Spectral Decomposition for Optimal Graph Index Prediction

Liyan Song[1], Yun Peng[1], Byron Choi[1], Jianliang Xu[1], and Bingsheng He[2]

[1]Department of Computer Science, Hong Kong Baptist University, Hong Kong
{lysong,ypeng,bchoi,xujl}@comp.hkbu.edu.hk
[2]School of Computer Engineering, Nanyang Technological University, Singapore
{bshe}@ntu.edu.sg

Abstract. Recently, there has been ample of research on indexing for structural graph queries. However, as verified by our experiments with a large number of random graphs and scale-free graphs, the performances of indexes of graph queries may vary greatly. Unfortunately, the structures of graph indexes are too often complex and ad-hoc; and deriving an accurate performance model appears a daunting task. As a result, database practitioners may encounter difficulties in choosing the optimal index for their data graphs. In this paper, we address this problem by a spectral decomposition for predicting relative performances of graph indexes. Specifically, given a graph, we compute its spectrum. We propose a similarity function to compare the spectrums of graphs. We adopt a classification algorithm to build a model and a voting algorithm for the prediction of the optimal index. Our empirical studies on a large number of random graphs and scale-free graphs and four structurally distinguishable indexes demonstrate that our spectral decomposition is robust and almost always exhibits accuracies higher than 70%.

1 Introduction

Due to the flexibility of graph model, it has a wide range of recent applications, such as biological databases, social networks and XML. To optimize query processing on graph data, many indexing techniques for graph queries have been recently proposed. Unfortunately, graph data are often heterogeneous and the structures of indexes are complex and often ad-hoc. As revealed by our experiments, the performances of graph indexes on graphs may vary greatly. This leads to a natural question for database practitioners: *What is the index that is the most efficient for a given graph?*

When compared to the relational counterparts, the structures of many graph indexes are far more complex. This causes a few unique problems. Firstly, the construction of graph indexes is sometimes time-consuming. For example, we tested via experiments on our commodity computer that given a random graph with a modest size (with $\sim$3,000 vertices and a density 0.02), the construction time for a graph index, namely `2-hop labeling` [14], is already 8.3 seconds. (The background details of the indexes discussed are presented in Appendix.) While some other graph indexes can be constructed within a second, one may know the most time-efficient index only after *all* indexes are constructed and benchmarked. Furthermore, the performances depend on not only

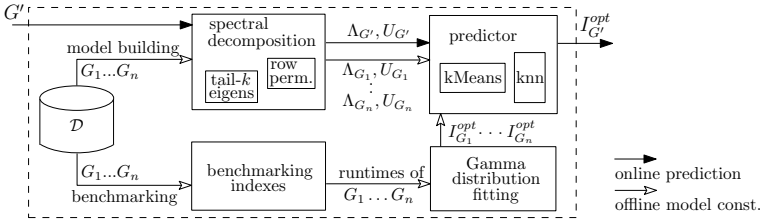


Fig. 1. Overview of our spectral decomposition prediction framework

algorithms but also the implementation quality and details, as there has not been well-received (commerical) implementation yet. Secondly, graph indexes are sometimes several times larger than the graph itself. Following up the example, the size of `2-hop labeling` is 19 times larger than the graph and `Interval labeling` [1] is almost as large as the graph itself. Hence, it is not space-efficient to use and maintain multiple indexes simultaneously on the entire graph. In all, it is desirable to predict the optimal index and construct the optimal index alone for efficient query processing.

As a proof of concept, we focus on *reachability queries* — "given two vertices, is one vertex reachable from another?" — which is a fundamental query of graphs. However, our proposed technique is not dependent to any specific type of graph query.

In this paper, we apply data mining techniques to predict the relative performances of graph indexes. One of the core problems is to extract important features (or characteristics) from data graphs. While there have been a large variety of features from graph theories [10], it has not been clear what are the features that are the most relevant to index performances. Therefore, we propose to apply a general technique from spectral graph theories [5], namely spectral decomposition, to solve our problem. To the best of our knowledge, this is the first work that investigates the relationships between graph spectrums and index performances. In general, graph spectrums have known to be *characteristics of graphs* and have known to be related to many important graph properties. Another advantage of spectrums is that they have been supported by industrial-strength softwares, not to mention the use of advanced optimizations on determining spectrums.

The second core problem is to represent the performance of a graph index. Our preliminary experiments show that the runtimes of 1,000 random queries on an index, even on a same graph, can often exhibit large variances. For example, we have tried 1,000 random queries on each of the 8,000 random graphs and each data graph has been indexed by `2-hop` and `Prime labeling` [13]. The mean and standard deviation of `2-hop` are 14.1 seconds and 4.2, respectively. Those of `Prime labeling` are 11.6 seconds and 59.7, respectively. Furthermore, the runtimes are sometimes skewed and have a long tail at large runtimes. (In later section, we visualize some runtime distributions in Figure 2.) We observe a similar phenomenon from our collection of scale-free graphs. A possible explanation is that a graph may contain many different sub-structures and the indexes themselves are complex structures as well. These lead to a wide range of runtimes. While average runtimes are often used to quantify index performances, it is desirable to propose a flexible metric for the performances.

