' \in \mathbb{C}} N(c')$. Thus,

$$N(g) \cap N(g') \supseteq \bigcup_{c' \in \mathbb{C}} N(c').$$

Since clusters in $\mathbb{C}$ are disjoint,

$$\pi(g')^* \leq \pi(g') - \sum_{c' \in \mathbb{C}} \frac{|c'_q|}{|\mathbb{L}_q|} \qquad \square$$

The three theorems above allow us to obtain tight upper bounds on the representative power of a graph by performing a small number of distance computations. An important constraint that drives the efficiencies of Theorems 7 and 8 are the diameters of clusters. If the diameter of any cluster is above θ, then it cannot be used to estimate the overlap between the θ-neighborhoods of two graphs. Therefore, the lower the value of θ, the less effective Theorems 7 and 8 become. Fortunately, the efficiency of Theorem 6 is inversely proportional to θ. As a result, the combined efficiency is *elastic* in nature; for a small θ, Theorem 6 is effective, whereas for a large θ, Theorems 7 and 8 are effective.

6.4 NB-Index

In this section, we develop the index structure called *NB-Index*, which unifies VO and the cluster based bounds into one coherent framework. NB-Index contains two components:

1. Vantage Points: A set of VPs, $\mathbb{V}$, is chosen to capture a feature space representation of database graphs. For each VP, NB-Index maintains the VO of the entire database.

2. NB-Tree: A hierarchical clustering is performed on the database to group structurally similar graphs into disjoint sub-spaces. In our clustering procedure, disjoint clusters are formed in a top-down, recursive manner to form a tree. In this tree, each leaf node is a graph and non-leaf nodes are clusters of its children. As a result, two clusters can overlap only if one is a descendant of the other. Each non-leaf node stores the centroid, radius, and diameter of the corresponding cluster. At leaf nodes, only the feature vectors of the corresponding graphs are stored. While building the tree, we ensure that each non-leaf node contains at most b children. The fan-out controls the desired depth of the tree. For an on-disk implementation, b should be chosen based on the disk-page size for optimizing the cost of node look-ups. On the other hand, in a memory-resident implementation, a small b yields better performance.

The clustering procedure starts at the root node, which contains the entire database. First, b graphs are chosen as pivots to partition the dataset. The first of the b pivots is chosen randomly. For the second pivot, we choose the graph that is farthest from first one. Continuing in the same manner, in each iteration, the pivot is the graph whose minimum distance to the already chosen pivots is the highest. The process is repeated through b iterations. Next, the pivots are assigned as cluster centroids, and all remaining graphs in the database are assigned to the cluster with the closest centroid. Finally, the process is repeated recursively on each of the clusters till the size of a cluster drops below b.

Since computing edit distance is NP-hard, and the indexing procedure requires a large number of pairwise edit distance computation, we use VPs to expedite NB-Tree construction. More specifically, to determine if pivot p is the closest pivot to a graph g, we first compute the VP-based distance $\forall v \in \mathbb{V}$, $\max\{d_v(g,p)\}$, which is a lower bound on the actual edit distance $d(g,p)$. If the lower bound is larger than the currently closest edit distance between g and any of the already evaluated pivots, then p is discarded from consideration. Similarly, to compute the radius of a cluster from the centroid p, if the upper bound $\forall v \in \mathbb{V}$, $\min\{d(v,p)+d(v,g)\}$ is smaller than the current radius, then g is ignored. As a result, expensive edit distances are computed on a small minority of pairs. The diameter of a cluster is set to the summation of the two largest distances from the centroid.

EXAMPLE 1. *Fig. 4 demonstrates the NB-Tree built on the corresponding graph database at* $b = 2$. *For simplicity, we use a 1D feature vector, denoted as* $\vec{g_i}$, *to characterize each graph, and these are chosen arbitrarily. The explanation of* $\hat{\pi}$*-vectors (highlighted in red), which are computed during query time, is discussed in the next section.*

Storage Cost: The storage cost for maintaining the VO of the entire database is $O(|\mathbb{V}||\mathbb{D}|)$, where $\mathbb{V}$ is the set of VPs, and $\mathbb{D}$ is the graph database. The second component of the index is the NB-Tree. Assuming a balanced branching factor of b, the height of NB-Tree is bounded within $\log_b |\mathbb{D}|$. Since the number of nodes at each level of NB-Tree follows a geometric progression, the storage cost is bounded by $O(\frac{b|\mathbb{D}|-1}{b-1})$. The total storage cost is therefore $O(\frac{b|\mathbb{D}|-1}{b-1}) + |\mathbb{V}||\mathbb{D}|)$.

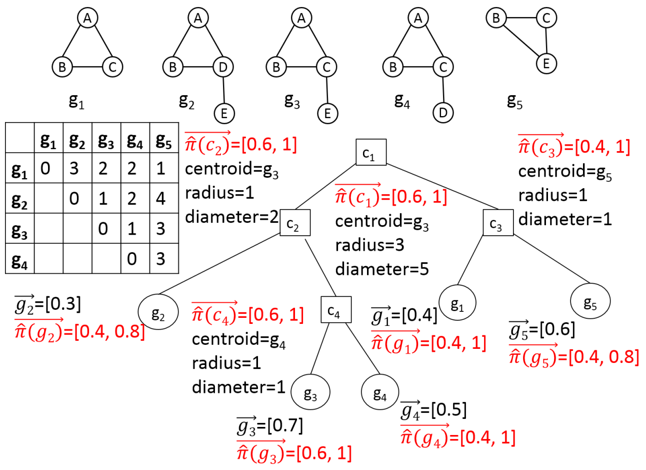
Index Construction Time: For each VP, we need to compute its distance to all graphs in the database. Thus, the cost of constructing the VO is $O(|\mathbb{V}||\mathbb{D}|)$. For NB-Tree, at each level, each graph is compared to b pivots to identify the closest one. Since the height of the tree is bounded by $O(\log_b |\mathbb{D}|)$, a graph is compared to b pivots $\log_b |\mathbb{D}|$ times. Thus, the computation cost is $O(|\mathbb{D}| \times b \log_b |\mathbb{D}|)$. Combining the costs of both VPs and the NB-Tree, the total index construction time is $O\left(|\mathbb{D}| \times (|\mathbb{V}| + b \log_b |\mathbb{D}|)\right)$.

