Scaling Manifold Ranking Based Image Retrieval

Yasuhiro Fujiwara[†], Go Irie[‡], Shari Kuroyama[∗], Makoto Onizuka[§†]
†NTT Software Innovation Center, 3-9-11 Midori-cho Musashino-shi, Tokyo, Japan
‡NTT Service Evolution Laboratories, 1-1 Hikarinooka Yokosuka-shi, Kanagawa, Japan
∗California Institute of Technology, 1200 East California Boulevard Pasadena, California, USA
§Osaka University, 1-5 Yamadaoka, Suita-shi, Osaka, Japan
{fujiwara.yasuhiro, irie.go}@lab.ntt.co.jp, kuroyama@caltech.edu, oni@acm.org

ABSTRACT

Manifold Ranking is a graph-based ranking algorithm being successfully applied to retrieve images from multimedia databases. Given a query image, Manifold Ranking computes the ranking scores of images in the database by exploiting the relationships among them expressed in the form of a graph. Since Manifold Ranking effectively utilizes the global structure of the graph, it is significantly better at finding intuitive results compared with current approaches. Fundamentally, Manifold Ranking requires an inverse matrix to compute ranking scores and so needs $O(n^3)$ time, where n is the number of images. Manifold Ranking, unfortunately, does not scale to support databases with large numbers of images. Our solution, *Mogul*, is based on two ideas: (1) It efficiently computes ranking scores by sparse matrices, and (2) It skips unnecessary score computations by estimating upper bounding scores. These two ideas reduce the time complexity of Mogul to $O(n)$ from $O(n^3)$ of the inverse matrix approach. Experiments show that Mogul is much faster and gives significantly better retrieval quality than a state-of-the-art approximation approach.

1. INTRODUCTION

Digital images have become widely available due to the proliferation of web services such as Flickr. Many researchers have developed image retrieval approaches for multimedia databases. The goal of image retrieval is to find the images that semantically match the query image. The key problem in designing a successful image retrieval system is how to rank the images in the database to suit the user's understanding of semantics, i.e., how to establish the correspondence between image content and semantic tags [2]. The 1970's saw the first image retrieval approaches based on keyword annotation [6]. In this paradigm, images in the database are first annotated with keywords, and then retrieved via their keywords. Although this keyword-based approach is still used in many actual image retrieval systems like Google and Yahoo, these systems suffer from problems such as insufficient text information and semantic inconsistency between the texts and images [21].

The above problems were tackled by content-based image retrieval, which was proposed in the early 1990's. This approach is based on the idea of returning the most visually similar images; the query image is compared to each database image. This approach takes an example image as a query (Query-By-Example) and ranks images based on low-level features such as color and shape where $\mathcal{L}^p$-norms are typically used as the similarity measure [6]. The advantage of this approach over the keyword-based approach lies in the fact that feature extraction can be performed automatically and the features are always consistent with the image's content [6]. In the database community, several researchers have proposed efficient techniques for nearest neighbor search in $\mathcal{L}^p$ spaces [19, 24]. However, semantically adjacent images may not always share the same neighborhood in the $\mathcal{L}^p$ spaces. Therefore, these techniques permit semantic gaps between low-level features and higher-level concepts [2]. How to bridge the semantic gap remains the main challenge in content-based image retrieval [2].

He et al. proposed to apply Manifold Ranking instead of $\mathcal{L}^p$-norms based similarity search for image retrieval [6]. Since semantically similar images are not always adjacent in the $\mathcal{L}^p$-spaces, results of k-nearest neighbor search can include semantically different images from query images [2]. For example, if the query image is a blue triangle, returned images can include a blue square, same color but different shape. Manifold Ranking computes ranking scores of images along with their underlying clusters (typically referred as manifolds). Unlike k-nearest neighbor search, Manifold Ranking can find semantically relevant images. Manifold Ranking exploits the property of the underlying clusters collectively revealed by a large number of images [2]. For example, images of blue triangles and blue squares construct different clusters even though they can occupy neighboring points in the $\mathcal{L}^p$-spaces. Since Manifold Ranking increases the ranking score of images that share the cluster of the query node [26], it can reliably evaluate the semantics of images in the database. Therefore, Manifold Ranking can suppress the gap between the low-level feature space and the semantic keyword space [6].

However, one of the most important deficiencies of Manifold Ranking is its speed, especially for large-scale data [8, 21]. Theoretically, the ranking scores of Manifold Ranking are those that minimize the cost function [25]. Since the optimal solution that minimizes the cost function is obtained by means of an inverse matrix of size $n \times n$, $O(n^3)$ time is

Proceedings of the VLDB Endowment, Vol. 8, No. 4
Copyright 2014 VLDB Endowment 2150-8097/14/12.

needed to compute the ranking scores where n is the number of data points. Moreover, identifying the optimal solution requires $O(n^2)$ space since all the elements in the inverse matrix must be kept in memory [8]. In this paper, we propose a fast and memory efficient solution to overcome these deficiencies of Manifold Ranking.

1.1 Problem Statement

In Manifold Ranking based image retrieval, a k-NN graph is typically used to model images in the database where nodes correspond to images [3]. In a k-NN graph, a node pair has an undirected edge if the two nodes are k-nearest neighbors. We address the following problem in this paper:

Problem (Top-k search for Manifold Ranking).
Given: *k-NN graph, query node, and required number of answer nodes.*
Find: *top-k nodes with respect to Manifold Ranking scores for the query node, efficiently.*

While image retrieval is one of the promising applications of the proposed approach, it can also be used in various other applications such as music recommendation [1], video concept detection [23], and biological analysis [20].

1.2 Contributions

In this paper, we propose *Mogul*, a novel approach that can efficiently find top-k nodes for Manifold Ranking. In order to reduce search cost, we (1) exploit sparse matrices to compute the ranking scores of selected nodes, and (2) prune low score nodes by estimating upper bounding scores. Our approach has the following attractive characteristics:

- **Efficient**: The computation cost of the proposed approach is $O(n)$, i.e. linear with respect to the number of images, while the inverse matrix approach requires cubic time $O(n^3)$ (Section 4.5). Our experiments demonstrate that the proposed approach is seven orders of magnitude faster than the inverse matrix approaches (Section 5.1).
- **High accuracy**: Although the proposed approach finds the approximate top-k nodes, it is much more accurate than the previous approximation approach [21]; the precision of our approach in finding semantically similar images is higher than 90% (Section 5.2.1). Furthermore, our approach provides users with the additional option of finding the top-k nodes exactly, thus matching the inverse matrix approach (Section 4.6).
- **Small memory**: The proposed approach needs $O(n)$ space. Mogul requires less memory space than the inverse matrix approach, $O(n^2)$ (Section 4.5). This indicates that our approach can handle large-scale image datasets efficiently and effectively.
- **Parameter-free**: Mogul does not require the user to set any inner-parameters (Section 4.4) unlike the previous approaches which impose a trade-off between computation time and approximation accuracy (Section 5.2.1). Our approach provides a simpler implementation of Manifold Ranking for image retrieval.

The theoretical search cost of Mogul, $O(n)$, is independent of the number of answer nodes; this indicates that Mogul is theoretically faster than the inverse matrix approach even if it computes the ranking scores of all nodes for a query node.

Even though Manifold Ranking has been known to improve the quality of image retrieval, it has been difficult to apply it to large-scale multimedia databases due to the high computation cost. The proposed approach, however, can handle large data sets efficiently and so will improve the effectiveness of future image retrieval systems.

The remainder of this paper is organized as follows: Section 2 describes related work. Section 3 overviews the background. Section 4 introduces the main ideas and details our approach. Section 5 reviews the results of our experiments. Section 6 provides our conclusions.

2. RELATED WORK

Manifold Ranking is a graph-based ranking algorithm. Unlike other graph-based ranking algorithms [4, 5, 13], since Manifold Ranking can effectively capture the manifold structures present in $\mathcal{L}^p$ spaces, it is successfully being applied in image retrieval as a distance metric defined on the manifold. The theoretical differences of Manifold Ranking from the other graph-based ranking algorithms are described in detail in the original paper of Manifold Ranking [26]. Several approaches have been proposed to raise the computation efficiency of Manifold Ranking.

Zhou et al. exploited the iterative method to enhance the computation speed of Manifold Ranking [26]. The iterative method updates scores by a given equation until convergence; their approach recursively updates the ranking score of each node by utilizing all edges in the k-NN graph. Even though it iteratively updates scores until convergence, one common practice is to fix the number of iterations or to prespecify some termination condition; the scores after termination differ from the theoretical ones. That is, their approach approximately computes the ranking scores. If t is the number of iterations, their approach needs $O(nt)$ time since the number of edges in a k-NN graph is $O(n)$.

FMR, presented by He et al., is a fast approximation algorithm for Manifold Ranking [8]. Their approach takes advantage of the block-wise structure and linear correlations in the adjacent matrix of the k-NN graph. In a precomputation process, they partition the graph by spectral clustering [3]. Next, for the adjacency matrix, they use a low-rank approximation such as SVD to effectively approximate the graph. FMR significantly outperforms the iterative approach by Zhou et al. [26] in terms of computation time. If spectral clustering is effective in partitioning the graph and there are no edges between partitions, the time complexity of FMR is $O(n^2/N)$ where N is the number of clusters. This is because the adjacency matrix is partitioned into N blocks of $n/N \times n/N$ size. However, the spectral clustering used in FMR is essentially a balanced (normalized) cut, so it may not work well for those datasets whose distributions of partition sizes are highly unbalanced. If the graph is not well partitioned by spectral clustering, their approach must hold the $n \times n$ matrix in the worst case. This indicates that the time complexity of FMR is $O(n^3)$.