In this paper, we fit the runtimes of queries into a distribution. From our experiments on estimating the parameters of distributions, the goodness of fit of Gamma distribution is always the best. By comparing distributions of runtimes, we can obtain a more robust and flexible way to compare performances. While we may apply the research on

the distribution for further analysis, in this paper, we apply inverse cumulative distribution function to estimate the time when y% of queries finish. Depending on users' applications, they may specify a value y% to express how reasonable are long query runtimes. For instance, some Internet connection providers (*e.g.* hotels and cafe) charge their users by connection times and long query runtimes can be undesirable. To cater for the needs of those users, data practitioners may choose the index that is optimal at 98%, instead choosing the one that has the optimal average runtime.

Contributions. To our knowledge, this is the first investigation on the spectrums of graphs in relation to index performances. We summarize the contributions of this work below. The overview of our contributions is presented in Figure 1.

- Given a graph G_i in a database $\mathcal{D}$, we propose a spectral decomposer to determine G_i's Laplacian matrix and a set of eigenvalues Λ_{G_i} and eigenvectors U_{G_i} . The eigenvalues and eigenvectors are transformed into a unified representation for comparisons.
- We propose a spectral similarity function between two graphs .
- We propose to fit the runtimes of an index on a given graph into Gamma distribution using a distribution fitter. Users may then tune their desired notion of optimal index.
- We adopt the k-Means algorithm and k-nearest neighbor with a voting method to predict the optimal index $I^{opt}_{G'}$ of a given graph G'.
- We have conducted experiments with a large number of random graphs and scale-free graphs. The results show that our proposed technique can achieve accuracies almost always higher 70% and very often higher than 80%.

The rest of the paper is organized as follows. Section 2 presents the backgrounds of graph spectral decomposition. We present our problem statement in Section 3. Section 4 presents the definition of optimal index. A uniform spectral representation of graphs and a similarity function between graphs are proposed in Section 5. Our prediction method is detailed in Section 6. We report our experimental evaluation in Section 7. Section 8 discusses the related work and Section 9 concludes this paper. Background details of the indexes discussed are presented in Appendix.

2 Backgrounds on Graph Spectral Decomposition

In this section, we provide the background of graph spectrums. Graph spectrums have been widely used to study many interesting properties of graphs, such as spectral partitioning and expansion [9], cut problem [8], graph drawing [15].

Graph spectrum is often defined with Laplacian matrix. Laplacian matrix L of a directed or undirected graph $G = (V, E)$ is defined as $L = D - A$, where D is the degree matrix of G and A is the adjacency matrix of G. More specifically, $A_{i,j} = 1$ if $(v_i, v_j) \in E$; and $A_{i,j} = 0$ otherwise, where i and i are the *ID*'s of the vertices v_i and v_j. The degree matrix is a diagonal matrix and $D_{i,i}$ is the outdegree of v_i

The Laplacian matrix L of G can be eigendecomposed as $L = U\Lambda U^{-1}$, where Λ is a diagonal matrix of eigenvalues of L, and U is a matrix of the corresponding eigenvectors. Specifically, let $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$, where $n = |V|$, denote the eigenvalues; $X_1, X_2, ..., X_n$ denote the corresponding eigenvectors, where $\Lambda_{i,i} = \lambda_i$; and the i-th column of U is X_i. We may call U *eigenvector matrix* and use Λ to refer to eigenvalues. We may use a subscript in U_G and Λ_G to denote the U and Λ of graph G when needed.

Eigenvalues Λ are called *spectrums* in spectral graph theories. Since eigenvectors U are also known to be closely related to characteristics of vertices, we include eigenvectors in our algorithm. We use Λ and U of the underlying undirected graphs of the data graphs, as they capture the graphs' structures and their properties are well-studied [5].

3 Problem Formulation

In this section, we formulate the optimal graph index prediction problem based on graph eigenvalues and eigenvectors.

We assume a graph database $\mathcal{D}$ containing a large number of directed graphs $\{G_1, G_2, ..., G_m\}$. The reachability query on the graphs is formally defined as follows.

Definition 3.1. *Given a directed graph $G = (V, E)$, u, $v \in G$, v is reachable from u, denoted as $u \rightsquigarrow v$ = true, if and only if there is a path from u to v in G.*

As discussed, various types of indexes are available to support reachability query on graphs in $\mathcal{D}$. However, it is desirable to predict the optimal one without building and benchmarking all indexes available. This problem can be described as follows.

Problem statement. *Given a set of indexes $\mathcal{I} = \{I_1, I_2, ..., I_n\}$ and a graph database $\mathcal{D}$, we want to build a predictive model M using the eigenvalues and eigenvectors of graphs in $\mathcal{D}$, in order to efficiently determine the optimal index I_G^{opt} for a graph $G \notin \mathcal{D}$.*

There are two main sub-problems in the problem statement.

(P1) How to represent the performance of an index on a graph?