7. QUERY PROCESSING

The goals of the query processing algorithm are two-fold:

1. Flexibility: Maximize flexibility by allowing query time definitions of the relevance function $q(\cdot)$ and the distance threshold θ.

2: Interactive Refinement: While domain scientists have a crude idea on the appropriate θ for the task at hand, the optimal one might often be reached through a series of trials. This operation is similar in nature to finding the optimal "zoom" level in Google Maps. In



g1
g2
g3
g4
g5
A
B
C
D
E
$\overrightarrow{\hat{\pi}(c_2)}$=[0.6, 1]
centroid=g3
radius=1
diameter=2
c1
$\overrightarrow{\hat{\pi}(c_1)}$=[0.6, 1]
centroid=g3
radius=3
diameter=5
c2
c3
$\overrightarrow{\hat{\pi}(c_3)}$=[0.4, 1]
centroid=g5
radius=1
diameter=1
c4
$\overrightarrow{\hat{\pi}(c_4)}$=[0.6, 1]
centroid=g4
radius=1
diameter=1
$\overrightarrow{g_2}$=[0.3]
$\overrightarrow{\hat{\pi}(g_2)}$=[0.4, 0.8]
$\overrightarrow{g_1}$=[0.4]
$\overrightarrow{\hat{\pi}(g_1)}$=[0.4, 1]
$\overrightarrow{g_5}$=[0.6]
$\overrightarrow{\hat{\pi}(g_5)}$=[0.4, 0.8]
$\overrightarrow{g_3}$=[0.7]
$\overrightarrow{\hat{\pi}(g_3)}$=[0.6, 1]
$\overrightarrow{g_4}$=[0.5]
$\overrightarrow{\hat{\pi}(g_4)}$=[0.4, 1]

	g1	g2	g3	g4	g5
g1	0	3	2	2	1
g2		0	1	2	4
g3			0	1	3
g4				0	3

Figure 4: The NB-Tree built on the shown 5 graphs at $b = 2$. The $\hat{\pi}$-vectors, highlighted in red, are maintained only during query time. For simplicity, we assume all graphs are relevant.

other words, fine-tuning θ is an interactive procedure and a good index structure should adapt to those needs by performing refinements faster than a brand new query with a new relevance function.

To achieve the above outlined goals, the query processing algorithm runs in two phases: the *initialization* phase and the *search-and-update* phase. The initialization phase is performed only once as long as $q(\cdot)$ remains unchanged. For any subsequent refinements of θ, only the search-and-update phase needs to be repeated.

1. Initialization phase: VPs drive the initialization phase. First, VOs are used to compute a $\hat{\pi}$*-vector* for each relevant graph.

DEFINITION 6. $\hat{\pi}$-VECTOR. *The* $\hat{\pi}$*-vector of a graph* g *is a vector of* $\hat{\pi}_{\theta_i}(g)$ *computed at multiple distance thresholds* θ_i, *where* $\hat{\pi}_{\theta_i}(g) \geq \pi_{\theta_i}(g)$ *is an upper bound on the representative power of* g *computed using Theorem 5. Mathematically,*

$$\overrightarrow{\hat{\pi}(g)} = [\hat{\pi}_{\theta_1}(g), \cdots, \hat{\pi}_{\theta_n}(g)] \tag{13}$$

where $\theta_1 < \theta_2 < \cdots < \theta_{n-1} < \theta_n$ *are* n *different thresholds.*

The $\hat{\pi}$-vector allows us to compute an upper bound on the representative power for any given distance threshold θ. More specifically, a binary search can be performed on the indexed thresholds in the $\hat{\pi}$-vector to find the smallest $\theta_i \geq \theta$ and an upper bound $\hat{\pi}_{\theta_i}(g)$ can be derived to guide the searching process. We discuss which θs to index in Sec. 7.1.

Once the $\hat{\pi}$-vector is computed for all leaf-nodes (or graphs), the information is propagated upwards on NB-Tree by recursively computing and storing the ceiling of the $\hat{\pi}$-vectors of all children of a non-leaf node. Owing to this initialization procedure, an upper bound on the marginal gain in representative power of any graph g can be computed from any of its ancestor nodes. More specifically, for any non-leaf node n, and any set of graphs $\mathbb{S}$, the following property can be guaranteed.

$$\pi(\mathbb{S} \cup \{n\}) \geq \pi(\mathbb{S} \cup \{c\}) \text{ for any child } c \text{ of } n \tag{14}$$

Note that in the absence of interactive refinement, the $\hat{\pi}$-vector is not required since the input θ is already known; $\hat{\pi}_\theta(g)$ can be computed directly. However, if the threshold is refined to θ', then $\pi_{\theta'}(g)$ needs to be recomputed. More importantly, the information contained in any of the previously used θs cannot be leveraged. Thus, we pre-compute the representative powers at a series of distance thresholds in the initialization phase and use them as required

