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. There is an ample body of recent research on indexing for structural graph queries. However, as verified by our experiments with a large number of random and scale-free graphs, there may be a great variation in the performances of indexes of graph queries. Unfortunately, the structures of graph indexes are often complex and *ad-hoc*, so deriving an accurate performance model is 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 proposing a spectral decomposition method for predicting the relative performances of graph indexes. Specifically, given a graph, we compute its spectrum. We then propose a similarity function to compare the spectrums of graphs. We adopt a classification algorithm to build a model and a voting algorithm for predicting the optimal index. Our empirical studies on a large number of random and scale-free graphs, using four structurally distinguishable indexes, demonstrate that our spectral decomposition method is robust and almost always exhibits an accuracy of 70% or above.

1 Introduction

Due to the flexibility of the 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 have recently been proposed. Unfortunately, graph data are often heterogeneous and the structures of their indexes are often complex and *ad-hoc*. As our experiments reveal, the performances of such graph indexes may vary greatly. This leads to a natural question for database practitioners: *Which index is the most efficient for a given graph?*

When compared to their relational counterparts, the structures of many graph indexes are far more complex. This causes a few unique problems. Firstly, it is sometimes time-consuming to construct the indexes. For example, our experiments on a commodity computer show that given a random graph of a modest size ($\sim$3,000 vertices and a density of 0.02), the construction time for a graph index, namely `2-hop labeling` [12], is already 8.3 seconds. (For background details of the indexes discussed, please refer to our technical report [13].) While some other graph indexes can be constructed in less than a second, the most time-efficient index can only be identified after *all* indexes have been constructed and benchmarked. Furthermore, performance depends not only on the

J. Pei et al. (Eds.): PAKDD 2013, Part I, LNAI 7818, pp. 187–200, 2013.

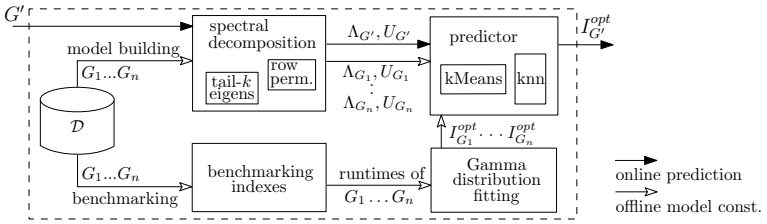


Fig. 1. Overview of our spectral decomposition prediction framework

algorithms but also the details and quality of the implementation, as there is not yet a well-received (*i.e.*, commercial) implementation in place. Secondly, graph indexes are sometimes several times larger than the graph itself. Using the example given above, the index generated by `2-hop labeling` is 19 times larger than the graph whereas that by `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. Overall, it is clearly desirable to be able to predict the optimal index and construct it, and only it, for efficient query processing.

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

In this paper, we apply data mining techniques to predict the relative performances of different graph indexes. One of the core problems is to extract important features (or characteristics) from data graphs. While a large variety of features have been studied in graph theories [9], it is not yet clear which are the most relevant to index performances. Therefore, we propose to apply a general technique derived from spectral graph theories [5], namely spectral decomposition, to solve this problem. Spectral methods have been reported successful in many applications such as VLSI design and computer vision. To the best of our knowledge, this is the first work to empirically demonstrate the relationships between graph spectrums and index performances. In general, graph spectrums are known to be *characteristics of graphs* and to be related to many important graph properties. Another advantage is that they are supported by industrial-strength softwares, not to mention the availability of their advanced optimizations.

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 the same graph, can often exhibit large variances. For example, we ran 1,000 random queries on each of 8,000 random graphs and indexed each using `2-hop` and `Prime labeling` [11]. The mean and standard deviation of `2-hop` are 14.1 seconds and 4.2 and those of `Prime labeling` are 11.6 seconds and 59.7, respectively. Moreover, the runtimes are often skewed and have a long tail at large values. (In a later section, we illustrate some of the runtime distributions in Figure 2.) We observe a similar phenomenon in our collection of scale-free graphs. A possible explanation is that a graph may contain many different sub-structures and the indexes are also complex structures. This leads to a wide range of runtimes. While average runtimes are often used to quantify index performances, it is desirable to propose a more flexible metric.