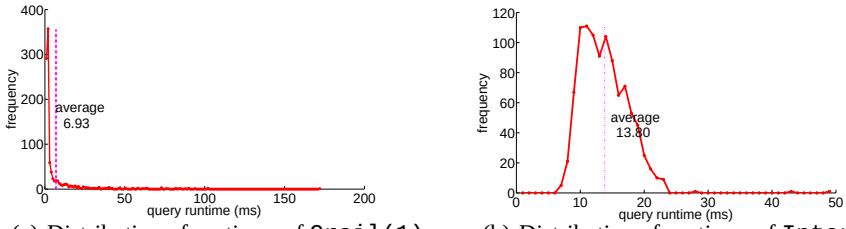
As motivated in Section 1, the query runtimes of a particular index on a particular graph may deviate from the average runtime. Therefore, in Section 4, we investigate a flexible notion of *optimal index* in expressing the desirable runtimes.

(P2) How to compare spectrums of graphs?

As motivated in Section 1, we propose to represent a graph G with eigenvalues Λ_G and eigenvectors U_G. A similarity function between the spectral graph representation is needed. In addition, the number of eigenvalues and the dimension of the eigenvectors of a graph G is the number of vertices of G, which are not uniform in a graph database. They are transformed into a uniform representation for comparisons. Finally, while the eigenvalues Λ_G are invariants of G, the row vectors of the eigenvector matrix U_G are dependent to the permutation of vertex IDs of G. Therefore, there is a row permutation problem in comparing U's of graphs.

4 Performance Metric

We define the performance of an index using the query times of 1,000 random queries, as the query workloads are often not known when an index is chosen. To study the query performances, we plot the query time distribution, where x-axis is the query time; y-axis is the number of queries finished at x. For example, Figure 2(a) and Figure 2(b) show the query runtime distributions of two indexes on a random data graph. We demonstrate that average runtime alone may lack the flexibility to describe a desired notion of performances. For example, Figure 2(a) shows `Grail(1)` [20] has the smallest average time but it has a long tail. In comparison, the rutntimes of `Interval` exhibit a relatively small variance, while `Interval` has a relatively large average time.



(a) Distribution of runtimes of `Grail(1)` (b) Distribution of runtimes of `Interval`

Fig. 2. Some distributions of the runtimes of 1000 random queries on a random graph

Therefore, `Interval` could be a better choice in applications where long query times are not acceptable or commercially unfavorable.

Once the query runtimes are presented as a distribution, we fit the distribution with some well-studied distributions, *e.g.*, Normal distribution, Poisson distribution and Gamma distribution. To measure the goodness of fit, we adopt the L_2-norm to the estimated distribution and the real distribution. From our experiments on a large number of random and scale-free graphs, we observe that Gamma distribution almost always yields the best fitting. A possible reason is that Gamma distribution is often used to model the waiting time and query runtime may be considered as the waiting time until queries finish. (The detailed experiment on the fitting is presented in Section 7.) Moreover, the parameters of Gamma distribution can be efficiently estimated. Therefore, we use Gamma distribution to represent the query runtime.

While the optimal index is intuitively the most efficient on 1,000 random queries, we define the optimal index to be the one with the smallest estimated runtime w. r. t. the user-dened parameter y% (*e.g.*, 98%). Once the parameters of Gamma distribution are estimated, the notion of optimal index can be tuned and determined mathematically by adjusting y.

Definition 4.2. *Given a set of indexes $\mathcal{I} = \{I_1, I_2,..., I_n\}$, a graph G, and a set of random queries Q, the* optimal index *in $\mathcal{I}$ of G is the index that finishes y% of queries of Q in the shortest time.*

An application of Definition 4.2 is that database practitioners may check the robustness of an index. For instance, given a graph database, we may use the estimated Gamma parameters to construct prediction models for a few y values, *e.g.*, 75%, 85% and 95%, respectively, *without rerunning the benchmarking queries*. An index can be considered robust if it is optimal for all those y values, instead of a specific y value.

5 Spectral Similarity of Graphs

This section presents a similarity measure of the eigenvalues and eigenvectors of graphs. In particular, we first transform the eigenvalues and eigenvectors into a uniform representation. Secondly, we permutate the rows of the eigenvector matrix for similarity comparison. Finally, we propose a spectral similarity function of graphs.

5.1 Unifying the Dimensionalities of Graphs

The first issue on using Λ and U for representing and comparing graphs is that the dimensions of Λ and U of different graphs are different. That is, they cannot be directly compared. Therefore, we unify the dimensions of Λ and U as follows.

(i) We use the tail-k non-zero eigenvalues of Λ_G and the corresponding eigenvectors of U_G to represent a graph G. According to [16], the eigenvectors of the tail-k non-zero eigenvalues provide the best approximation of the U_G. This unifies not only the number of eigenvalues Λ_G but also the column dimension of U_G.

(ii) Each row of U_G corresponds to a vertex in G. The row dimension of U_G of the graphs can be unified by adding rows of zeros, until the dimension of U_G matches the largest graph in the database. By simple matrix theories, we have that adding zero rows to a matrix does not affect both its eigenvalues and the directions of its eigenvectors. In our context, a zero row vector corresponds to an isolated (virtual) vertex of a graph and does not affect the relative performance of indexes.