Algorithm 2 nextGraph($q(\cdot), \theta, k$)

```
1:  PQ ← priority queue containing $NB\text{-}tree.root$
2:  lb ← 0
3:  best ← **null**
4:  **while** PQ is not empty **do**
5:      entry ← PQ.dequeue()
6:      **if** entry.gain<lb **then**
7:          **return** best
8:      **if** entry is a graph **then**
9:          Compute $\hat{N}_\theta(entry)$ using VOs
10:         $N_\theta(entry) \leftarrow \{g \mid g \in \hat{N}_\theta(entry),\ d(entry, g) \le \theta\}$
11:         gain← $\pi(\mathbb{A} \cup entry) - \pi(\mathbb{A})$
12:         **if** gain> lb **then**
13:             best← entry
14:             lb← gain
15:     **else**
16:         **for** each child c ∈ entry **and** $q(entry) \ge \eta$ **do**
17:             gain← $\pi(\mathbb{A} \cup c) - \pi(\mathbb{A})$
18:             **if** gain> lb **then**
19:                 PQ.insert(gain,c)
20: **return** best
```

for refinements. In other words, the initialization phase is insulated from refinements of θ.

EXAMPLE 2. *Fig. 4 demonstrates how $\hat{\pi}$-vectors are constructed and propagated upwards in the tree through the ceiling operation. For simplicity, we assume all graphs are relevant. The $\hat{\pi}$-vector is computed on $\theta_1 = 1$ and $\theta_2 = 3$.*

2. Search and Update: The second phase can be divided into two subcomponents: search and update, which are performed iteratively k times.

2a. Search: Alg. 2 presents the pseudocode. First, a search is performed to identify the graph providing the highest marginal gain in representative power. The search starts from the root node (line 1). When a node is explored (line 5), each of its children is added to a priority queue where the nodes are ordered based on their marginal gains in representative power (lines 15-18). Since the marginal gain at any non-leaf node provides an upper bound on the marginal gains of all graphs in its subtree, they are used to prioritize the exploration of the unexplored nodes. The process is repeated iteratively till a graph is found whose actual marginal gain is higher than all candidates in the priority queue (lines 6-7). Only for such a graph, its θ-neighborhood is computed using graph edit distance to compute the exact marginal gain in the representative power (lines 8-14). As a result, expensive edit distance computations are reserved for only strong candidates and the computational cost is drastically reduced.

2b. Update: The cluster based bounds identified in Sec. 6.3 drive the update step. At the addition of a graph g to the answer set, the $\hat{\pi}$-vectors of nodes in NB-Tree are impacted. Now, instead of accurate re-computations of $\hat{\pi}$-vectors, a search is initiated from the root to identify clusters (or non-leaf nodes) that are within a distance of 2θ from g and consequently, affected (Theorem 6). For such clusters, using Theorems 7-8, the $\hat{\pi}$-vectors are updated. To identify the relevant graphs in a cluster, only graphs in its subtree need to be scanned. As noted earlier, Theorems 6-8 provide the benefit of computing a single upper bound for a group of structurally similar graphs that are located in the same cluster. Finally, the ceilings of the changed $\hat{\pi}$-vectors are propagated upwards, as in the initialization step, to reflect the "current state" of the search.

7.1 Choosing thresholds to index in the $\hat{\pi}$-vector

A bad choice of thresholds in the $\hat{\pi}$-vector would produce loose upper bounds which in turn would reduce the efficiency of the search-and-update procedure. Therefore, to make a more informed selection, the following factors need to be considered: domain knowledge, the history of previous queries, and the memory budget. Note that the selection of thresholds to index is an offline procedure and is decided by the index designer. The user who issues the query needs no prior information.

The number of thresholds to index can easily be decided based on the memory budget. For example, in an index with 1 million nodes, a $\hat{\pi}$-vector of size 10 would consume $\frac{4\times10\times10^6}{10^6}$ =40MB. For the more critical decision of choosing the thresholds, depending on the information and resources available, one of the following schemes can be adopted.

1. Query log: If the distribution of distance thresholds on previous top-k queries is available, then the thresholds to index can be sampled (without replacement) from that distribution.

2. No information is available: When no prior information is available, the distribution of $\pi(g)$ across θ can be computed from a small sample of graphs in the database. Next, the thresholds to index can be chosen proportional to the slopes in the distribution. More specifically, a higher number of thresholds should be indexed for the θ intervals where the slope is high since such intervals indicate regions where the upper bounds vary steeply even for a small difference in θ's.

8. EXPERIMENTS

This section summarizes experimental results that examine the quality and the scalability of our approach:

1. Information Density: The proposed model has a higher information density than DisC [9] and a diversity-based model [19].

2: Scalability: By indexing θ-neighborhoods of graphs instead of their nearest neighbors, NB-Index ensures scalability in answering top-k representative queries on graph databases.

8.1 Datasets

We use three different graph datasets to benchmark our techniques. Table. 3 summarizes the various properties of the datasets.

Table 3: Graph datasets used for evaluation

Dataset	Avg. # of nodes	Avg. # of edges	# of graphs
DUD	26	28	128332
DBLP	55	263	11362
Amazon	29	189	9120

1. DUD: The DUD [10] dataset contains 128,332 molecules that were assayed against 10 different protein targets. Consequently, each molecule is tagged with a 10-dimensional feature vector that represents its binding affinity to each of the targets. In the graph representation of a molecule, each vertex corresponds to an atom, and the edges represent atom-atom bonds. Our goal is to identify molecular structures that are compatible with the structure of the protein targets. The DUD dataset allows us to assess the natural correlations that exist between the feature and the structural space, and their resultant effects on top-k queries.