EMR is the state-of-the-art approximation approach by Xu et al.; it precomputes an anchor graph to enhance the computation speed of Manifold Ranking [21]. The anchor graph approximately represents each node on the manifold as a linear combination of weights to nearby anchor points. Anchor points are selected from the data points by using the k-means algorithm. They compute weights to anchor points of each data point from Nadaraya-Watson kernel regression with the Epanechnikov quadratic kernel. Since the data points outnumber the anchor points, the anchor graph

Table 1: Definition of main symbols.

Symbol	Definition
n	Number of nodes in the k-NN graph
u_i	i-th node
u_q	Query node
$u_i^{\prime}$	i-th node after node permutation
$x_i^{\prime}$	Approximate score of node $u_i^{\prime}$
$\overline{x}_i^{\prime}$	Upper bounding estimation of score $x_i^{\prime}$
$\overline{x}_{\mathbb{C}_i}^{\prime}$	Upper bounding estimation of cluster $\mathbb{C}_i$
N	Number of clusters in the graph
N_i	Number of nodes in the i-th cluster
c_i	Lowest node number in cluster $\mathbb{C}_i$
$\mathbf{x}$	$n \times 1$ ranking score vector
$\mathbf{q}$	$n \times 1$ query node vector, $q_q = 1$ and 0 for others
$\mathbf{A}$	$n \times n$ adjacency matrix of the k-NN graph
$\mathbf{P}$	$n \times n$ node permutation matrix
$\mathbf{L}$	Lower triangular matrix
$\mathbf{D}$	Diagonal matrix
$\mathbf{U}$	Upper triangular matrix
$\mathbb{C}_i$	i-th cluster

can be regarded as a low-rank approximation of the adjacency matrix. Therefore, they rewrite the equation of score computation by the low-rank approximation by using the Woodbury formula. As described in their paper [21], the computation cost of EMR is $O(nd + d^3)$ and memory cost is $O(nd)$ where d is the number of anchor points. Their experiments showed that EMR is superior to the iterative approach [26] and FMR [8] in terms of computation speed and approximation quality.

3. PRELIMINARY

We formally define the notations and introduce the background of this paper. Table 1 lists the main symbols and their definitions. In Manifold Ranking, a dataset is modeled as a k-NN graph [3]. Each node in this graph represents a data point, and two nodes are connected by an undirected edge if they are k-nearest neighbors. The number of k-nearest neighbors is usually set to 5-20 [10], and there is no loop in the k-NN graph [26].

The ranking task can be formulated as follows: given a set of nodes $\mathbb{U} = \{u_1, u_2, \dots, u_n\} \subset \mathcal{R}^m$ and assuming $u_q \in \mathbb{U}$ is the query node; rank the nodes according to their scores to the query node. Let $\mathbf{A} \in \mathcal{R}^{n \times n}$ be the adjacency matrix of the k-NN graph. In the k-NN graph, the number of edges is $O(n)$ and $\mathbf{A}$ is symmetric. Normally, edge weight can be defined by the heat kernel [21]; $A_{ij} = \exp\{-\mathbf{d}^2(u_i, u_j)/2\sigma^2\}$ if there is an edge linking node u_i and u_j otherwise $A_{ij} = 0$. Function $\mathbf{d}(u_i, u_j)$ is a distance metric of u_i and u_j defined on $\mathbb{U}$ in $\mathcal{L}^p$ spaces, usually the Euclidean distance [26]; σ is the standard variation of the function scores. Note that sum of each row/column in the adjacency matrix can be smaller or larger than 1 from the definition. Let $\mathbf{q}$ be the $n \times 1$ column vector of zeros with the element corresponding to query node u_q set to 1, i.e., $q_q = 1$. In addition, $\mathbf{x} : \mathbb{U} \to \mathcal{R}$ is a ranking function that assigns ranking score x_i to node u_i. In Manifold Ranking, the ranking scores are defined as the optimal solution that minimizes the cost function. The cost function of $\mathbf{x}$ is defined as follows [25]:

$$\mathbf{f}(\mathbf{x}) = \tfrac{1}{2}\textstyle\sum_{ij=1}^{n} A_{ij} \| x_i/\sqrt{C_{ii}} - x_j/\sqrt{C_{jj}} \|^2 + (\tfrac{1}{\alpha} - 1) \textstyle\sum_{i=1}^{n} \| x_i - q_i \|^2 \quad (1)$$

In Equation (1), $\mathbf{C}$ is a diagonal matrix where $C_{ii} = \sum_{j=1}^{n} A_{ij}$, and α is a constant parameter $0 < \alpha < 1$. The first and second terms correspond to the smoothness and the fitting constraint, respectively. The smoothness constraint ensures that nearby nodes have close ranking scores. The fitting constraint ensures that the ranking result fits the query node. The optimal ranking result is achieved when $\mathbf{f}(\mathbf{x})$ is minimized: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} \mathbf{f}(\mathbf{x})$. Differentiating $\mathbf{f}(\mathbf{x})$ with respect to $\mathbf{x}$, $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\big|_{\mathbf{x}=\mathbf{x}^*} = \mathbf{x}^* - \mathbf{C}^{-\frac{1}{2}}\mathbf{A}\mathbf{C}^{-\frac{1}{2}}\mathbf{x}^* + \left(\frac{1}{\alpha} - 1\right)(\mathbf{x}^* - \mathbf{q}) = 0$. Therefore, $(\mathbf{I} - \alpha\mathbf{C}^{-\frac{1}{2}}\mathbf{A}\mathbf{C}^{-\frac{1}{2}})\mathbf{x}^* = (1-\alpha)\mathbf{q}$ where $\mathbf{I}$ is the identity matrix. Consequently,

$$\mathbf{x}^* = (1-\alpha)(\mathbf{I} - \alpha\mathbf{C}^{-\frac{1}{2}}\mathbf{A}\mathbf{C}^{-\frac{1}{2}})^{-1}\mathbf{q} \quad (2)$$

In Equation (2), $(\mathbf{I} - \alpha\mathbf{C}^{-\frac{1}{2}}\mathbf{A}\mathbf{C}^{-\frac{1}{2}})^{-1}$ is inverse matrix of $(\mathbf{I} - \alpha\mathbf{C}^{-\frac{1}{2}}\mathbf{A}\mathbf{C}^{-\frac{1}{2}})$. Equation (2) indicates that the ranking score computation in Manifold Ranking involves the matrix inversion operation. However, this approach is not efficient in terms of computation time if the top-k nodes are the target. This is because it takes $O(n^3)$ time to obtain the inverse matrix [8, 21]. In addition, to find top-k nodes, the ranking scores of all nodes must be computed even though the scores of low ranking nodes are not needed in top-k search. Moreover, this approach requires $O(n^2)$ space to hold the inverse matrix [8, 16]. Therefore, a fast and memory efficient solution is essential for large-scale image retrieval [21].

4. PROPOSED METHOD

This section presents our approach, Mogul; it efficiently finds top-k nodes for Manifold Ranking. Section 4.1 overviews the ideas that underlie Mogul. That is followed by a full description in Sections 4.2, 4.3, and 4.4. We also theoretically analyze its performance in Section 4.5. We show that the proposed approach can exactly identify top-k nodes, mirroring the inverse matrix approach, by slightly modifying the algorithm in Section 4.6. In Section 4, we follow the original paper of Manifold Ranking by assuming that the query data point is in the database [25]; the query node is assumed to be included in the graph. However, in a real application, user would be able to select a query from outside the database [7]. Section 4.6 describes how we handle outside queries.

4.1 Main Ideas

Since the inverse matrix approach needs $O(n^3)$ time, Manifold Ranking does not scale for large datasets. In order to increase the search speed of Manifold Ranking, our approach, Mogul, exploits the following two ideas: (1) It computes the approximate scores from the sparse matrices obtained by *Incomplete Cholesky factorization* [15], and (2) It prunes unnecessary approximate computations by estimating the upper bounding scores to avoid computing the approximate scores of all nodes. Since each sparse matrix has $O(n)$ non-zero elements, we drastically reduce the computation cost to $O(n)$ from the $O(n^3)$ of the inverse matrix approach; the cost is linear with respect to the number of nodes. In addition, our approach achieves higher accuracy than the state-of-the-art approximation approach, EMR, in terms of finding similar images since our approach reduces the approximation error. Moreover, our approach does not need any user-defined inner-parameters, whereas EMR requires the number of anchor points, d, to be set, which induces a trade-off between computation speed and approximation quality. That is, our approach is user friendly.

4.2 Approximate Score Computation

We describe here our approach to efficiently computing the approximate scores for the query node. Section 4.2.1

describes how the ranking scores can be rewritten by Incomplete Cholesky factorization and thus inverse matrix computation can be avoided in score computation. Section 4.2.2 describes that enhancing the approximation quality is an **NP**-complete problem, and shows our efficient and effective solution to this problem. Finally, we describe our approach to computing the ranking scores of selected nodes by using manifold structures in Section 4.2.3.

4.2.1 Matrix Factorization

Since the ranking score computations of Manifold Ranking involve the inverse matrix operation with size of $n \times n$, the inverse matrix approach requires $O(n^3)$ time as described in Section 3. In order to improve the search speed, we exploit Incomplete Cholesky factorization instead of the inverse matrix for ranking score computations. Incomplete Cholesky factorization decomposes a matrix into the product of a lower triangular matrix, a diagonal matrix, and an upper triangular matrix, where the upper triangular matrix is the transpose of the lower triangular matrix. The ranking score is obtained by exploiting forward substitution for the lower triangular matrix and, analogously, back substitution for the upper triangular matrix [16]. Incomplete Cholesky factorization has the property that the factorized matrices have $O(n)$ non-zero elements [15]. This implies that (1) we can significantly reduce the search cost to $O(n)$, and (2) the proposed approach needs $O(n)$ space which is much smaller than the $O(n^2)$ required by the inverse matrix approach.