We remark that the computation of the similarity between the eigenvector matrices involves determining the cosine similarity between *each pair of the eigenvectors* (to be detailed in Formula 1). The computation complexity is quadratic to the number of eigenvectors in the matrices. Due to this performance issue, we opt not to introduce eigenvectors to unify the column dimension of U_G. In contrast, the computation time of similarity between U_G is linear to the size of the row dimension. Thus, our dimension unification does not lead to a significant increase in computation time while keeping the characteristics of vertices.

5.2 Permutation of Vertex ID

While the eigenvalues of G are graph invariants [3], the eigenvectors (columns) in U_G are not. Since a row in U_G represents the characteristics of a vertex in G and the IDs of rows are directly related to the IDs of vertices, graphs with similar structures may have very different eigenvector matrices. This can be illustrated with a simple example shown in Figure 3. In Figure 3, graphs G_1 and G_2 are isomorphic and they are expected to have the same Λ's and U's. However, U_{G_1} and U_{G_2} are different, and a direct similarity computation (to be defined in Section 5.3) yields a low similarity score 0.57. We reorder the rows in the eigenvector matrices of U_{G_1} and U_{G_2} to obtain U'_{G_1} and U'_{G_2}, respectively, as shown in Figure 3. Using U'_{G_1} and U'_{G_2} for similarity computation, we obtain a similarity score 1.0.

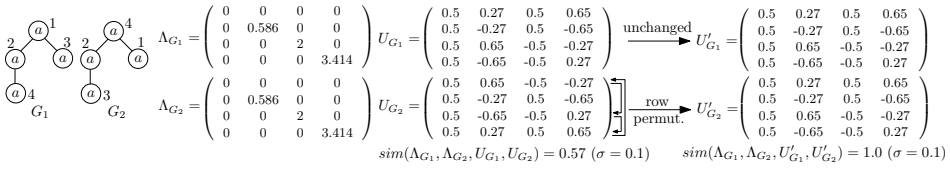


Fig. 3. Eigenvalues and eigenvectors of two isomorphic graphs

Unfortunately, the number of possible permutations are $n!$, where n is the number of rows. Therefore, we propose three practical heuristic functions to reorder the rows. The aim is that vertices with similar spectral characteristics are compared.

(i) VP_rad: for each row in U_G, we compute its L_2-norm. We order the rows by their L_2-norms in descending order. Intuitively, the L_2-norm of a row vector denotes its distance (radius) from the original point in a vector space. Therefore, the row vectors that are far (respectively, close) to the original point are compared.

(ii) VP_coor: According to D. Spielman [15], each row of U_G can be considered as the coordinates of a vertex of G in a vector space. The rows can then be ordered in a descending lexicographic order. It may be worth-remarking that such an ordering of rows is biased towards the first few dimensions of rows. As the first few dimensions correspond to smaller eigenvalues and hence, they are considered more important than the latter dimensions.

(iii) VP_W_rad: Since the eigenvalues indicate the importance of a dimension, we may integrate the heuristic function with eigenvalues. Specifically, for each row vector, we compute its weighted L_2-norm, where the weight of the i-th entry of a row is $1/\lambda_i$, and we order the rows by their weighted L_2-norms in descending order.

In summary, the processing presented above are applied to both rows and columns of all graphs to obtain a unified representation, for similarity computation. In subsequent discussions, we simply use Λ and U to refer to the matrices whose the dimensions have been unified and ordered.

5.3 Spectral Similarity Between Graphs

The central part of the prediction framework is the spectral similarity between graphs. The spectral similarity between two graphs has two major components.

Firstly, we determine the most comparable eigenvectors between two graphs G_1 and G_2. Specifically, for each eigenvector X_i in U_{G_1}, we pair X_i with an eigenvector Y_j of U_{G_2} whose direction is the most similar to X_i, defined as follows:

$$p[i] = argmax_{j=1,\ldots,n}(cos\_sim(X_i, Y_j)), \tag{1}$$

where X_i in U_{G_1}, Y_j in U_{G_2} and cos_sim denotes the cosine similarity between vectors.

Secondly, the *spectral similarity between two graphs* G_1 and G_2 is defined as a weighted sum of cosine similarity between paired eigenvectors, where the weights are proportional to the difference between the corresponding eigenvalues:

$$sim(G_1, G_2) = \frac{\sum_{\substack{(\lambda_i, X_i)\in G_1, \\ (\lambda_{p[i]}, Y_{p[i]})\in G_2}} (cos\_sim(X_i, Y_{p[i]}) \times e^{-\frac{(\lambda_i - \lambda_{p[i]})^2}{2\sigma^2}}))}{\sum_{\lambda_i \in G_1, \lambda_{p[i]} \in G_2} e^{-\frac{(\lambda_i - \lambda_{p[i]})^2}{2\sigma^2}}}, \tag{2}$$

where σ is a parameter that controls the importance between eigenvalues and eigenvectors. In particular, the larger the σ, the larger influence of the eigenvalues to the sim function. To compare two spectrums, (i) we compute the weight determined by a Gaussian function on the difference between the eigenvalues, $e^{-\frac{(\lambda_i - \lambda_{p[i]})^2}{2\sigma^2}}$. We assume the differences of eigenvalues follow a normal distribution. (ii) The cosine similarity between the corresponding eigenvectors is multiplied by the weight. (iii) The denominator normalizes the similarity function.