2. DBLP: In the DBLP network [1], each node corresponds to an author, and two authors are connected if they have at least one paper together. Furthermore, each author is tagged with the community he/she belongs too. For our evaluation, the goal is to understand if the most active groups collaborate within the community or span across multiple communities. For that purpose, we replace the author in each node label with the person's community. Next, we extract the complete 2-hop neighborhood subgraph around each node to construct our graph database. Finally, the combined activity level of each graph, denoting a collaboration group, is characterized with a one-dimensional feature vector.

Table 4: Compression ratios (CR) $|N_\theta(\mathbb{A})|/|\mathbb{A}|$ and $\pi(\mathbb{A})$ achieved by REP and DIV [19] at various ks. The last row indicates the compression ratio of DisC along with its answer set size in parentheses.

Property	DUD						DBLP						Amazon					
	REP		DIV(θ)		DIV(2θ)		REP		DIV(θ)		DIV(2θ)		REP		DIV(θ)		DIV(2θ)	
	CR	$\pi(\mathbb{A})$	CR	$\pi(\mathbb{A})$	CR	$\pi(\mathbb{A})$	CR	$\pi(\mathbb{A})$	CR	$\pi(\mathbb{A})$	CR	$\pi(\mathbb{A})$	CR	$\pi(\mathbb{A})$	CR	$\pi(\mathbb{A})$	CR	$\pi(\mathbb{A})$
$k = 10$	115	0.23	31	0.06	28.8	0.06	45.4	0.16	23.1	0.08	19.1	0.07	84.3	0.17	37.8	0.21	27	0.12
$k = 25$	80	0.40	22.9	0.11	16.4	0.08	25	0.22	12	0.11	10.7	0.09	41.9	0.46	19.6	0.22	15.9	0.17
$k = 50$	58	0.58	17	0.17	9.9	0.09	15.3	0.27	7.5	0.13	6.4	0.11	23.7	0.52	12	0.26	9.3	0.2
$k = 100$	35	0.70	10.1	0.2	4.7	0.09	11.6	0.41	4.7	0.17	4	0.14	15	0.66	6.9	0.3	5.4	0.24
DisC	2.77 (1782)						1.78 (1590)						2.5(912)					

3. Amazon: The Amazon network [1] is structured in a manner similar to DBLP. Each node corresponds to an item, and two items are connected if they are frequently co-purchased. In addition, each item is classified under a category. Our goal is to understand if cross-category coupling exists among items that are popular. Thus, similar to the processing in the DBLP dataset, we replace the node labels with the category of the item and then extract its 2-hop neighborhood subgraph. As in the DBLP dataset, a feature vector is used to characterize the popularity of each co-purchase graph.

8.2 Experimental setup

Our algorithms are implemented in Java 1.6.0 and benchmarked on an Intel i7 2.67GHz quad core processor PC with 12GB of main memory running Ubuntu 12.10. We refer to our proposed model as *REP*, and the diversity model in [19] is termed *DIV*.

1. Quality Evaluation: To evaluate the quality of the answer set produced REP, we compare its representative power with the full DisC answer set, and the top-k answer sets of DIV.

2. Scalability Evaluation: We benchmark the performance of NB-Index against the 'Grey-Greedy-DisC(Pruned)' algorithm of DisC [9], the 'div-cut' approach in DIV [19] and C-tree [12]. For DisC, we stop the computation as soon as it attains a size of k. For DIV, we use C-Tree to compute the 'diversity-graph', which is subsequently used by the 'div-cut' algorithm, to speed up processing.

Due to the extremely slow running times of DisC, DIV and C-tree on the entire DUD database, we select a random sample of $25,000$ graphs. Nonetheless, for scalability experiments against dataset size, we use the entire DUD database. For DBLP and Amazon, we use the entire dataset for all experiments.

8.2.1 Query Arguments

1. $q(\cdot)$: One primary goal in REP is to support query time definition of relevant graphs. To model this requirement, in DUD, we select a random subset of d dimensions where $1 \leq d \leq 10$, and the feature-space score is quantified as $\sum_1^d \frac{g_i}{d}$. Both the DBLP and Amazon datasets are characterized using 1D feature vectors, which act as their feature space scores. A graph is considered relevant if its feature-space score falls within the first quartile.

2. θ: Our goal is to select a θ that is realistic and yet poses a significant scalability challenge in answering top-k representative queries. Towards that goal, we study the cumulative distribution of the graph distances in each dataset (Figs. 5(a)-5(b)). While the distributions are similar for DUD and DBLP, distances between graphs in the Amazon dataset are much farther apart. Based on these observations, for DUD and DBLP we set $\theta = 10$ and for Amazon, $\theta = 75$. For DBLP and DUD, addition of a graph to the answer set impacts the $\pi(g)$ of any graph g within a distance of $2\theta = 20$; for Amazon, graphs within a distance of 150 are affected.

3. k: The default k is set to 10.

8.2.2 Parameters

1. $\hat{\pi}$-vector: The $\hat{\pi}$-vector is chosen based on the cumulative distance distributions in Figs. 5(a)-5(b). Given the high slope between $\theta = 10$ and $\theta = 40$ for both DUD and DBLP, we index 10 different distance thresholds at 5, 12, 16, 20, 25, 30, 35, 40, 75, 100. We index at θs beyond 40 just for the sake of completeness so that a query on any θ can be supported. For amazon, we adapt $\hat{\pi}$-vector using the same strategy. Owing to the high slope between $\theta = 50$ and $\theta = 200$, we index at 20, 60, 90, 120, 150, 180, 200, 300, 400, 500.