This section shows how the definition of ranking scores can be rewritten by Incomplete Cholesky factorization. In our approach, we permute nodes in the graph to enhance the approximation quality. Let $\mathbf{P}$ be a permutation matrix, the adjacency matrix of k-NN graph $\mathbf{A}$ is transformed into matrix $\mathbf{A}'$ in the form of $\mathbf{A}' = \mathbf{P}\mathbf{A}\mathbf{P}^T$ where matrix $\mathbf{P}^T$ is the transpose of $\mathbf{P}$ [16]. Similarly, the diagonal matrix $\mathbf{C}$ is transformed as $\mathbf{C}' = \mathbf{P}\mathbf{C}\mathbf{P}^T$. Let u'_i be the i-th node after node permutation. The $n \times n$ permutation matrix $\mathbf{P}$ is an orthogonal matrix where every row and column contains precisely a single 1 with 0s everywhere else, and $P_{ij} = 1$ indicates that the j-th row is permuted into the i-th row. Details of how to obtain matrix $\mathbf{P}$ are shown in Section 4.2.2.

By utilizing matrix $\mathbf{P}$, the ranking score computation defined in Equation (2) can be rewritten in the following matrix form since $\mathbf{I} = \mathbf{P}^T\mathbf{I}\mathbf{P}$ and $\mathbf{P}^T = \mathbf{P}^{-1}$ [16]:

$$\begin{aligned}\mathbf{x}^* &= (1-\alpha)(\mathbf{I} - \alpha\mathbf{C}^{-\frac{1}{2}}\mathbf{A}\mathbf{C}^{-\frac{1}{2}})^{-1}\mathbf{q} \\ &= (1-\alpha)\{\mathbf{P}^T\mathbf{I}\mathbf{P} - \alpha\mathbf{P}^T(\mathbf{C})^{-\frac{1}{2}}\mathbf{P}\mathbf{P}^T\mathbf{A}\mathbf{P}\mathbf{P}^T(\mathbf{C})^{-\frac{1}{2}}\mathbf{P}\}^{-1}\mathbf{q} \\ &= (1-\alpha)\mathbf{P}^T\{\mathbf{I} - \alpha(\mathbf{C}')^{-\frac{1}{2}}\mathbf{A}'(\mathbf{C}')^{-\frac{1}{2}}\}^{-1}\mathbf{P}\mathbf{q}\end{aligned} \quad (3)$$

Since matrix $\mathbf{I}$, $\mathbf{C}'$, and $\mathbf{A}'$ are all symmetric, matrix $\{\mathbf{I} - \alpha(\mathbf{C}')^{-\frac{1}{2}}\mathbf{A}'(\mathbf{C}')^{-\frac{1}{2}}\}$ is also symmetric. We utilize Incomplete Cholesky factorization to approximate the matrix. Incomplete Cholesky factorization decomposes a matrix into three matrices; lower triangular matrix $\mathbf{L}$, diagonal matrix $\mathbf{D}$, and upper triangular matrix $\mathbf{U}$ ($= \mathbf{L}^T$). More specifically, we compute $\mathbf{L}\mathbf{D}\mathbf{L}^T = \mathbf{L}\mathbf{D}\mathbf{U} \approx \{\mathbf{I} - \alpha(\mathbf{C}')^{-\frac{1}{2}}\mathbf{A}'(\mathbf{C}')^{-\frac{1}{2}}\}$. The approximate scores can be obtained by exploiting forward substitution and back substitution [16]. Let $\mathbf{L}' = \mathbf{L}\mathbf{D}$, $\mathbf{x}' = \mathbf{P}\mathbf{x}$, and $\mathbf{q}' = (1-\alpha)\mathbf{P}\mathbf{q}$ where vector $\mathbf{x}$ represents the approximate score vector. From equation (3), we have $\mathbf{L}'\mathbf{y} = \mathbf{q}'$ where $\mathbf{U}\mathbf{x}' = \mathbf{y}$. Since matrix $\mathbf{L}'$ is a lower triangular matrix, we can compute the elements of $n \times 1$ vector $\mathbf{y}$ by using forward substitution for $\mathbf{L}'\mathbf{y} = \mathbf{q}'$ as follows [16]:

$$y_i = \begin{cases} q'_i / L'_{ii} & (i = 1) \\ \left(q'_i - \sum_{j=1}^{i-1} L'_{ij} y_j\right)/L'_{ii} & (i \neq 1)\end{cases} \quad (4)$$

Similarly, the elements of $n \times 1$ vector $\mathbf{x}'$ can be obtained by back substitution for $\mathbf{U}\mathbf{x}' = \mathbf{y}$ as follows [16]:

$$x'_i = \begin{cases} y_i / U_{ii} & (i = n) \\ \left(y_i - \sum_{j=i+1}^{n} U_{ij} x'_j\right)/U_{ii} & (i \neq n)\end{cases} \quad (5)$$

Forward substitution starts with y_1 and, having solved for it, the result is used to solve y_2, and so on. Back substitution proceeds backwards, first x'_n is computed then that is substituted back to solve x'_{n-1}, with repetition for x'_1. Since matrix product $\mathbf{P}\mathbf{x}$ ($= \mathbf{x}'$) is the row permuted vector of $\mathbf{x}$, approximate scores of node u_i, x_i, can be easily obtained from an element of vector $\mathbf{x}'$ as $x_i = P^T_{ij} x'_j$. This indicates that vector $\mathbf{x}'$ corresponds to the approximate scores; element x'_j corresponds to the approximate score of node u'_j after node permutation.

Equations (4) and (5) imply that we can compute the approximate scores for query node u_q if we precompute the Incomplete Cholesky factorization of the matrix $\{\mathbf{I} - \alpha(\mathbf{C}')^{-\frac{1}{2}}\mathbf{A}'(\mathbf{C}')^{-\frac{1}{2}}\}$. We provide the following lemma to show the time complexity of our approximation:

Lemma 1 (Score computation cost). *We need $O(n)$ time to compute the approximate scores of all nodes by using Incomplete Cholesky factorization.*

Proof Since (1) $\mathbf{C}'$ and $\mathbf{I}$ are diagonal matrices of size $n \times n$, and (2) node-permuted matrix $\mathbf{A}'$ clearly has $O(n)$ non-zero elements, the number of non-zero elements of matrix $\{\mathbf{I} - \alpha(\mathbf{C}')^{-\frac{1}{2}}\mathbf{A}'(\mathbf{C}')^{-\frac{1}{2}}\}$ is $O(n)$. In addition, Incomplete Cholesky factorization has the property that the factorized matrices hold $O(n)$ non-zero elements [15]. Therefore, the resulting matrices $\mathbf{L}$, $\mathbf{D}$, and $\mathbf{U}$ all have $O(n)$ non-zero elements. Since matrix $\mathbf{D}$ is diagonal and $\mathbf{L}' = \mathbf{L}\mathbf{D}$, the number of non-zero elements in matrix $\mathbf{L}'$ is also $O(n)$. As shown in Equation (4) and (5), forward and back substitutions compute the elements of vector $\mathbf{x}'$ by exploiting each element in matrix $\mathbf{L}$ and $\mathbf{U}'$ only once. Therefore, forward and back substitutions need $O(n)$ time. Since the approximate score of a node can be obtained from the corresponding element of vector $\mathbf{x}'$, it requires $O(n)$ time to compute the approximate scores of all nodes. □

As described in Section 3, the exact ranking scores can be obtained as the optimal solution that minimizes the cost function defined by Equation (1). However, the solution involves the high computation cost of $O(n^3)$. Lemma 1 indicates that we can drastically reduce the computation cost from $O(n^3)$ to $O(n)$, which is linear with respect to the number of images. In the next section, we introduce our optimization approach that enhances the approximation quality by properly permuting the nodes in the graph.

4.2.2 Optimization

The previous section proposed the approximation approach to exploit Incomplete Cholesky factorization where we compute the approximation as $\mathbf{L}\mathbf{D}\mathbf{U} \approx \{\mathbf{I} - \alpha(\mathbf{C}')^{-\frac{1}{2}}\mathbf{A}'(\mathbf{C}')^{-\frac{1}{2}}\}$. However, the obtained ranking scores are different from those yielded by the inverse matrix approach. To enhance the approximation quality, we permute the nodes of the adjacency matrix before Incomplete Cholesky factorization. In other words, we permute rows/columns in adjacency matrix $\mathbf{A}$ to reduce the approximation error. By properly permuting the nodes, we can reduce the approximation error. Unfortunately, determining the node permutation that reduces the approximation error is an $\mathbf{NP}$-complete problem.