6 Prediction Algorithm

With the uniform representation of graphs in $\mathcal{D}$ and the spectral similarity function, we are ready to present our prediction algorithm. In this paper, the *label* of a graph G is its optimal index among a given set of indexes $\mathcal{I}$ = $\{I_1, I_2, ..., I_n\}$. We use the eigenvalues and eigenvectors of graphs in $\mathcal{D}$ and their labels to train a prediction model. Classical k-Means clustering is adopted to cluster the graphs in $\mathcal{D}$. As we shall see from

Procedure kMeans
Input: labels, Λ's and U's of graphs in $\mathcal{D}$, cluster number k,
and termination parameter δ
Ouput: the k clusters $\mathcal{C}$ and their centers

01 randomly choose k centers for k clusters $\mathcal{C}$
02 **for each** G **in** $\mathcal{D}$
03 i_{max} = $argmax_{i=1,\ldots,k}(sim(G, \mathcal{C}[i].ctr))$ //$\mathcal{C}[i]$ is ith cluster
04 assign G to the cluster $\mathcal{C}[i_{max}]$
05 **while** the clusters $\mathcal{C}$ have changed
06 **for each** $\mathcal{C}[i]$, where $\mathcal{C}[i] \in \mathcal{C}$, δ% of graphs in $\mathcal{C}[i]$ changed
07 // recalculate the center of $\mathcal{C}[i]$
08 **for each** $G \in \mathcal{C}[i]$ $G.sim$ = $\sum_{G' \in \mathcal{C}[i]}(cos\_sim(G, G')$
09 $\mathcal{C}[i].ctr$ = $argmax_{G \in \mathcal{C}[i]}(G.sim)$ // $\mathcal{C}[i].ctr$ is center of $\mathcal{C}[i]$
10 **for each** G **in** $\mathcal{D}$
11 i_{max} = $argmax_{i=1,\ldots,k}(sim(G, \mathcal{C}[i].ctr))$
12 assign G to $\mathcal{C}[i_{max}]$

Fig. 4. Procedure kMeans

experiments, prediction with eigenvalues Λ alone is not as accurate as that with both Λ and U. However, some other classical methods, such as neural networks or decision trees, require to cast U into some numerical values for training, while preserving their semantics. This does not appear trivial and those methods are not adopted.

A trained model is then used to predict the label of a graph G', where $G' \notin \mathcal{D}$. Specifically, given a graph G', we determine its k-nearest cluster centers and apply a voting method to obtain the weighted majority of the label of those centers. Such label is returned as the optimal index of G'.

6.1 Clustering Algorithm

In this subsection, we highlight the adoption of k-Means clustering for training a prediction model. The overall algorithm (Procedure kMeans) for the construction is summarized in Figure 4. The label (*i.e.*, the optimal index) of each graph $G \in \mathcal{D}$ is determined by the definition presented in Section 4. Then, similar graphs in $\mathcal{D}$ are clustered by Procedure kMeans. The label of a cluster center represents the label of the cluster.

While Procedure kMeans follows the general framework of classical k-Means algorithm, we highlight the adoption, *i.e.*, Lines 06-09. Most importantly, the major modification is on the recalculations of the new cluster centers in Lines 08-09. Classical k-Means algorithms often use averaging of a distance function between data objects in a cluster to obtain a new cluster center. However, the averages of eigenvectors or eigenvalues do not correspond to any clear semantics. Therefore, in Line 08, we determine the sum of the spectral similarity between each graph and all other graphs in a cluster. The new cluster center is the graph with the highest similarity sum (Line 09).

Optimization. The clustering algorithm recalculates cluster centers in each iteration. However, in later iterations of Procedure kMeans, the changes of clusters are often small. In other words, the algorithm may then recompute the spectral similarity of the same graphs many times. To optimize this, we store the spectral similarities between graphs, where they are computed once and retrieved in later iterations.

6.2 Prediction Algorithm

To predict the optimal index of a graph $G' \notin \mathcal{D}$, one may be tempted to use the label of the cluster center that is the most similar to G'. However, as presented before, the cluster centers are data graphs themselves and sometimes they may not be the optimum centers and the prediction using one cluster center may be overly sensitive to the quality of the clusters. To enhance the robustness of the prediction, we opt to adopt a simple k-nearest neighbor algorithm, where k is a user-defined parameter. Specifically, given a graph G', we decompose it into Λ'_G and U'_G. We determine the k clusters $\mathcal{C}[i_1]$, $\mathcal{C}[i_2]$,..., $\mathcal{C}[i_k]$ from $\mathcal{C}$, whose centers are the most similar to G'. Suppose the label of $\mathcal{C}[i]$'s center is L. The vote of $\mathcal{C}[i]$ to L is defined as the spectral similarity between $\mathcal{C}[i]$'s center and G'. The optimal index of G' is the label with the largest sum of votes.

7 Experimental Evaluation

In this section, we conducted an experiment to verify the accuracies and efficiencies of the spectral decomposition approach to predict the optimal graph index.

7.1 Experimental Setup

Implementation: We ran our implementation on a commodity PC with a Quad-core 2.4GHz CPU with 4G memory running Windows 7. We implemented our proposed technique on MATLAB R2011a. We used the functions provided by MATLAB as far as possible, such as the functions to determine the eigenvalues and eigenvectors, and statistical distribution fittings.