2. Number of VPs: The number of VPs is chosen based on Eqn. 11. The distance distributions in the three datasets (Figs. 5(c)-5(e)) are approximated as a Gaussian of their means and standard deviations. To limit the FPR below 5% at the realistic zone of $5 \leq \theta \leq 20$ for DUD and DBLP, and at $50 \leq \theta \leq 200$ for Amazon, we choose 100 VPs.

3. Branching Factor: The maximum branching factor in NB-Index is set to 40.

8.3 Quantitative analysis

In this section, we quantify the performance of the proposed techniques over the current state of the art.

8.3.1 Efficacy

One of our primary motivations in this work is to allow users to control the size of the answer set, so that a smaller set of exemplars can be identified and investigated in further detail. Towards that goal, we compare the *compression ratios (CR)* of REP, with the ones achieved by DisC and DIV. The CR of an answer set $\mathbb{A}$ is defined as $\frac{|N_\theta(\mathbb{A})|}{|\mathbb{A}|}$, which is simply the ratio of the number of relevant objects represented to the answer set size. In this experiment, for DUD, we consistently use the same subset of feature vector dimensions since it is essential to capture the natural correlations between the feature and structural space, which in turn affects the CR.

Table 4 presents the results. The CRs in REP are significantly higher than both DisC and DIV. This result establishes the utility of operating in a budgeted setting as well as the advantage of a representative-aware model over a diversity-aware model. For DIV, we compute the result at two different settings. In the first setting, DIV(θ), we use the original model in [19], which ensures all answer set objects are at least θ apart. However, this model assumes that the scores of graphs are mutually independent. In reality, the representative powers of the graphs are dependent on each other. Therefore, to also study the performance when the independence assumption is followed, we enforce the stricter condition of $\forall g_1, g_2,\ d(g_1, g_2) > 2\theta$. The produced answer set is denoted as DIV(2θ). As can be seen, DIV(2θ) further lowers the CRs since the stricter condition to guarantee score independence rules out many representative graphs from inclusion in the answer set.

To further analyze the representation of relevant objects, we also compute the growth of $\pi(\mathbb{A})$ with k. As shown in Table 4, REP covers close to a quarter of the relevant objects using just 10 exemplars. As k increases beyond 100, it is futile to try to represent relevant objects using exemplars, since either the remaining non-

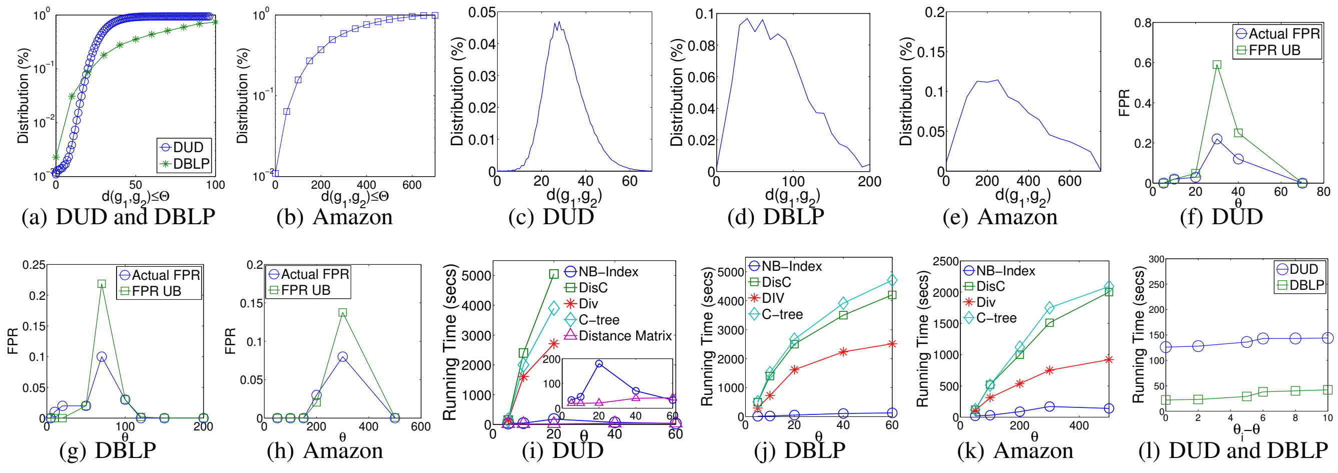


Figure 5: (a-b). The cumulative distance distributions. (c-e) The distance distributions. (f-h) Observed FPRs, and FPR Upper Bounds (FPR UB) with θ. (i-k) Growth rates of query times against θ and (l) distance to the closest indexed threshold θ_i.

represented objects are outliers or their θ-neighborhoods have large overlaps with already represented objects. Consistent with the results observed while analyzing the CRs, DIV(θ) and DIV(2θ) have up to 6 times lower representative powers due to relying on an indirect maximization approach. Overall, DIV is unable to model the semantics of representativeness, and consequently, the quality of the answer set is compromised.

A detailed theoretical analysis on the optimal number of VPs is performed in Sec 6.2.1 to predict an upper bound on the False Positive Rate (FPR). We now verify how well the empirical results conform to the theoretical analysis. Figs 5(f)-5(h) present the results. The FPR is highest in the DUD dataset, due to the low standard deviation of the distances. Intuitively, if all graphs are similar to each other and clustered together, the performance of VPs deteriorates. For this reason, on DBLP and Amazon, which are not as densely clustered, the FPR is much lower. At low θs in DBLP and DUD, the FPR Upper Bound (≈ 0.01) is slightly lower than the observed FPR. This results from the fact that the distance distribution in DBLP and Amazon slightly deviates from a normal distribution.