Theorem 1 (Permutation problem). *Setting the node permutation that minimizes the approximation error in Incomplete Cholesky factorization is* $\mathbf{NP}$*-complete.*

Proof We prove the theorem by a reduction from the *minimum fill-in* problem [22]. We transform instances of the minimum fill-in problem into instances of the permutation problem as follows: for the graph of the minimum fill-in problem, we create the adjacency matrix $\mathbf{A}$. For node elimination ordering, we create the permutation matrix $\mathbf{P}$, and then the approximation error for the chordal graph. As a result, it is easy to show that there exists a solution to the minimum fill-in problem that has the minimum number of edge additions if and only if there exists a solution to the permutation problem with minimum approximation error in Incomplete Cholesky factorization. Therefore, the permutation problem is trivial in $\mathbf{NP}$. □

Before detailing our solution, we here explain why Incomplete Cholesky factorization is called "incomplete". Let $\mathbf{W} = \{\mathbf{I} - \alpha(\mathbf{C}')^{-\frac{1}{2}}\mathbf{A}'(\mathbf{C}')^{-\frac{1}{2}}\}$. Matrix $\mathbf{L}$ ($= \mathbf{U}^T$) and $\mathbf{D}$ can be computed as follows [15]:

$$L_{ij} = \begin{cases} 0 & (i<j, i>j \cap W_{ij}=0) \\ 1 & (i=j) \\ \big(W_{ij} - \sum_{k=1}^{j-1} L_{ik}L_{jk}D_{kk}\big)/D_{jj} & (i>j \cap W_{ij} \neq 0) \end{cases} \quad (6)$$

$$D_{ij} = \begin{cases} 0 & (i \neq j) \\ W_{ij} - \sum_{k=1}^{j-1} L_{jk}^2 D_{kk} & (i=j) \end{cases} \quad (7)$$

Equation (6) indicates that, if an element in matrix $\mathbf{W}$ is zero, the corresponding element in $\mathbf{L}$ is also zero, i.e., $L_{ij} = 0$ if $W_{ij} = 0$. This is the origin of *incomplete*; matrix $\mathbf{L}$ is limited to a particular sparsity pattern, the same as the pattern of matrix $\mathbf{W}$. Without this limitation, we can exactly compute the ranking scores from Equation (6) and (7) [15]. This implies that approximation error is expected to be large as many elements are forced to be zero in Incomplete Cholesky factorization. In addition, Equation (6) and (7) indicate that elements in matrix $\mathbf{L}$ and $\mathbf{D}$ are obtained from the left-side elements in the matrices. Therefore, elements in the matrices are likely to be zero as left-side elements are sparse; if all the left-side elements are zero, the elements must be zero. Our approach is based on these observations; we reduce the numbers of non-zero elements that are forced by the limitation to be zero by letting the left-side elements of matrix $\mathbf{W}$ be sparse.

Algorithm 1 shows the permutation algorithm that enhances the approximation quality by obtaining the left-side sparse matrices. We exploit the property that the manifolds are locally separated and multiple data points in each manifold are locally connected to each other [26]. In this algorithm, N is the number of clusters in the graph, N_i is the number of nodes in the i-th cluster $\mathbb{C}_i$, and $e(u)$ is the number of within-cluster edges of node u. The algorithm first initializes permutation matrix $\mathbf{P}$ to a zero matrix (line 1). It computes the clusters of the graph by the state-of-the-art clustering approach by Shiokawa et al. [17] (line 2). Their approach is more efficient and effective than other clustering approaches as reported in [17]. It divides the graph into clusters so as to increase the number of within-cluster edges. In other words, their approach is expected to reduce the number of cross-cluster edges. Note that the number of clusters is automatically determined. Our approach then removes all the nodes that have cross-cluster edges and appends them to the N-th cluster $\mathbb{C}_N$ (lines 3-7). As a result, only nodes included in $\mathbb{C}_N$ have cross-cluster edges. That is, a node must have only within-cluster edges if the node is included in $\mathbb{C}_i$; $i = 1, 2, \dots, N-1$. It finally selects clusters one by one from $\mathbb{C}_1$ to $\mathbb{C}_N$ and determines the elements in the permutation matrix by arranging nodes in each cluster based on the numbers of within-cluster edges (lines 8-17). Since the algorithm arranges nodes in ascending order of within-cluster edges, the left-side elements in matrix $\mathbf{W}$ are expected to be sparse. As a result, we can reduce the numbers of elements forced to be zero as set by Incomplete Cholesky factorization. Thus, our permutation approach can enhance the approximation quality.

Algorithm 1 Optimization

```
Input: given graph
Output: permutation matrix
 1: $\mathbf{P} = \mathbf{0}$;
 2: divide the graph into clusters $\mathbb{C}_1, \mathbb{C}_2, \dots, \mathbb{C}_{N-1}$ by the graph clustering approach by Shiokawa et al. [17];
 3: create new empty cluster $\mathbb{C}_N = \emptyset$;
 4: for $i = 1$ to $N-1$ do
 5:     remove nodes that have cross-cluster edges from $\mathbb{C}_i$;
 6:     append the removed nodes to $\mathbb{C}_N$;
 7: end for
 8: $k = 1$;
 9: for $i = 1$ to $N$ do
10:     $\mathbb{P} = \emptyset$;
11:     for $j = 1$ to $N_i$ do
12:         $u_l = \text{argmin}(e(u) | u \in \mathbb{C}_i \backslash \mathbb{P})$;
13:         $P_{kl} = 1$;
14:         append node $u_l$ to $\mathbb{P}$;
15:         $k = k + 1$;
16:     end for
17: end for
18: return $\mathbf{P}$;
```

While the permutation approach is designed to reduce the approximation error, this approach has an additional advantage; we can avoid computing the approximate scores of all nodes by skipping unnecessary nodes. The next section shows this advantage in detail. The following lemma describes the cost of obtaining the matrices:

Lemma 2 (Precomputing cost). *Our approach needs* $O(n)$ *time and* $O(n)$ *space to compute matrices* $\mathbf{L}$ *and* $\mathbf{D}$.

Proof In order to obtain the node permutated matrices, we first exploit Algorithm 1 to compute matrix $\mathbf{P}$ and then apply Incomplete Cholesky factorization for the graph. The computation cost of Algorithm 1 is $O(n)$. This is because (1) the computation cost of the graph clustering approach is linear to the number of edges in the graph [17], (2) we check all the edges to obtain the node permutation matrix after dividing the graph into clusters, and (3) the number of edges is $O(n)$ in the k-NN graph. In addition, as shown in Equation (6) and (7), elements in matrix $\mathbf{L}$ and $\mathbf{D}$ can be computed from the left-side elements in the matrices. Since the number of edges connected to a node is constant in the k-NN graph, the computation cost to obtain matrix $\mathbf{L}$ and $\mathbf{D}$ is linear with respect to the number of nodes in Equation (6) and (7). In addition, the memory cost to hold matrix $\mathbf{L}$ and $\mathbf{D}$ is $O(n)$ since the numbers of non-zero elements in matrix $\mathbf{L}$ and $\mathbf{D}$ are $O(n)$. As a result, we need $O(n)$ time and $O(n)$ space to obtain matrix $\mathbf{L}$ and $\mathbf{D}$. □

This lemma indicates that we can efficiently compute matrix $\mathbf{L}$ and $\mathbf{D}$ from the given graph. Note that all the processes to compute matrix $\mathbf{L}$ and $\mathbf{D}$ are independent of the query node. Therefore, we can precompute matrices $\mathbf{L}$ and $\mathbf{D}$ before commencing the search process; we can flexibly set query node and the number of answers in accordance with user demand after the precomputing process.

4.2.3 Skipping Unnecessary Computations

As described in Section 4.2.1, we can find the answer nodes by applying Incomplete Cholesky factorization. This approach employs forward and back substitutions for the factorized matrix. However, since previously computed scores are needed in forward and back substitutions, this approach cannot limit the score computations to selected nodes. This section introduces an approach that computes the approximate score of selected nodes to efficiently find top-k nodes.

As we discuss in this section, matrix $\mathbf{L}$ has a particular non-zero pattern due to the property of underlying manifold structures in real datasets [26]. In order to realize efficient search, we utilize the particular non-zero pattern to skip the approximate score computations that are unnecessary in obtaining the scores of selected nodes. This implies that we can efficiently find answer nodes by using manifold structures. In this section, we first describe that matrix $\mathbf{L}$ has the particular non-zero pattern. Next, we show a particular non-zero pattern of vector $\mathbf{y}$. We then introduce our approach to computing the approximate score of selected nodes. In this section, we assume that node u'_q corresponds to the query node after node permutation and node u'_q is included in cluster $\mathbb{C}_Q$. Note that $\mathbb{C}_Q = \mathbb{C}_N$ if node u'_q is included in cluster $\mathbb{C}_N$. We introduce the following property of matrix $\mathbf{L}$ yielded by the permutation approach:

Lemma 3 (Zero elements in matrix $\mathbf{L}$). *In matrix* $\mathbf{L}$*,* $L_{ij} = 0$ *if (1) node* u'_i *and* u'_j *lie in different clusters and (2) neither node* u'_i *nor* u'_j *are included in* $\mathbb{C}_N$*.*

Proof If node u'_i is not included in cluster $\mathbb{C}_N$, the node must have only within-cluster edges as described in Section 4.2.2. Therefore, if nodes u'_i and u'_j are included in different clusters, we have $W'_{ij} = 0$ in matrix $\mathbf{W}$. Since matrices $\mathbf{W}$ and $\mathbf{L}$ have the same sparsity pattern due to the property of Incomplete Cholesky factorization, we have $L_{ij} = 0$ if the corresponding element, $W_{ij}, = 0$, which completes the proof. □

Lemma 3 indicates that elements among the first $N-1$ clusters must be 0 in matrix $\mathbf{L}$; matrix $\mathbf{L}$ has non-zero pattern of singly bordered block diagonal [16]. In Section 5.2.2, we show examples of matrix $\mathbf{L}$ in real datasets. In addition, by exploiting Lemma 3, we can suppress the precomputation time by pruning unnecessary computations in matrix $\mathbf{L}$ as demonstrated in Section 5.2.4. We utilize the following property of vector $\mathbf{y}$ which is derived from Lemma 3:

Lemma 4 (Zero elements in vector $\mathbf{y}$). *If node* u'_i *is not included in cluster* $\mathbb{C}_Q$ *or* $\mathbb{C}_N$*, the corresponding element in vector* $\mathbf{y}$ *must be zero, i.e.,* $y_i = 0$ *if* $u'_i \notin \mathbb{C}_Q \cup \mathbb{C}_N$*.*

Proof The elements of vector $\mathbf{y}$ can be obtained by applying forward substitution for $\mathbf{L}'\mathbf{y} = \mathbf{q}'$. Since $\mathbf{L}' = \mathbf{LD}$ and $\mathbf{D}$ is a diagonal matrix, matrix $\mathbf{L}'$ has the same non-zero pattern as matrix $\mathbf{L}$. In addition, the q-th element of vector $\mathbf{q}'$ is a non-zero element; all other elements are 0.