Table. 1 Meanings & default values of parameters

parameter	meaning	default
k_knn	k in KNN	7
k_kmeans	k in KMeans	64
$kmeans$	whether KMeans is used in prediction	yes
$\|T\|$	no. of the samples used for testing	28
$\|D\|$	no. of samples used for both training and testing	256
k_tail	tail k eigenvalues and their corresponding eigenvectors used in graphs' similarity	32
σ	parameter in sim to balance the effection of eignvalues and eigenvectors	0.1

Table. 2 Fitting error (L_2-norm)

fitting error	Poisson	Normal	Gamma
R (`Interval`)	0.16	0.09	0.06
R (`2-hop`)	0.17	0.16	0.14
R (`Grail(1)`)	0.50	0.43	0.27
R (`Prime`)	0.43	0.49	0.26
S (`Interval`)	0.18	0.07	0.04
S (`2-hop`)	0.25	0.23	0.20
S (`Grail(1)`)	0.52	0.42	0.29
S (`Prime`)	0.77	0.70	0.51

Graph collections: We used both random graphs and scale-free graphs for our experiments, as they are popular classes of graphs used in analysis of graphs. Moreover, we had controlled over the sizes and densities of the generated graphs. The generators used were provided by Zhu *et al.* [21]. We generated 1,024 graphs for each kind of graphs. For random graphs, the average number of vertices and fanout were 3.4k and 6.8, respectively. For scale-free graphs, we set $\alpha = 0.27$ and $\beta = 10$ and obtained 1,024 graphs with an average number of vertices 3k and average fanout 7.2. We used "R" and "S" to denote experiments with random graphs and scale-free graphs, respectively.

Reachability query time collections: We ran the implementations of `Interval` [21], `Grail` [20], `2-hop` [2] and `Prime labeling` [13] on our graph collections. We ran 1,000 random reachability queries on each graph in our graph collections. The runtimes of 1,000 queries on each index were then stored and fitted into Gamma distributions.

The runtimes were obtained from warm runs. The estimated α and β of Gamma distributions were stored for determining the optimal index.

We observe that there were cases that the prediction problem was trivial, *e.g.*, one index was almost always more efficient than the others. These cases were omitted as our prediction model was very accurate. In other words, neither of the indexes chosen in our experiments dominated each other.

Default parameter settings: We conducted a set of experiments to show the effect of each parameter in our technique. Unless specified otherwise, we used the default settings shown in Table 1. We used the VP_rad utility function for the vertex permutation in sim computation by default. For ease of exposition, we predict the optimal index from two graph indexes. That is, the prediction label of the experiments was binary.

Performance metric: Unless otherwise specified, we ran each experiment 100 times and reported the average accuracies.

7.2 Experiments on Distribution Fittings

To verify that the distributions of the reachability query times on each index can be fit into some well-known distributions, we tested fitting functions of various distributions. We generated the runtime distributions of all of our random and scale-free graphs and all of our index implementations. We used y%=98% in our experiments. In Table 2, we showed the L_2-norm between the actual distribution and the estimated distribution of Poisson, Normal and Gamma distributions. We note that Gamma distributions almost always clearly offered more accurate fittings and the fitting errors were often small. Therefore, in this work, we adopted Gamma distribution.

Table. 3 Effects of training dataset size on R

$\|D\|$	average accuracy	
	2-hop vs Grail(1)	Interval vs Prime
64	84.18%	79.18%
128	87.57%	83.14%
256	87.11%	82.68%
512	85.18%	82.54%
1024	81.79%	81.07%

Table. 4 Effects of k in kMeans on R and S

k_kmeans	average accuracy (R)		average accuracy (S)	
	2-hop vs Grail(1)	Interval vs Prime	2-hop vs Grail(3)	2-hop vs Grail(5)
8	71.21%	70.43%	63.64%	65.11%
16	79.61%	77.29%	76.75%	79.68%
32	85.25%	80.50%	79.79%	81.18%
64	89.46%	83.75%	78.71%	80.00%
128	90.54%	84.21%	76.00%	76.89%

Table. 5 Effects of k in knn on R and S

k_knn	average accuracy (R)		average accuracy (S)	
	2-hop vs Grail(1)	Interval vs Prime	2-hop vs Grail(3)	2-hop vs Grail(5)
1	83.57%	79.61%	67.86%	67.46%
3	86.79%	81.43%	73.68%	74.86%
5	88.82%	82.61%	77.36%	77.50%
7	89.79%	83.00%	79.86%	80.57%
9	89.29%	82.82%	81.07%	80.75%

Table. 6 Effects of k in k_tail on R and S

k_tail	average accuracy (R)		average accuracy (S)		time
	2-hop vs Grail(1)	Interval vs Prime	2-hop vs Grail(3)	2-hop vs Grail(5)	(s)
8	81.68%	79.54%	69.43%	68.93%	3.65
16	84.00%	81.11%	73.75%	74.57%	10.00
32	88.29%	83.75%	77.79%	80.04%	30.10
64	88.43%	83.43%	80.25%	82.89%	105.88
128	88.82%	84.39%	81.71%	83.04%	413.09

7.3 Prediction Accuracies

In this experiment, we tuned the parameters of our prediction model and studied their effects on prediction accuracies. Due to space limitations, we present the results on

some pairs of indexes for each dataset: `2-hop` vs `Grail(1)` and `Interval` vs `Prime` for random graphs, and `2-hop` vs `Grail(3)` and `2-hop` vs `Grail(5)` for scale-free graphs.