In this paper, we fit the runtimes of queries into a distribution. From our experiments on estimating its parameters, the goodness of fit of the Gamma distribution is always the best. By comparing the distributions of runtimes, we can obtain a more robust and flexible way to compare performances. While we may apply the research on the Gamma distribution for further analysis, in this paper, we apply the inverse cumulative distribution function to estimate the time when $y\%$ of queries finish. Depending on users' applications, they may specify a value for $y\%$ to express their "tolerance" of long query runtimes. For instance, some Internet connection providers (*e.g.* hotels and cafes) charge their users according to connection time and so long query runtimes can be undesirable. To cater for their needs, data practitioners may choose the index that is optimal at 98%, instead of the one that has the optimal average runtime.

Contributions. To our knowledge, this is the first investigation of graph spectrums in relation to index performances. We summarize the contributions of this work below and present an overview of these 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}. These eigenvalues and eigenvectors are transformed into a unified representation for comparison purposes.
- We propose a spectral similarity function between two graphs.
- We propose to fit the runtimes of an index on a given graph into the Gamma distribution using a distribution fitter. Users may then evaluate this in terms of their desired optimal index.
- We adopt the k-Means algorithm and k-nearest neighbor with a voting method for predicting the optimal index $I^{opt}_{G'}$ of a given graph G'.
- We conduct experiments with a large number of random and scale-free graphs. The results show that our proposed technique can achieve accuracies that are almost always above 70% and very often above 80%.

The rest of the paper is organized as follows. Section 2 presents the backgrounds to graph spectral decomposition. We present our problem statement in Section 3. Section 4 presents the definition of the 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 and our experimental evaluation is reported in Section 7. Section 8 discusses related work and Section 9 concludes this paper.

2 Background to Graph Spectral Decomposition

In this section, we provide a background to graph spectrums. Graph spectrums have been widely used to study many interesting properties of graphs, such as spectral partitioning and expansion [8], the cut problem [7] and graph drawing [14].

Graph spectrum is often defined using 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 call U the *eigenvector matrix* and use Λ to refer to eigenvalues. We use a subscript in U_G and Λ_G to denote the U and Λ of graph G if needed.

Eigenvalues Λ are called *spectrums* in spectral graph theories. Since eigenvectors U are also known to be closely related to the characteristics of vertices, we include them in our algorithm. We use the Λ and U of the underlying undirected graphs of the data graphs, as they capture their 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 earlier, various types of indexes are available to support the reachability query on graphs in $\mathcal{D}$. However, it is desirable to predict the optimal one without building and benchmarking all the available indexes. 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 within the problem statement.

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

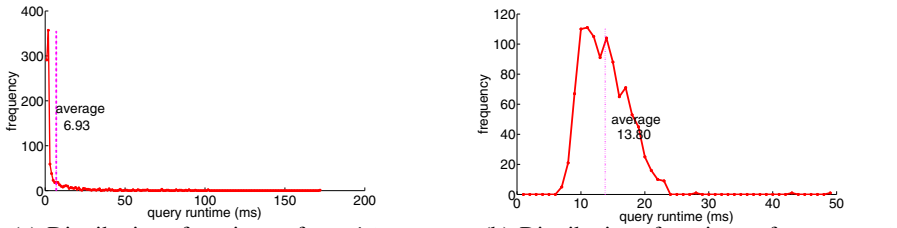
As motivated in Section 1, the query runtimes of a particular index on a particular graph may deviate from the average. Therefore, in Section 4, we investigate the more flexible notion of the *optimal index* in expressing desirable runtimes.

(P2) How do we compare spectrums of graphs?

As mentioned 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 representations 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 therefore transformed into a uniform representation for comparison purposes. Finally, while the eigenvalues Λ_G are invariants of G, the row vectors of the eigenvector matrix U_G are dependent on the permutations of the vertex IDs of G. There is hence a row permutation problem when 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 performance, we plot the query time distribution, where the x-axis is the query time and the y-axis the number of queries finished at time x. For example, Figures 2(a) and (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 the desired notion of performance. For example, Figure 2(a) shows that `Grail(1)` [18] has the smallest average time but has a long tail. In comparison, the rutntimes of `Interval` exhibit a relatively small variance, although `Interval` has a relatively large average time. Therefore, `Interval` could be a better choice in applications where long query times are unacceptable or commercially unfavorable.