If $u'_q \in \mathbb{C}_N$, we have $q'_i = 0$ for $u'_i \notin \mathbb{C}_N$. Therefore, $y_1 = 0$ from Equation (4). By recursive substitution into Equation (4), we have $y_i = 0$ for $u'_i \notin \mathbb{C}_N$. Otherwise (i.e., $u'_q \notin \mathbb{C}_N$), we similarly have $y_i = 0$ for $u'_i \in \mathbb{C}_j$ such that $j = 1, 2, \ldots, Q-1$ from Equation (4). That is, if node u'_i is included in a lower number cluster than cluster $\mathbb{C}_Q$, the corresponding element in vector y_i equals 0. Furthermore, we have $y_i = 0$ if node u'_i is included in cluster $\mathbb{C}_j$ such that $j = Q+1, Q+2, \ldots, N-1$. This is because (1) the corresponding elements in vector $\mathbf{q}'$ are zero and (2) elements in matrix $\mathbf{L}$ are zero between cluster $\mathbb{C}_j$ and $\mathbb{C}_Q$ from Lemma 3. That completes the proof. □

Lemma 4 reveals the property of vector $\mathbf{y}$ that non-zero elements in vector $\mathbf{y}$ are restricted only in clusters $\mathbb{C}_Q$ and $\mathbb{C}_N$. That is, an element that corresponds to node u'_i must be zero in vector $\mathbf{y}$ if node u'_i is not included cluster $\mathbb{C}_Q$ or $\mathbb{C}_N$. By utilizing this property, we can efficiently compute the elements in vector $\mathbf{y}$. Similarly, Lemma 3 yields the following property of the approximate score vector $\mathbf{x}'$:

Lemma 5 (Independency in vector $\mathbf{x}'$). *If we have the elements of vector* $\mathbf{x}'$ *in cluster* $\mathbb{C}_N$*, the elements of vector* $\mathbf{x}'$ *in cluster* $\mathbb{C}_i$ *such that* $i = 1, 2, \ldots, N-1$ *can be computed without recourse to the elements of vector* $\mathbf{x}'$ *in* $\mathbb{C}_j$ *such that* $j = i+1, i+2, \ldots, N-1$ *even though back substitution is applied.*

Proof We assume that nodes u'_i and u'_j are included in cluster $\mathbb{C}_i$ and $\mathbb{C}_j$, respectively. The elements of vector $\mathbf{x}'$ are obtained by applying back substitution for $\mathbf{U}\mathbf{x}' = \mathbf{y}$. Since we have $\mathbf{U} = \mathbf{L}^T$, $U_{ij} = 0$ for node u'_i and u'_j from Lemma 3. Therefore, it is clear from Equation (5) that element x'_i, which corresponds to node u'_i, can be computed without element x'_j by back substitution. □

Lemma 5 indicates that, if we compute the elements of vector $\mathbf{x}'$ in cluster $\mathbb{C}_N$, we can compute the elements of vector $\mathbf{x}'$ for arbitrarily selected clusters.

As shown in Lemma 4, vector $\mathbf{y}$ has a particular non-zero pattern. Therefore, in order to compute the approximate scores of selected nodes after node permutation, we first compute the scores of vector $\mathbf{y}$ for cluster $\mathbb{C}_Q$ and $\mathbb{C}_N$. Next, we compute elements of the approximate score of vector $\mathbf{x}'$ for cluster $\mathbb{C}_N$, and then compute the approximate score of the selected nodes based on Lemma 5. Lemma 4 and 5 validate the effectiveness of these steps in computing the approximate scores of selected nodes.

4.3 Upper Bounding Estimation

In order to perform efficient top-k search, we estimate the ranking scores of a cluster to select answer-likely nodes. If the estimation indicates that the cluster has an answer-likely node, we compute approximate scores for all nodes in the cluster. Otherwise, we prune the cluster, which has low score nodes, without computing the approximate scores. The advantage of the estimation approach lies in finding the top-k nodes exactly in terms of approximate scores even though it uses estimation. This is because we estimate upper bounding approximate scores; answer nodes cannot be pruned by the estimation since they have high estimation scores. As a result, we can safely discard unlikely clusters along with their score estimations. In this section, we formally introduce the estimation approach and show its theoretical aspects. Let $\overline{x}'_{\mathbb{C}_i}$ $(i \neq Q, N)$ be the estimation of cluster $\mathbb{C}_i$, $\overline{x}'_{\mathbb{C}_i}$ is defined as follows:

Definition 1 (UPPER BOUND). *The upper bounding estimation, $\overline{x}'_{\mathbb{C}_i}$, is given as follows for cluster $\mathbb{C}_i$ such that $\mathbb{C}_i \neq \mathbb{C}_Q$ and $\mathbb{C}_i \neq \mathbb{C}_N$:*

$$\overline{x}'_{\mathbb{C}_i} = \overline{X}_i(1+\overline{U}_i)^{N_i-1} \tag{8}$$

In Equation (8), $\overline{X}_i$ and $\overline{U}_i$ is defined as follows:

$$\overline{X}_i = \textstyle\sum_{j=c_N}^{n} \overline{U}_{i:j}|x'_j| \tag{9}$$

$$\overline{U}_i = \max\{|U_{jk}| : u'_j, u'_k \in \mathbb{C}_i \text{ and } u'_j \neq u'_k\} \tag{10}$$

where $|x'_j|$ is the absolute value of x'_j and c_N is the lowest node number in cluster $\mathbb{C}_N$, i.e., $c_N = \min\{k : u'_k \in \mathbb{C}_N\}$. In addition, $\overline{U}_{i:j} = \max\{|U_{kj}| : u'_k \in \mathbb{C}_i\}$ in Equation (9).

Note that we can precompute $\overline{U}_i$ in $O(n)$ time for each cluster since the number of non-zero elements in matrix $\mathbf{U}$ is $O(n)$. Our estimation is designed for all nodes except for cluster $\mathbb{C}_Q$ and $\mathbb{C}_N$. This is because (1) the nodes belonging to cluster $\mathbb{C}_Q$ are expected to have high approximate scores [2] and (2) the approximate scores of cluster $\mathbb{C}_N$ are needed for the score computations of the nodes in other clusters as shown in Lemma 5. In order to describe the theoretical property of the cluster estimation $\overline{x}'_{\mathbb{C}_i}$, we introduce the following estimation of node u'_i:

Definition 2 (NODE ESTIMATION). *Let u'_i be a node such that $u'_i \notin \mathbb{C}_Q \cup \mathbb{C}_N$. If c_i is the lowest node number in cluster $\mathbb{C}_i$ after node permutation, the following equation gives the estimation of node u'_i, $\overline{x}'_i$:*

$$\overline{x}'_i = \begin{cases} \sum_{j=c_N}^{n} \overline{U}_{i:j}|x'_j| & (i = c_i+N_i-1) \\ \overline{U}_i \sum_{j=i+1}^{c_i+N_i-1} |\overline{x}'_j| + \sum_{j=c_N}^{n} \overline{U}_{i:j}|x'_j| & (otherwise) \end{cases} \tag{11}$$

For the estimation $\overline{x}'_i$, we have the following lemma:

Lemma 6 (NODE ESTIMATION). *We have $\overline{x}'_i \geq x'_i$ for node u'_i such that $u'_i \notin \mathbb{C}_Q \cup \mathbb{C}_N$.*

Proof If $u'_i \neq \mathbb{C}_Q \cup \mathbb{C}_N$, we have $y_i = 0$ from Lemma 4. Since $\mathbf{U} = \mathbf{L}^T$ and $L_{jj} = 1$ from Equation (6), we have $U_{jj} = 1$. Therefore, from Equation (5), we have the following equation for node $u'_i \notin \mathbb{C}_Q \cup \mathbb{C}_N$:

$$x'_i = -\textstyle\sum_{j=i+1}^{n} U_{ij}x'_j \tag{12}$$

If node u'_i and u'_j are included in different clusters and $u'_i, u'_j \notin \mathbb{C}_N$, we have $U_{ij} = 0$ from the property of matrix $\mathbf{L}$ (Lemma 3). Therefore, if $i \neq c_i + N_i - 1$, we have

$$\begin{aligned} x'_i &= -\textstyle\sum_{j=i+1}^{c_i+N_i-1} U_{ij}x'_j - \sum_{j=c_i+N_i}^{c_N-1} U_{ij}x'_j - \sum_{j=c_N}^{n} U_{ij}x'_j \\ &= \textstyle\sum_{j=i+1}^{c_i+N_i-1}(-U_{ij})x'_j + \sum_{j=c_N}^{n}(-U_{ij})x'_j \\ &\leq \textstyle\sum_{j=i+1}^{c_i+N_i-1}|U_{ij}||x'_j| + \sum_{j=c_N}^{n}|U_{ij}||x'_j| \\ &\leq \overline{U}_i \textstyle\sum_{j=i+1}^{c_i+N_i-1} |\overline{x}'_j| + \sum_{j=c_N}^{n} \overline{U}_{i:j}|x'_j| = \overline{x}'_i \end{aligned} \tag{13}$$

Similarly, if $i = c_i + N_i - 1$, we have

$$x'_i = -\textstyle\sum_{j=c_i+N_i}^{c_N-1} U_{ij}x'_j - \sum_{j=c_N}^{n} U_{ij}x'_j \leq \sum_{j=c_N}^{n} \overline{U}_{i:j}|x'_j| = \overline{x}'_i \tag{14}$$

which completes the proof. □

Lemma 6 indicates that the estimation of each node gives an upper bounding approximate score. By exploiting the estimation of each node, we show the following property of the estimation of cluster $\mathbb{C}_i$:

Lemma 7 (UPPER BOUND). *For any node that is included in cluster $\mathbb{C}_i$, the corresponding element in vector $\mathbf{x}'$ cannot be larger than the estimation $\overline{x}'_{\mathbb{C}_i}$ of the cluster. That is, $x'_i \not> \overline{x}'_{\mathbb{C}_i}, \forall u'_i \in \mathbb{C}_i$.*

Proof Prior to proving Lemma 7, we prove that the upper bounding estimations of nodes in the same cluster are monotonic decreasing in terms of node numbers, i.e., $\overline{x}'_i \geq \overline{x}'_{i+1}$ if $u'_{i+1}, u'_i \in \mathbb{C}_i$. From Equation (11), $\overline{x}'_i$ clearly has a positive value, therefore,

$$\begin{aligned} \overline{x}'_i &= \overline{U}_i \textstyle\sum_{j=i+1}^{c_i+N_i-1} \overline{x}'_j + \sum_{j=c_N}^{n} \overline{U}_{i:j}|x'_j| \\ &= \overline{U}_i\overline{x}'_{i+1} + \overline{U}_i \textstyle\sum_{j=i+2}^{c_i+N_i-1} \overline{x}'_j + \sum_{j=c_N}^{n} \overline{U}_{i:j}|x'_j| \\ &= (1+\overline{U}_i)\overline{x}'_{i+1} \end{aligned} \tag{15}$$

Since $1 + \overline{U}_i \geq 1$, it is clear that $\overline{x}'_i \geq \overline{x}'_{i+1}$ from Equation (15). Since $\overline{x}'_{c_i+N_i-1}$ corresponds to the highest number node in cluster $\mathbb{C}_i$, we have $\overline{x}'_i \geq \overline{x}'_{c_i+N_i-1} = \overline{X}_i$; the estimation of cluster $\mathbb{C}_i$ cannot smaller than $\overline{X}_i$. The equation of $\overline{x}'_{c_i+N_i-1} = \overline{X}_i$, along with Equation (15), indicates that $\overline{x}'_i$ can be regarded as a geometric progression where $\overline{X}_i$ is the first term and $(1+\overline{U}_i)$ is the common ratio. Since $\overline{x}'_{c_i}$ corresponds to the lowest number node in cluster $\mathbb{C}_i$,

$$\overline{x}'_i \leq \overline{x}'_{c_i} = \overline{X}_i(1+\overline{U}_i)^{N_i-1} = \overline{x}'_{\mathbb{C}_i} \tag{16}$$

From Lemma 6, we have $x'_i \leq \overline{x}'_i$ for cluster $\mathbb{C}_i$. Therefore, we have $x'_i \not> \overline{x}'_{\mathbb{C}_i}, \forall u'_i \in \mathbb{C}_i$. □

Lemma 7 enables us to safely prune unnecessary approximate computations. We describe our top-k search algorithm based on this lemma in Section 4.4. The following lemma indicates the efficiency of the estimation approach:

Lemma 8 (EFFICIENCY OF THE ESTIMATION APPROACH). *If we have elements of vector $\mathbf{x}'$ that correspond to cluster $\mathbb{C}_N$, it needs $O(n)$ time to compute the estimations for clusters such that $\mathbb{C}_i \neq \mathbb{C}_Q$ and $\mathbb{C}_i \neq \mathbb{C}_N$ in the worst case.*

Proof From Equation (8), the estimation of a cluster is computed as $\overline{X}_i(1+\overline{U}_i)^{N_i-1}$. Since the non-zero pattern in matrix $\mathbf{U}$ ($= \mathbf{L}^T$) is the same as matrix $\mathbf{W}$ which is obtained from the given k-NN graph (Equation (6)), it needs $O(n)$ time to compute $\overline{X}_i$ from Equation (9) for the clusters. In addition, since $\overline{U}_i$ can be precomputed, it needs $O(n)$ time to obtain $(1+\overline{U}_i)^{N_i-1}$ for the clusters where the number of such clusters is at most n. Therefore, $O(n)$ time is required to compute the estimations for the clusters. □

Lemma 8 indicates that we can efficiently prune unnecessary score computations in the top-k search process.

4.4 Search Algorithm

Algorithm 2 shows, Mogul, our proposal that finds top-k nodes for Manifold Ranking. In this algorithm, $\mathbb{K}$ and θ are the set of top-k nodes and the lowest approximate score of the top-k nodes, respectively. The algorithm sets θ as 0 (line 1) and initializes top-k nodes set $\mathbb{K}$ by appending dummy nodes that have approximate similarity of 0 (lines 2-3). It next obtains node u'_q from the query node (line 4). Since only the nodes included in cluster $\mathbb{C}_Q$ or $\mathbb{C}_N$ have non-zero elements in vector $\mathbf{y}$ (Lemma 4), it computes the elements of vector $\mathbf{y}$ for the nodes by forward substitution (lines 5-7). As described in Section 4.3, nodes in cluster $\mathbb{C}_Q$ are expected to be answer nodes, and, the approximate scores in cluster $\mathbb{C}_N$ are required to compute the approximate scores

Algorithm 2 Mogul

```
Input: query node
Output: top-k nodes
 1: $\theta = 0$;
 2: $\mathbb{K} = \emptyset$;
 3: append dummy nodes to $\mathbb{K}$;
 4: obtain node $u'_q$ after the node permutation from the query node;
 5: for each $u'_i \in \mathbb{C}_Q \cup \mathbb{C}_N$ do
 6:     compute element $y_i$ of node $u'_i$ by Equation (4);
 7: end for
 8: for each $u'_i \in \mathbb{C}_Q \cup \mathbb{C}_N$ do
 9:     compute score $x'_i$ of node $u'_i$ by Equation (5);
10:     if $x'_i \geq \theta$ then
11:         $v' = \text{argmin}(x'_j | u'_j \in \mathbb{K})$;
12:         append node $u'_i$ to $\mathbb{K}$;
13:         remove node $v'$ from $\mathbb{K}$;
14:         $\theta = \min(x'_j | u'_j \in \mathbb{K})$;
15:     end if
16: end for
17: for each $\mathbb{C}_i$ such that $\mathbb{C}_i \neq \mathbb{C}_Q, \mathbb{C}_N$ do
18:     compute estimation $\overline{x}'_{\mathbb{C}_i}$ by Equation (8);
19:     if $\overline{x}'_{\mathbb{C}_i} \geq \theta$ then
20:         for each $u'_j \in \mathbb{C}_i$ do
21:             compute score $x'_j$ of node $u'_j$ by Equation (5);
22:             if $x'_j \geq \theta$ then
23:                 $v' = \text{argmin}(x'_k | u'_k \in \mathbb{K})$;
24:                 append node $u'_j$ to $\mathbb{K}$;
25:                 remove node $v'$ from $\mathbb{K}$;
26:                 $\theta = \min(x'_k | u'_k \in \mathbb{K})$;
27:             end if
28:         end for
29:     end if
30: end for
31: for each $u'_i \in \mathbb{K}$ do
32:     permute node $u'_i$ by permutation matrix $\mathbf{P}$;
33: end for
34: return $\mathbb{K}$;
```

of selected nodes from Lemma 5. Therefore, it computes the approximate scores of cluster $\mathbb{C}_Q$ and $\mathbb{C}_N$ to update the top-k nodes set (lines 8-16). It next computes the estimation of each cluster in turn (line 18). If the estimation of a cluster is lower than θ, all the approximate scores of the cluster's nodes must be lower than θ from Lemma 7. Therefore, it does not compute the approximate scores for the cluster. Otherwise, the cluster can contain an answer node, so it computes the approximate scores of the cluster's nodes (lines 19-29). It finally obtains the original node number for the answer nodes from the permutation matrix (lines 31-33).

As shown in Algorithm 2, our search algorithm does not require any user-defined inner-parameter. As a result, it provides the user with a simple approach to finding the top-k nodes with enhanced search speed for Manifold Ranking.

4.5 Theoretical Analyses

In this section, we show the computation and memory costs of Mogul. Note that the inverse matrix approach needs $O(n^3)$ time and $O(n^2)$ space. We introduce the following theorem to discuss the time complexity of our approach:

Theorem 2 (Computation cost of Mogul). *Mogul requires $O(n)$ time to find top-k nodes.*

Proof We find the top-k nodes with the following three steps: (1) permute the nodes in the graph by Algorithm 1, (2) compute matrix $\mathbf{L}$ and $\mathbf{D}$ by exploiting Incomplete Cholesky factorization, and (3) find top-k nodes by Algorithm 2. As described in Section 4.2.2, the first and second steps correspond to the precomputing process of our approach. The precomputing process requires $O(n)$ time as shown in Lemma 2. The third step corresponds to the search process of Mogul. In the worst case, we cannot prune any node and so must compute the approximate scores of all nodes. As described in Lemma 8, it needs $O(n)$ time to compute the estimation of the clusters. As shown in Lemma 1, it takes $O(n)$ time to compute the approximate scores of all nodes by forward and back substitutions. Therefore, Mogul takes $O(n)$ time to identify top-k node. □

We have the following theorem for the memory cost:

Theorem 3 (Memory cost of Mogul). *Mogul needs $O(n)$ space to identify top-k nodes.*

Proof As shown in Lemma 2, our approach needs $O(n)$ space in its precomputing process. In order to compute the approximate scores, Mogul holds vectors $\mathbf{q}'$, $\mathbf{y}$, and $\mathbf{x}'$ as described in Section 4.2.1. Since these vectors all have size of $n \times 1$, it takes $O(n)$ space to hold these vectors. For the approximate score computations, Mogul exploits matrices $\mathbf{L}'$, $\mathbf{U}$, and $\mathbf{P}$. Since we utilize Incomplete Cholesky factorization, the numbers of non-zero elements in matrices $\mathbf{L}'$ and $\mathbf{U}$ are both $O(n)$. Moreover, in matrix $\mathbf{P}$, the number of non-zero elements is $O(n)$ even though the size of matrix $\mathbf{P}$ is $n \times n$. In addition, as shown in Definition 2, Mogul uses value $\overline{U}_{i:j}$ and $\overline{U}_i$ for the estimation. Since the number of edges in the k-NN graph is $O(n)$, it needs $O(n)$ space to hold these values. Therefore, Mogul requires $O(n)$ space. □

Theorems 2 and 3 show that Mogul has lower time and space complexities than the inverse matrix approach. The computation cost of Mogul is, in practice, smaller than the theoretical cost of $O(n)$ because of the effectiveness of the estimation approach. In Section 5, the results of extensive experiments confirm the effectiveness of our approach.