Effects of the size of training dataset. We studied the effects of the training dataset size to the prediction accuracies on random graphs as shown in Table 3. $kmeans$ was set to `no` in this experiment. From Table 3, we observe that our prediction accuracies were almost above 80% on training sets of different sizes. We also note that the prediction accuracy first increased and then slightly reduced with the growth of training dataset size. It was possibly because our model was overfitted by large training sets.

Effects of k_kmeans. We studied the effects of k_kmeans on our prediction accuracies as shown in Table 4. From Table 4, we observe that our prediction accuracies increased as the growth of k_kmeans on random graphs. It was because that the clusters would be more refined with a larger k_kmeans. Thus, we had a higher probability to choose similar cluster centers for prediction. However, the prediction accuracy may reduce slightly if each cluster is too fine, as shown in the results on scale-free graphs.

Effects of k_knn. Table 5 presents the effects of k_knn on our prediction accuracies. From Table 5, we observe that our prediction accuracy increases with the growth of k_knn. It is because that we have more votings with larger k, which reduces the effects of outliers in prediction. Our prediction accuracy was over 80% on both random graphs and scale-free graphs when $k \geq 7$. The prediction accuracy was stable when $k \geq 9$.

Effects of the number of tail-k eigens. We used different number of eignvalues and eigenvectors in prediction and studied the prediction accuracies as shown in Table 6. From Table 6, we observe that the prediction accuracy increased with more eignvalues and engnvectors, while the prediction time increased roughly linearly. We exclude the time for determining the spectrum of a graph as it mainly depends on the algorithm we used. From our data collections, the MATLAB function ran from 2.6s to 173s and 39s on average. However, this is often more efficient than choosing the optimal index by constructing all candidate indexes and running a large number of benchmark queries.

Effects of vertex permutation and σ. In this experiment, we studied the effects of vertex permutation and σ on our prediction accuracies. Table. 7 presents the results. From Table. 7, we observe that while VP_coor, VP_W_rad and VP_none could sometimes be accurate, they could be sensitive to the choice of σ, in the similarity function. In comparison, VP_rad is both robust and accurate. Moreover, when σ increased, the relative importance of eigenvectors lowered and the accuracies decreased.

Table. 7 Effects of vertex permutation and σ on `R`

Vertex	σ (`2-hop` vs `Grail(1)`)					σ (`Interval` vs `Prime`)				
Permutation	0.01	0.1	1	10	100	0.01	0.1	1	10	100
VP_rad	89.43%	89.71%	82.25%	83.25%	71.21%	82.93%	83.21%	78.64%	74.14%	73.64%
VP_coor	91.00%	90.21%	69.25%	69.11%	68.93%	82.71%	85.96%	73.00%	73.00%	72.43%
VP_W_rad	89.39%	90.68%	70.61%	72.21%	71.14%	82.71%	84.89%	72.86%	74.36%	75.25%
VP_none	87.36%	80.86%	82.25%	69.50%	70.00%	81.89%	82.82%	78.61%	72.86%	72.14%

8 Related Work

To the best of our knowledge, there have been only few preliminary studies that use graph features to predict the relative query performances of graph indexes. Deng *et al.* [7]. extract features from data graphs and use neural networks for prediction. However, there have been many features in graph theories and it is unclear which of these are the principal ones. In contrast, we use the tail-k eigenvalues and eigenvectors for prediction. Moreover, the optimal index of [7] is defined by the best average runtimes. In comparison, we also allow users to fine-tune their notion of the optimal index. Another work by Zhu *et al.* [21] applies multiple graph indexes to partitioned subgraphs of a data graph. An analytical cost model is proposed and illustrated with `2-hop` and `Interval`. Our approach has been applied to various indexes. Spectral methods have been applied to produce k partitions of graphs *e.g.*, [12]. Our aim is not to produce exactly k partitions but to predict to the optimal index.

Finally, there is a large body of work on determining graph features, *e.g.*, [18], for query processing, *e.g.*, [19, 4]. Due to space limitations, we cannot include a detailed survey on this area. However, in these studies, features are graphs (structures). It remains unclear how to exploit them to build a predictive model.

9 Conclusions

In this paper, we propose spectral decomposition for predicting optimal graph index of a given graph. Specifically, we have proposed a uniform representation of a graph, spectral similarity function and a prediction algorithm. We obtained the implementation of four structurally different graph indexes. One observation is that the runtime distributions of the indexes fit accurately into Gamma distribution. This allows us to refine the notion of the optimal index, with the inverse cumulative distribution function. We conducted detailed experiments on the parameters in our techniques on both random graphs and scale-free graphs. We noted that our technique is robust and can achieve approximately higher than 70% accuracies in most cases. As for future works, we are investigating on the support of other graph queries, such as subgraph queries.