(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

Once the query runtimes are presented as a distribution, we fit this to some well-studied distributions, such as Normal, Poisson and Gamma distributions. To measure goodness of fit, we adopt the L_2-norm between the estimated and the real distributions. From our experiments on a large number of random and scale-free graphs, we observe that the Gamma distribution almost always yields the best fit. A possible reason for this is that it is often used to model the waiting time and the 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 the Gamma distribution can be efficiently estimated. Therefore, we use it to represent the query runtime.

While the optimal index is intuitively the most efficient on 1,000 random queries, we define it as the one with the smallest estimated runtime w. r. t. the user-defined parameter $y\%$ (*e.g.*, 98%). Once the parameters of the Gamma distribution have been estimated, the notion of the 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%, *without rerunning the benchmarking queries*. An index can be considered robust if it is optimal for all those y values, instead of a specific one.

5 Spectral Similarity of Graphs

This section presents a similarity measure for the eigenvalues and eigenvectors of graphs. Specifically, we first transform these into a uniform representation and then permute the rows of the eigenvector matrix to allow a similarity comparison. Finally, we propose a spectral similarity function for graphs.

5.1 Unifying the Dimensionalities of Graphs

The first issue in using Λ and U to represent and compare 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 [15], the eigenvectors of the tail-k non-zero eigenvalues provide the best approximation of 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. Using simple matrix theories, we have that adding zero rows to a matrix affect neither its eigenvalues nor 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 further 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 the 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 retaining the characteristics of the 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 using the simple example shown in Figure 3. Here, graphs G_1 and G_2 are isomorphic and so 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 of 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},

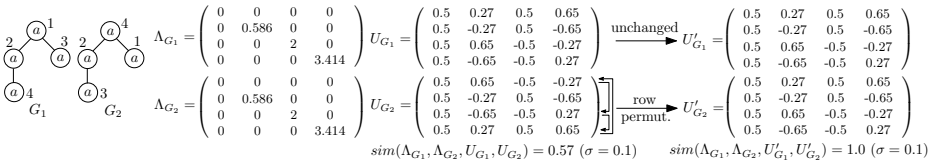


Fig. 3. Eigenvalues and eigenvectors of two isomorphic graphs

respectively, as shown in Figure 3. Using U'_{G_1} and U'_{G_2} for the similarity computation, we obtain a similarity score of 1.0.

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

(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 (or, respectively, close) to the original point are compared.

(ii) VP_coor: According to [14], 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 descending lexicographic order. It is worth-remarking here that such an ordering of rows is biased towards the first few dimensions, which correspond to smaller eigenvalues and hence are considered more important than the latter dimensions.

(iii) VP_W_rad: Since the eigenvalues indicate the importance of a dimension, we may integrate them with the heuristic function. 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 then order the rows by their weighted L_2-norms in descending order.

In summary, the processing presented above can be applied to both the 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 dimensions have been unified and ordered in this manner.

5.3 Spectral Similarity between Graphs

The central part of the prediction framework is the spectral similarity between graphs. This 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 the weighted sum of the cosine similarity between paired eigenvectors, where the weights are proportional to the difference between the corresponding eigenvalues:

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

where σ is a parameter that controls the importance between eigenvalues and eigenvectors. In particular, the larger the value of σ, the greater the influence of the eigenvalues on 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 such a difference of eigenvalues follows a normal distribution; (ii) the cosine similarity between the corresponding eigenvectors is multiplied by the weight; and (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. To summarize the graphs in $\mathcal{D}$, we simply adopt classical k-Means clustering, while other clustering techniques may be applied. As we shall see from experiment, prediction with eigenvalues Λ alone is not as accurate as that achieved 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 so these 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 their labels. Such a 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 that cluster.