4.6 Extension of Mogul

The previous sections (Section 4.1 to 4.5) focused how Mogul approximately finds top-k nodes for the query node that lies in the database. In this section, we briefly describe the extensions of the proposed approach to (1) exactly find top-k nodes same as the inverse matrix approach and (2) handle the query node outside the database.

4.6.1 Exact Top-k Search

In order to exactly identify the top-k nodes, we use Modified Cholesky factorization [15] instead of Incomplete Cholesky factorization. Incomplete Cholesky factorization has the limitation that its sparsity pattern is the same as that of matrix W as described in Section 4.2.2. If we drop this limitation, Incomplete Cholesky factorization is equivalent to Modified Cholesky factorization [15]. Since Modified Cholesky factorization is not an approximation approach as is Incomplete Cholesky factorization, we can use it to compute the exact ranking scores. Let m be the number of non-zero elements in matrix $\mathbf{L}$ obtained by Modified Cholesky factorization, we can exactly find the top-k nodes at $O(m)$ time and $O(m)$ space. While omitting the proof of this property due to space limitations, we note that it is founded on two theoretical properties. The first is that it requires $O(m)$ time to compute the exact scores of all nodes by Modified Cholesky factorization. This is because the number of non-zero elements in matrix $\mathbf{L}$ is $O(m)$. This property corresponds to Lemma 1. The second property is that Lemma 3 holds even if Modified Cholesky factorization is applied; we have

$L_{ij} = 0$ in Modified Cholesky factorization if (1) $u'_i, u'_j \notin \mathbb{C}_N$ and (2) node u'_i and u'_j are in different clusters. This second property implies that subsequent lemmas (Lemma 4 to 8) hold one after the other even if Modified Cholesky factorization is applied. Since our optimization approach is designed to reduce the non-zero elements in matrix $\mathbf{L}$, the extended version is expected to have sparse data structures.

4.6.2 *Out-of-sample Query*

In the previous sections, we assumed that the query node is in the database by following the original paper [26]. However, in real applications, the user can select a query node outside the database (out-of-sample query). This section describes our solution for out-of-sample query.

The naive approach is to compute the k-NN graph by adding the query node and apply Incomplete Cholesky factorization. This approach, however, is not practical especially for large data. Our approach is to set the scores in the query vector $\mathbf{q}$ by utilizing the neighbor nodes. We compute neighbors of the query node and use the neighbors as query nodes. Since this approach does not add the query node to the graph, it can efficiently compute the answer nodes for out-of-sample queries. The theoretical background of this approach is shown in [7]. In order to efficiently find the neighbors, we first compute the nearest cluster by utilizing the average feature of each cluster, and then obtain the neighbors from the nearest cluster. This approach has $O(n)$ time and space complexities since the computational and memory costs of obtaining neighbors for the query node are linear with respect to the number of nodes. This indicates that we can find the answer nodes in $O(n)$ time and space even for out-of-sample queries.

5. EXPERIMENTAL EVALUATION

We performed experiments to demonstrate the effectiveness of our approach. In this section, "Mogul", "EMR", "FMR", "Iterative", and "Inverse" represent the results of our approach, the state-of-the-art approximation approach by Xu et al. [21], the low-rank approximation based approach by He et al. [8], the iterative method by Zhou et al. [26], and the optimal solution based on the inverse matrix computation [25], respectively. Note that EMR is superior to the other approximation approaches in terms of search time and approximation quality as reported in [21]. We used the following standard datasets for image retrieval:

- COIL-100 [14]: This dataset contains images of 100 objects taken in Columbia University. The objects were placed on a turntable that was rotated through 360 degrees to vary object pose. Images of the objects were taken at pose intervals of 5 degrees; 72 poses per object, resulting in 7,200 images. We resized all images to 32×32 and used RGB pixel values as the feature vector, resulting in 3,048 dimensions.

- PubFig [11]: This dataset consists of 58,797 images of 200 famous persons. The images were downloaded from the Internet using the person's name as the search query. We represent each image by using a set of *attributes*, a state-of-the-art semantic image representation framework [11]. Each attribute itself is a semantic description of images relevant to human faces; they were automatically detected by pre-trained classifiers. We used 73 attributes published by [11].

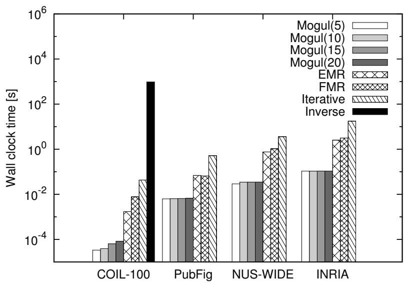


Figure 1: Search time.

- NUS-WIDE [18]: This dataset was collected by randomly downloading photographs from Flickr through its public API. The dataset was created by a research group of National University of Singapore. Images that were too small or had inappropriate length-width ratio were removed yielding 267,465 images. According to the previous paper [18], 150-D color moment was used as the image feature.

- INRIA [9]: This dataset is a large-scale set of image features collected by INRIA which is a French national research institution. The dataset consists of 1,000,000 image features extracted from both personal and Flickr photos, which of each is represented by a 128-D SIFT descriptor [12].

Note that graph sizes increase in the order of COIL-100, PubFig, NUS-WIDE, and INRIA. We used the top five nodes to construct k-NN graphs and set the parameter of Manifold Ranking $\alpha = 0.99$ [25, 26]. All experiments were conducted on a Linux quad 3.33 GHz Intel Xeon server with 32 GB of main memory. We implemented all approaches using GCC.

5.1 Efficiency of Mogul

We assessed the search time needed for each approach. Figure 1 shows the results. Since we can precompute matrix $\mathbf{L}$ and $\mathbf{D}$ without the query node as described in Section 4.2.2, we evaluated the efficiency of our search algorithm (Algorithm 2) in this experiment. The results of Mogul are referred to as "Mogul(k)" where k is the number of answer nodes. Since placing too many images on a screen confuses the user and degrades the user experience, image retrieval engines present at most 20 images at one time [21]. Therefore, we set the number of answer nodes k as 5, 10, 15, and 20. We set the number of anchor points, d, to 10 in EMR, and we exploited SVD as the low-rank approximation for FMR where the target rank was set to 250. In the iterative method, iteration was terminated when the residual dropped below 10^{-4} [21]. The inverse matrix approach cannot be configured for PubFig and NUS-WIDE datasets since it has quadratic memory consumption $O(n^2)$.

Figure 1 shows that our approach is much faster than the existing approaches. Specifically, Mogul is seven orders of magnitude faster than the inverse matrix approach. This is because the inverse matrix approach needs $O(n^3)$ time as described in Section 3 while the search cost of Mogul is $O(n)$ as shown in Theorem 2. Even though our approach approximately finds the top-k nodes unlike the inverse matrix approach, the accuracy of our approach is high as shown in the next section. Furthermore, the proposed approach is 50 times faster than EMR, the state-of-the-art approximation approach. The reasons are twofold. First, Mogul pruned

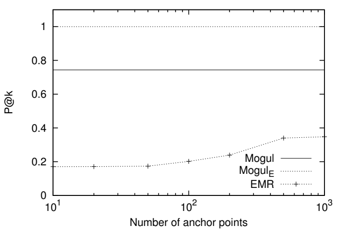


Figure 2: P@k vs. number of anchor points.

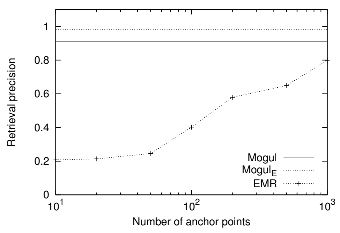


Figure 3: Retrieval precision vs. number of anchor points.

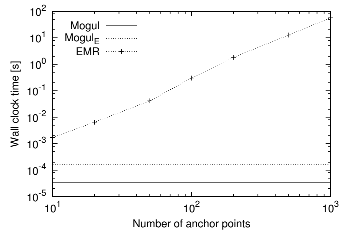


Figure 4: Efficiency vs. number of anchor points.

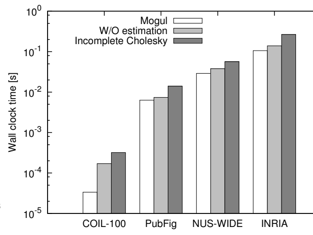


Figure 5: Effect of the pruning approach.

many unnecessary score computations due to the effectiveness of its estimation approach; the approximate scores of most nodes were not computed. In addition, as described in Section 2, EMR theoretically requires $O(nd + d^3)$ time. This computation cost is linear with respect to the number images n, but cubic to the number of anchor points d. This is the second reason. As a result, we can find top-k nodes more efficiently than the previous approaches.

5.2 Effectiveness of Each Approach

The following experiments examine the effectiveness of the two main techniques of Mogul: approximate score computation and pruning unnecessary computations.