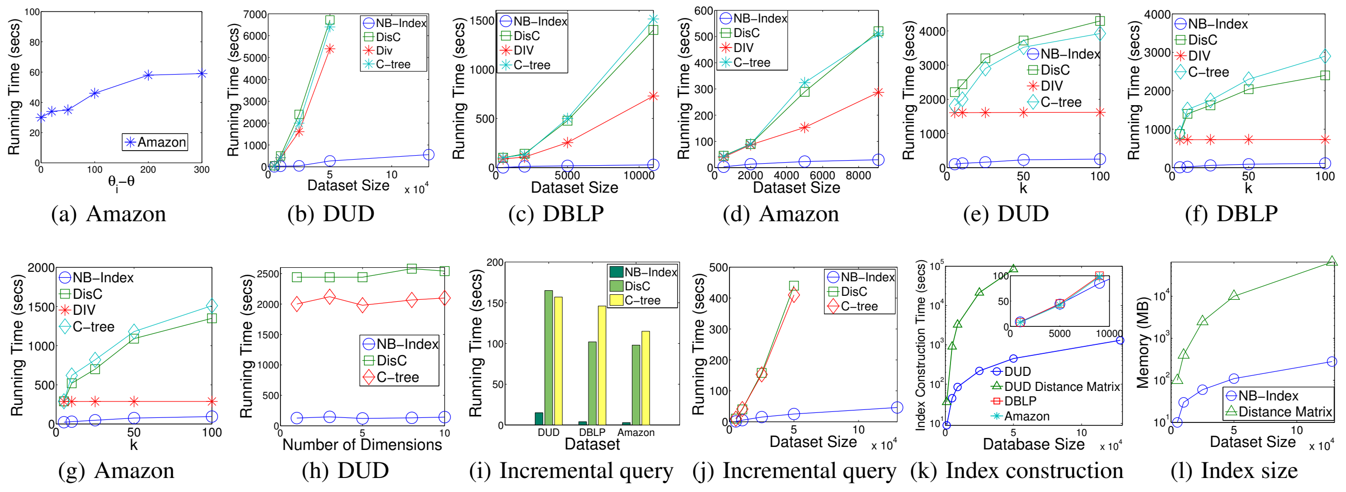
8.3.2 Scalability

First, we evaluate the performance of NB-Index against the distance threshold θ. Figs. 5(i)-5(k) show the growth rates of query times as θ is varied. NB-Index is up to two orders of magnitude faster than existing techniques. Due to the enormous running times of DisC, C-tree and DIV on DUD, we do not show their results beyond $\theta > 20$. The growth rates of existing techniques are similar since they all index nearest neighbor queries through triangular inequality; while DisC and C-tree propose their own index structures, in DIV, we use C-tree as the underlying graph indexing technique to construct the 'diversity-graph', on which DIV performs further processing. A bulk of the computation time is spent in initializing the θ-neighborhoods of relevant graphs. This operation needs to be performed online since both θ and the relevance function are query parameters. DIV assumes the representative powers of graphs to be mutually independent, and thus does not re-compute them after each answer set update. Consequently, at the cost of quality, it is faster than both DisC and C-tree. In NB-Index, majority of the computations happen in the feature space through VPs. As already shown in Figs. 5(f)-5(h), the FPRs in VPs are low. Furthermore, beyond VPs, the cluster based bounds outlined in Theorems 6-8 index the θ-neighborhoods in structural space itself and allows batch updates of representative power.

To further evaluate our performance, we also compare its running time to the scenario where the distance matrix of the entire graph database is pre-computed. Due to the fast running times of both the approaches, the inset in Fig. 5(i) presents a zoomed-in view to better compare the two approaches. Practically, storing the entire distance matrix is not feasible in large databases since it incurs a huge storage cost. Nonetheless, we benchmark our performance against the best-case scenario from the running time perspective. Except at $\theta = 20$, the performance of NB-Index is 1.5 to 2 times higher than storing the distance matrix. Another important property that is emphasized in the inset of Fig. 5(i) is that NB-Index is most efficient at the two extreme ends of the θ range. When θ is large, Theorems 7 and 8 are more effective in pruning the search space. On the other hand, Theorem 6 is more efficient at low θs. Thus, mid-range θs present the most challenging scenario, which results in a bell shaped growth rate of the query time.

Next, we analyze the performance deterioration as the difference between the user-provided distance threshold θ and the closest higher indexed threshold θ_i in NB-Tree increases. This analysis shows how sparsity in the $\hat{\pi}$-vector affects the performance of NB-Index. Note that a non-indexed θ only affects the performance of the structural bounds. The performance of VOs remain unaffected. Figs. 5(l), 6(a) show the growth rates of running times. Even when the difference is as high as 10 for DUD and DBLP, and 300 for Amazon, NB-Index consumes less than 25 seconds of additional time. As evident from Figs. 5(a)-5(b), the slopes are extremely steep between $10 \leq \theta \leq 20$ for DUD and DBLP and $50 \leq \theta \leq 200$ for Amazon. Thus, upper bounding π_{10} with π_{20} (in DUD and DBLP) provides a particularly tough setting. However, even in such cases, the performance of NB-Index is significantly better thanks to the efficiency of the VOs and the fast upper bounds derived using Theorems 7 and 8.