While Procedure `kMeans` follows the general framework of classical k-Means algorithm, we highlight this particular adoption, *i.e.*, Lines 06-09. Most importantly, the major modification is to the recalculation 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 center. However, the averages of the 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).

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

Optimization. The clustering algorithm recalculates cluster centers in each iteration. However, in later iterations of Procedure kMeans, the changes in 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 when they are first computed and then 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 discussed above, the cluster centers are data graphs themselves and sometimes may not be optimal. Accordingly, the prediction using one cluster center may be overly sensitive to the quality of the clusters. To enhance the robustness of the prediction, we adopt a simple k-NN 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 report on an experiment conducted to verify the accuracy and efficiency 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 using MATLAB R2011a. We used the functions provided by MATLAB as far as possible, such as those determining the eigenvalues and eigenvectors, and statistical distribution fittings.

Graph Collections: We used both random and scale-free graphs for our experiments, as they are popular classes used in graph analysis. Moreover, we controlled for their sizes and densities. The generators used were provided by Zhu et al. [19]. We generated 1,024 graphs of each type. 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 of 3k and average fanout 7.2. We use `R` and `S` to denote experiments with random and scale-free graphs, respectively.

Reachability Query Time Collections: We ran the implementations of `Interval` [19], `Grail` [18], `2-hop` [2] and `Prime labeling` [11] on our graph collections. We ran 1,000 random reachability queries on each graph. The runtimes of these 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. The label of a graph is determined by the y value.

We observe that there were cases where 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 another.

Table 1. Meanings & default values of parameters

parameter	meaning	default
k_knn	k in k-NN	7
k_kmeans	k in KMeans	64
$kmeans$	whether KMeans is used in prediction	`yes`
$\|T\|$	no. of 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 importance 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

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 the 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 report here the average accuracies.

7.2 Experiments on Distribution Fittings

To verify that the distributions of the reachability query times on each index can be fitted to some well-known distributions, we tested their fitting functions. We generated the runtime distributions of all of our random and scale-free graphs and all of our index

Table 3. Effects of $|D|$ on R

$\|D\|$	average accuracy (ours / [6])	
	2-hop vs Grail(1)	Interval vs Prime
64	84.18% / 85.06%	79.18% / 75.86%
128	87.57% / 87.35%	83.14% / 85.31%
256	87.11% / 88.18%	82.68% / 85.25%
512	85.18% / 92.03%	82.54% / 89.36%

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 (s)
	2-hop vs Grail(1)	Interval vs Prime	2-hop vs Grail(3)	2-hop vs Grail(5)	
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

implementations. We used $y\%$=98% in our experiments. In Table 2, we show the L_2-norm between the actual and estimated distribution of Poisson, normal and Gamma distributions. We note that the Gamma distributions almost always clearly offered more accurate fittings and the fitting errors were often small. Therefore, in this work, we have adopted the Gamma distribution.

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 only the results obtained from 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 effect of the dataset size on prediction accuracy for 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 always above 80% on training sets of different sizes. We also note that the prediction accuracy first increased and then slightly declined as the training dataset grew larger. This was possibly because our model was overfitted by large training sets. Moreover compared our method with a previous work [6]. While our method is either comparable or slightly less accurate, it is less *ad-hoc* than [6]. As discussed, spectrums have known to be useful in many real applications. However, in [6], the features were chosen simply because they are verified useful by experiments.

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

Effects of k_knn. Table 5 presents the effects of k_knn on prediction accuracy. It can be seen that accuracy increases with the growth of k_knn. This is because that a large k results in more votings, which reduces the effects of outliers. Our accuracy was over 80% on both random and scale-free graphs when $k \geq 7$. The prediction accuracy was stable when $k \geq 9$.

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%

Effects of the Number of Tail-k Eigens. We used different number of eignvalues and eigenvectors in prediction and studied the effect on accuracy, as shown in Table 6. It can be seen that accuracy increases with more eignvalues and eignvectors, while the prediction time increased roughly linearly. We exclude the time taken to determine the spectrum of a graph as this mainly depends on the algorithm used. From our data, the MATLAB function ran from 2.6s to 173s with an average of 39s. However, this is often still 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 prediction accuracy. Table. 7 presents the results. It may be observed that while VP_coor, VP_W_rad and VP_none could all sometimes be accurate, they were 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 reduced, and accuracy decreased.