References

1. R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *SIGMOD*, pages 253–262, 1989.
2. R. Bramandia, B. Choi, and W. K. Ng. Incremental maintenance of 2-hop labeling of large graphs. *TKDE*, 22:682–698, 2010.
3. A. E. Brouwer and W. H. Haemers. *Spectra of Graphs*. Springer, 2012.
4. J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. In *SIGMOD*, pages 857–872, 2007.
5. F. Chung. *Spectral Graph Theory*. Conference Board of the Mathematical Sciences, 1997.
6. E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In *SODA*, pages 937–946, 2002.
7. J. Deng, F. Liu, Y. Peng, B. Choi, and J. Xu. Predicting the optimal ad-hoc index for reachability queries on graph databases. In *CIKM*, pages 2357–2360, 2011.
8. I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *TPAMI*, 29:1944–1957, 2007.
9. B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Sci. Comput.*, 16(2):452–469, 1995.

10. J. Y. Jonathan L Gross. *Handbook of Graph Theory*. CRC Press, 2004.
11. Y. Ke, J. Cheng, and J. X. Yu. Querying large graph databases. In *DASFAA*, pages 487–488, 2010.
12. A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856. MIT Press, 2001.
13. Y. Peng, B. Choi, and J. Xu. Selectivity estimation of twig queries on cyclic graphs. In *ICDE*, pages 960–971, 2011.
14. R. Schenkel, A. Theobald, and G. Weikum. Efficient creation and incremental maintenance of the hopi index for complex xml document collections. In *ICDE*, pages 360–371, 2005.
15. D. A. Spielman. Spectral graph theory and its applications. In *FOCS*, pages 29–38, 2007.
16. D. A. Spielman. Spectral graph theory. In *Combinatorial Scientific Computing.*, Chapman and Hall/CRC Press, pages 1–23, 2011.
17. X. Wu, M. L. Lee, and W. Hsu. A prime number labeling scheme for dynamic ordered xml trees. In *ICDE*, pages 66–, 2004.
18. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.
19. X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004.
20. H. Yildirim, V. Chaoji, and M. J. Zaki. Grail: scalable reachability index for large graphs. *PVLDB*, 3(1-2):276–284, 2010.
21. L. Zhu, B. Choi, B. He, J. X. Yu, and W. K. Ng. A uniform framework for ad-hoc indexes to answer reachability queries on large graphs. In *DASFAA*, pages 138–152, 2009.

Appendix: Background on Graph Indexes

For self-containedness, we include a brief discussion on the index benchmarked in this section. There have been many graph indexes proposed recently to support reachability queries, not to mention other graph queries. This section only reviews the four indexes adopted in this paper. Interested readers may refer to tutorials by Cheng et al. [11] for a comprehensive survey on graph indexes.

We remark that since indexes often associate labels to nodes, we may use the terms *indexes* and *labels* interchangeably.

(1) `Interval labeling` is a popular indexing technique for trees, in particular `XML`. It exhibits good query performances and has simple implementation. `Interval labeling` associates each vertex with an interval (i, j), where i is the vertex's preorder traversal number and j is its postorder traversal number. Vertex w can be reached from vertex v iff the interval of w is contained in that of v. `Interval labeling` is then extended to support `DAG`s [1]. In particular, each vertex may be associated with multiple intervals, as there may be multiple paths between any two nodes. To evaluation a query correctly, the extended interval labeling requires comparisons on all pairs of intervals of vertices.

(2) `Grail` [20] is a recent index derived from interval labeling. Users may specify a parameter k. `Grail` determines k random spanning trees from a graph and applies the interval indexes on the k spanning trees. The remaining non-tree edges are considered as exceptions and handled specially. The query performance of the interval indexes is more superior to that of the handling of exceptions. In other words, the performance of `Grail` of a given query depends on what parts of `Grail` indexes the query requires.

(3) `2-hop labeling` [6] is another popular index for reachability queries. Each vertex v is associated with two labels $L_{in}(v)$ and $L_{out}(v)$, where $L_{in}(v)$ stores *some* vertices that can reach v whereas $L_{out}(v)$ contains some vertices that v can reach. Then, w is reachable from v iff $L_{out}(v) \cap L_{in}(w) \neq \emptyset$. There are many possible `2-hop labeling` for a graph. To determine the `2-hop labeling` with the minimum size, various heuristics, optimization and partitioning algorithms have been proposed. It has been to complex to mathematically model the performance of the resulting `2-hop labeling`.

(4) `Prime labeling` [17] uses products of prime numbers to encode reachability information of trees. A follow-up work [13] extends `Prime labeling` to support reachability queries on DAGs. Specifically, each vertex is associated with a product of prime numbers and vertex v can reach vertex w iff v's label is divisible by w's label. An implementation concern of `Prime labeling` is that for graphs with large depths, `Prime labeling` may result in large products, which require the support of very large integers.

It is evident that the indexes discussed above are structurally complex and there are little relationships between their structures. Deriving an accurate analytical model to dictate their relative performances is known to be a daunting task. In this paper, we apply data mining techniques and spectral decomposition to build a model to predict the relative performances. In addition, the implementations of these indexes are available for our experiments. Hence, we include these four indexes in this paper.