Figs. 6(b)-6(d) demonstrate the growth rate of running time against dataset size. NB-Index scales significantly better than all three existing techniques and achieves a performance that is more than an order of magnitude faster. This result stems from indexing the θ-neighborhoods of graphs, which negates the need to perform $O(n^2)$ nearest neighbor queries. Figs. 6(e)-6(g) further establish the superiority of NB-Index as k is varied from 5 to 100. The growth rate of



(a) Amazon (b) DUD (c) DBLP (d) Amazon (e) DUD (f) DBLP

(g) Amazon (h) DUD (i) Incremental query (j) Incremental query (k) Index construction (l) Index size

Figure 6: Growth rate of query times with (a) difference between user-provided distance threshold θ and closest indexed threshold θ_i, (b-d) dataset size, (e-g) k and (h) number of dimensions. (i) Running times for incremental zoom-in and zoom-out. (j) Growth rate of incremental zoom-in and zoom-out query time with dataset size. Growth rate of (k) index construction time and (l) indexing memory footprint against dataset size.

query time with k is much lower for NB-Index due to the same reason of indexing the θ-neighborhoods of graphs. The running time of DIV is almost constant across k since it assumes the representative powers of graphs to be mutually independent. Thus, after the diversity-graph is constructed, all remaining computations occur in the feature space, which is minuscule when compared to the cost of constructing the diversity-graph.

Fig. 6(h) investigates the performance as the number of dimensions in feature vectors is varied between 1 and 10 in the DUD dataset. A feature vector of dimension d is constructed by randomly choosing a subset of d dimensions from the overall 10. The query time is almost identical across different numbers of dimensions for all three techniques since the cost of operating in the feature space is negligible compared to the cost of computing graph edit distances while updating θ-neighborhoods. The minor variation in running times results from the correlations between feature and structural space where a higher correlation results in faster query times.

Fig. 6(i) compares query times in the interactive θ refinement scenario. In this experiment, first, a query is performed on the default θ outlined in Sec. 8.2.1. Next, a new θ is selected, which is either 10% smaller or larger, and the answer set is re-computed. This process is repeated 20 times and the average computation time is reported. NB-Index tackles refinements of θs within 10 seconds across all three datasets. On the other hand, DisC and C-tree takes up to 160 seconds to adapt to a new θ. To further study the response times in an interactive setting, we analyze its growth rate against dataset size in Fig. 6(j). Similar to the previous results, NB-Index is more than an order of magnitude faster.

Finally, we analyze the computation and storage costs of the proposed index structure. Fig. 6(k) presents the growth rate of the index construction time against dataset size. While the main plot compares the construction time of NB-Index with that of computing the entire distance matrix in the DUD database, the inset shows the construction times across all three datasets. In DUD, which contains a total of $\approx 130,000$ graphs, the index construction completes within 20 minutes. Compared to the cost of computing the entire distance matrix, we are more than two orders of magnitude faster. This efficiency is achieved through VP-based pruning in pivot and radius computations where actual edit distances are computed for less than 1% of the candidate pairs. The scalability of NB-Index also extends to its storage cost. As expected from the theoretical analysis, a linear growth rate is observed in Fig. 6(l) and less than 300MB is required to store the index for the entire DUD dataset. The reported result includes the memory consumption from storing the $\hat{\pi}$-vectors, which are computed at query-time.

8.4 Qualitative analysis

In this section, we highlight the utility of a representative model over the traditional top-k framework. Towards that goal, we perform a top-k query and a top-k representative query using the same scoring function and compare the two answer sets. To construct the database, we extract molecules from the DUD repository that are active against the enzyme Acetylcholine ezterase (AChE). Extracted molecules are characterized based on their binding affinity to AChE, which is one of the primary targets for drugs treating Alzheimer's disease [4]. A molecule is considered relevant, if its binding affinity, denoted as BA, is within the top quartile.

Fig. 8.4 shows the two top-5 answer sets. As illustrated in Fig. 8.4 using color coding, all molecules in the traditional answer set are structurally similar to each other. Although they exhibit high binding affinity toward AChE, the traditional answer set represents just one structural family of high scoring molecules. On the other hand, the representative answer set contains molecules that are all structurally diverse, high scoring, and representative of other high-scoring molecules in the dataset. The efficacy of the proposed model is well highlighted in the shown result where the entire class of molecules in the traditional answer set is summarized by the top molecule in the representative answer set. In the context of drug discovery, the traditional answer set provides only one class of molecules that can be further optimized to develop drugs. Whereas, the representative answer set provides five different classes of molecules, each of which can be studied further for drug discovery.

9. RELATED WORK

A detailed comparison with DisC [9] and [19] has already been performed. A closely related query, called top-k typicality queries

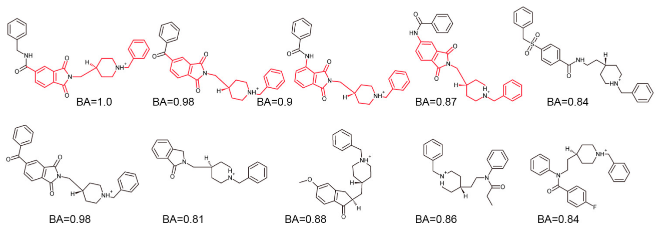


Figure 7: Answer sets computed using traditional top-k query (first row) and top-k representative query (second row). In the top row, the subgraph in red occurs in four of the five molecules, indicating the core structure of a single class of molecules.

is formulated in [13, 14]. An object o is considered "typical" if its distances to other objects in the database are small. While the typicality score is conceptually similar to the proposed idea of representative power, the paper does not explore the coverage maximization aspect. Consequently, the typicality scores of two objects are independent and there is no penalty when highly typical objects from the same cluster are included in the answer set. As a result, the problem in [14] is not NP-hard, whereas ours is. In addition, instead of high-dimensional objects, we focus on graphs where computing the distance itself is NP-hard. Consequently, the scalability challenges faced are unique to our problem.