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. [6]. 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 [6] 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. [19] 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.*, [10]. 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.*, [16], for query processing, *e.g.*, [17,4]. 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 a spectral decomposition method for predicting the optimal graph index of a given graph. Specifically, we propose a uniform representation of a graph, a spectral similarity function and a prediction algorithm. We obtain the implementation of four structurally different graph indexes. One observation is that the runtime distributions of the indexes fit accurately into a Gamma distribution. This allows us to refine the notion of the optimal index, using the inverse cumulative distribution function. We report on detailed experiments on the parameters in our techniques on both random graphs and scale-free graphs. We note that our technique is robust and can achieve approximately 70% accuracy or higher. In future work, we will investigate methods for other subgraph queries.

Acknowledgment. This work is partly supported by FRG2/10-11/106, GRF HKBU210510 and GRF 211512.

References

1. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. In: SIGMOD, pp. 253–262 (1989)
2. Bramandia, R., Choi, B., Ng, W.K.: Incremental maintenance of 2-hop labeling of large graphs. TKDE 22, 682–698 (2010)
3. Brouwer, A.E., Haemers, W.H.: Spectra of Graphs. Springer (2012)
4. Cheng, J., Ke, Y., Ng, W., Lu, A.: Fg-index: towards verification-free query processing on graph databases. In: SIGMOD, pp. 857–872 (2007)
5. Chung, F.: Spectral Graph Theory. Conference Board of the Mathematical Sciences (1997)
6. Deng, J., Liu, F., Peng, Y., Choi, B., Xu, J.: Predicting the optimal ad-hoc index for reachability queries on graph databases. In: CIKM, pp. 2357–2360 (2011)
7. Dhillon, I.S., Guan, Y., Kulis, B.: Weighted graph cuts without eigenvectors: A multilevel approach. TPAMI 29, 1944–1957 (2007)
8. Hendrickson, B., Leland, R.: An improved spectral graph partitioning algorithm for mapping parallel computations. SIAM J. Sci. Comput. 16(2), 452–469 (1995)
9. Yellen, J., Gross, J.L.: Handbook of Graph Theory. CRC Press (2004)

10. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: NIPS, pp. 849–856. MIT Press (2001)
11. Peng, Y., Choi, B., Xu, J.: Selectivity estimation of twig queries on cyclic graphs. In: ICDE, pp. 960–971 (2011)
12. Schenkel, R., Theobald, A., Weikum, G.: Efficient creation and incremental maintenance of the hopi index for complex xml document collections. In: ICDE, pp. 360–371 (2005)
13. Song, L., Peng, Y., Choi, B., Xu, J., He, B.: Spectral decomposition for optimal graph index prediction. Technique Report (2012), http://www.comp.hkbu.edu.hk/~ypeng/papers/TechRep2012.pdf
14. Spielman, D.A.: Spectral graph theory and its applications. In: FOCS, pp. 29–38 (2007)
15. Spielman, D.A.: Spectral graph theory. In: Combinatorial Scientific Computing, pp. 1–23. Chapman and Hall/CRC Press (2011)
16. Yan, X., Han, J.: gspan: Graph-based substructure pattern mining. In: ICDM, pp. 721–724 (2002)
17. Yan, X., Yu, P.S., Han, J.: Graph indexing: a frequent structure-based approach. In: SIGMOD, pp. 335–346 (2004)
18. Yildirim, H., Chaoji, V., Zaki, M.J.: Grail: scalable reachability index for large graphs. PVLDB 3(1-2), 276–284 (2010)
19. Zhu, L., Choi, B., He, B., Yu, J.X., Ng, W.K.: A uniform framework for ad-hoc indexes to answer reachability queries on large graphs. In: Zhou, X., Yokota, H., Deng, K., Liu, Q. (eds.) DASFAA 2009. LNCS, vol. 5463, pp. 138–152. Springer, Heidelberg (2009)