The idea of representativeness has been explored in the context of skyline queries [16]. An object in a database is a skyline point if it is not *dominated* by any of the database objects. The representativeness of a skyline point is the number of point it dominates. The goal in this work is to identify the k skyline points that dominates the maximum number of database points. Thus, while in our work, an object represents another object if their distance is within a threshold, in [16], an object o_1 represents object o_2 if o_1 dominates o_2. Consequently, the modeling requirements and the proposed indexing techniques are different. In addition, we focus on graphs whereas [16] focuses on high-dimensional points.

Beyond diversification of results, indexing graphs for similarity queries [12, 26, 28, 30] is related to our problem. As shown in our experimental results, indexing graphs for nearest neighbor queries is not enough to scale top-k representative queries. Consequently, our work takes a more direct approach by indexing the θ-neighborhoods of graphs, which lies at the core of updating graph representative powers. Embedding graphs into a feature space has been explored by Zou et. al. [31]. Their goal is to index shortest part queries between nodes by embedding nodes in a feature space. In our work, instead of a single large graph, we have multiple graphs each of which is embedded in a feature space. Additionally, instead of indexing shortest paths, our goal is to index their θ-neighborhoods, which is a function of graph edit distance. Due to this basic difference in the entities being embedded and their associated indexing goals, the technique, the bounds derived from the techniques, and the error guarantees do not transfer.

10. CONCLUSION

In this work, we formulated the problem of top-k representative queries on graph databases. By allowing online definitions of object relevance and answer set budget, the proposed formulation provides a higher degree of flexibility to users. We showed that the problem is NP-hard and submodular. Based on this result, we designed a greedy constant factor approximation of the optimal answer set. To achieve scalability, we designed an index structure called NB-Index that indexes the θ-neighborhoods of database graphs by employing a novel combination of Lipschitz embedding and agglomerative clustering. NB-Index not only facilitates fast answering of queries, but also allows interactive refinement of θ to reach the optimal zoom level. Empirical evaluations on real graph datasets demonstrate significantly higher information density and a performance improvement of up to two orders of magnitude over the state of the art.

Acknowledgements: The work was supported by NSF grant IIS-1219254. We also thank Google for travel grant PO 55022251.

11. REFERENCES

[1] SNAP, http://snap.stanford.edu/.
[2] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, 2009.
[3] A. Angel and N. Koudas. Efficient diversity-aware search. In *SIGMOD Conference*, pages 781–792, 2011.
[4] C. G. Ballard. Advances in the treatment of alzheimer's disease: benefits of dual cholinesterase inhibition. *Eur Neurol*, 47(1):64–70, 2002.
[5] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.
[6] G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri. Efficient diversification of web search results. *PVLDB*, 4, 2011.
[7] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt. Recommendations to boost content spread in social networks. In *WWW*, pages 529–538, 2012.
[8] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan. Identifying bug signatures using discriminative graph mining. In *Symposium on software testing and analysis*, 2009.
[9] M. Drosou and E. Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *PVLDB*, 6(1):13–24, 2012.
[10] DUD. http://dud.docking.org/r2/.
[11] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45:634–652, July 1998.
[12] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, 2006.
[13] M. Hua, J. Pei, A. Fu, X. Lin, and H.-F. Leung. Top-k typicality queries and efficient query answering methods on large databases. *The VLDB Journal*, 18(3):809–835, 2009.
[14] M. Hua, J. Pei, A. W. C. Fu, X. Lin, and H. fung Leung. Efficiently answering top-k typicality queries on large databases. In *VLDB*, pages 890–901, 2007.
[15] R. Li and J. X. Yu. Scalable diversified ranking on graphs. In *ICDM*, 2011.
[16] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007.
[17] K. Macropol and A. Singh. Content-based modeling and prediction of information dissemination. In *ASONAM*, 2011.
[18] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1):265–294, 1978.
[19] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *PVLDB*, 5(11):1124–1135, 2012.
[20] S. Ranu, B. T. Calhoun, A. K. Singh, and S. J. Swamidass. Probabilistic substructure mining from small-molecule screens. *Molecular Informatics*, 30(9):809–815, 2011.
[21] S. Ranu, M. Hoang, and A. Singh. Mining discriminative subgraphs from global-state networks. In *SIGKDD*, pages 509–517, 2013.
[22] S. Ranu and A. K. Singh. Answering top-k queries over a mixture of attractive and repulsive dimensions. *PVLDB*, 5(3):169–180, 2011.
[23] S. Ranu and A. K. Singh. Indexing and mining topological patterns for drug discovery. In *EDBT*, pages 562–565, 2012.
[24] H. Tong, J. He, Z. Wen, R. Konuru, and C.-Y. Lin. Diversified ranking on large graphs: an optimization viewpoint. In *SIGKDD*, 2011.
[25] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, 2008.
[26] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD*, 2005.
[27] C. Yu, L. Lakshmanan, and S. A. Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, 2009.
[28] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1), 2009.
[29] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search - The Metric Space Approach*. Springer-Verlag, 2006.
[30] Y. Zhu, L. Qin, J. X. Yu, and H. Cheng. Finding top-k similar graphs in graph databases. In *EDBT*, pages 456–467, 2012.
[31] L. Zou, L. Chen, and M. T. Özsu. Distance-join: Pattern match query in a large graph database. *PVLDB*, 2(1):886–897, 2009.