5.2.1 Accuracy of the Search Results

Following the state-of-the-art approach, EMR, our approach approximately identifies the top-k nodes to reduce the search cost. We compared the accuracy of the two approaches. Note that Xu et al. reported that EMR can find the answer nodes more accurately than other approximation approaches such as FMR. We used two types of metrics to evaluate the accuracy: P@k and retrieval precision. P@k is the fraction of answer nodes among the top-k results that match those of the inverse matrix approach. P@k takes a value between 0 and 1, and, P@k is 1 if the detected nodes exactly match those of the inverse matrix approach. Retrieval precision is the ratio of answer nodes that correspond to the same objects as the query nodes; retrieval precision corresponds to semantic retrieval quality with regard to ground-truth. Since EMR utilizes the anchor points to compute the local approximation for each data point as described in Section 2, the number of anchor points, d, is expected to have an impact to the accuracy. Therefore, we changed the number of anchor points, d. We also evaluated the accuracy of the extended version of Mogul ("Mogul$_E$" hereafter) that utilizes the Modified Cholesky factorization introduced in Section 4.6. In addition, we evaluated the search time for various numbers of anchor points since EMR needs $O(nd + d^3)$ time. Figure 2, 3, and 4 show P@k, retrieval precision, and the search time, respectively; the COIL-100 dataset was accessed to find the top five nodes.

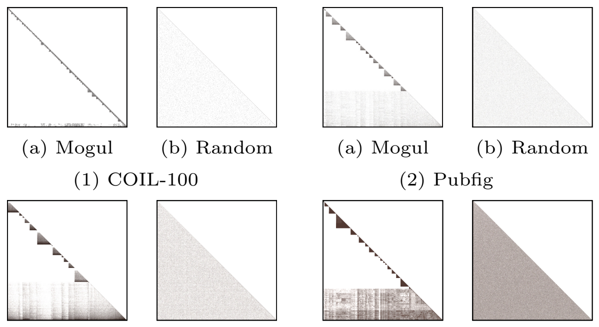
Figures 2 and 3 indicate that Mogul can more accurately find the top-k nodes than EMR. Furthermore, Mogul and Mogul$_E$ can find semantically the same images as the queries with over 90% accuracy as shown in Figure 3. As described in Section 2, since EMR utilizes the anchor nodes to approximately capture the intrinsic manifold structures present in the data points, EMR cannot well approximate the manifold structures if the number of anchor points is small. In addition, our approach exploits the optimization approach introduced in Section 4.2.2 to enhance the approximation quality. As a result, we can identify the answer nodes more accurately than the state-of-the-art approximation approach.

As shown in Figure 4, EMR is much slower than our approaches and takes even more computation time as the number of anchor points increases. This is because the time complexities of Mogul, Mogul$_E$, and EMR are $O(n)$, $O(m)$ and $O(nd + d^3)$, respectively. Note that m is the number of non-zero elements in matrix $\mathbf{L}$ of Mogul$_E$ as described in Section 4.6. This figure also indicates that Mogul$_E$ needs more computation time than Mogul. This is because the number of non-zero elements increases if we use Modified Cholesky factorization; the number of non-zero elements in matrix $\mathbf{L}$ of Mogul and Mogul$_E$ are $28,293$ and $132,818$, respectively, in this experiment. Since the elements in matrix $\mathbf{L}$ are forced to be zero in Incomplete Cholesky factorization as described in Section 4.2.2, Mogul yields a sparser data structure than Mogul$_E$. However, Mogul$_E$ has the advantage that it can find the answer nodes exactly at the cost of increased search time as shown in Figure 2 and 4.

Figures 2 and 4 also indicate that EMR forces a trade-off between speed and accuracy. That is, as the number of anchor points increases, the precision increases but the search speed decreases. This is because EMR approximately computes the ranking scores by using anchor points to reduce the search time. The proposed approach also uses approximate score computations to enhance the search speed, but it can more accurately find the top-k nodes than EMR. Moreover, our approach provides users the additional option of identifying the exact top-k nodes more efficiently than the state-of-the-art approximation approach.

5.2.2 Pruning Unnecessary Computations

As mentioned in Section 4.2.3 and 4.3, we use the sparse structure and the estimations to prune unnecessary score computations. To show the effectiveness of our approach, we evaluated the search time of an Incomplete Cholesky factorization based approach in which the approximate scores are computed by not using the sparse properties. This approach only applied Incomplete Cholesky factorization to obtain the approximate scores. It is clear that this approach needs $O(n)$ time from Lemma 1. In addition, we remove the pruning step from Mogul; this version computes the approximate scores by using the property of vector $\mathbf{y}$ (Lemma 4) that is derived from the sparse structure of matrix $\mathbf{L}$ (Lemma 3). Figure 5 shows the results to find the top five nodes. In this figure, "Incomplete Cholesky" represents the results of the Incomplete Cholesky factorization based approach, and Mogul without the estimation technique is abbreviated to "W/O estimation". Figure 6 plots the sparsity patterns of


(a) Mogul (b) Random (a) Mogul (b) Random

(1) COIL-100 (2) Pubfig

(a) Mogul (b) Random (a) Mogul (b) Random

(3) NUS-WIDE (4) INRIA

Figure 6: Non-zero elements in matrix L.

lower triangular matrix **L** where gray dots correspond to non-zero elements. In this figure, "Random" is the non-zero pattern when nodes are permuted in random order.

Figure 5 shows that Mogul without the estimation technique can cut the score computation time by up to 47% compared to the Incomplete Cholesky factorization based approach. Since image datasets have manifold structures, by using graph clustering approach, we can effectively obtain the sparse structure in matrix **L** as shown in Figure 6. By utilizing this sparse structure, we can efficiently compute the approximate scores as described in Section 4.2.3. Furthermore, the comparison of "Mogul" and "Incomplete Cholesky" in Figure 5 shows that we can additionally cut the search time by up to 90% from the Incomplete Cholesky factorization based approach by using the estimations. Since the estimation technique can effectively find the top-k nodes, we can efficiently find the answer nodes. This indicates that, in practice, the computation cost of Mogul is expected to be lower than the theoretical $O(n)$ time in which none of the node estimations are assumed to be pruned by (Theorem 2). A comparison of Figure 1 and 5 indicates that Mogul can find answer nodes more efficiently than the previous approaches even if it computes the scores of all nodes.

5.2.3 *Search Time for Out-of-sample Queries*

Section 4.6.2 shows our approach to handling query nodes outside the database; Mogul supports out-of-sample queries. In this section, we evaluate the search time of our approach for out-of-sample queries. Figure 7 show the search time of our approach and EMR for an out-of-sample query. Table 2 itemizes the search time of our approach.

As shown in Figure 7, our approach is up to 35 times faster than EMR for out-of-sample queries. As described in Section 2, EMR uses the anchor graph to approximately find top-k nodes. Given an out-of-sample query, EMR dynamically updates the anchor graph by adding the query node. Therefore, the search cost of EMR for the out-of-sample query is $O(nd + d^3)$. In contrast, our approach is static; we do not change matrix **L**. Instead, we effectively compute the scores in the query vector **q** by computing the neighbor nodes for the query node. Furthermore, we can efficiently obtain the neighbor nodes by using the nearest cluster for the query node as shown in Table 2. The search cost of our approach is $O(n)$ for the out-of-sample query as described in Section 4.6.2. As a result, Mogul is more efficient than EMR in handling the out-of-sample query.

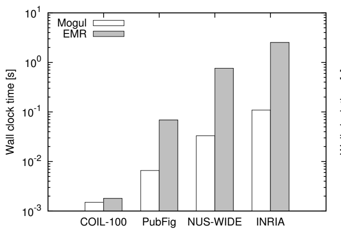


Figure 7: Search time for out-of-sample.

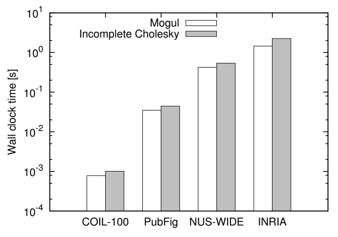


Figure 8: Precomputation time of Mogul.

Table 2: Breakdown of out-of-sample search.

Dataset	Search time [ms]		
	Nearest neighbor	Top-k search	Overall
COIL-100	1.47	0.03	1.50
Pubfig	0.29	6.30	6.59
NUS-WIDE	4.01	29.00	33.01
INRIA	2.81	106.20	109.01

5.2.4 *Precomputation Time*

Our optimization technique reduces the approximation error by permuting the nodes. As described in Section 4.2.3, the optimization approach has the additional advantage that we can efficiently precompute matrix **L** by employing the particular non-zero pattern in the matrix. In this experiment, we evaluated the effectiveness of the optimization technique in terms of precomputation time. In Figure 8, "Incomplete Cholesky" represents the results where the non-zero pattern is not used in the precomputation by randomly permuting the nodes.

Figure 8 indicates that the precomputing time of our approach is linear with respect to the number of nodes in the graph. These results confirm our theoretical analysis on the precomputing time (Lemma 2). In addition, our technique can cut the precomputation time by up to 20% by employing the non-zero pattern. Equations (6) and (7) imply that an element is expected to be used more times if it lies in the left-side of the matrix. As shown in Section 4.2.2, the optimization technique is designed to obtain the matrix such that left-side elements are sparse. However, without the optimization approach, the left-side of the matrix is not expected to be sparse. As a result, the optimization approach can reduce the precomputation time.

5.3 Case Studies

Since Manifold Ranking determines the rankings of data points with respect to the intrinsic manifold structures, it can effectively find images similar to the query image as described in Section 1. In Section 5.2.1, by using P@k and retrieval precision, we quantitatively showed that Mogul has higher accuracy than EMR. This section evaluates Mogul in a qualitative manner; it shows the results of case studies demonstrating that Mogul can find more similar images than the existing approaches. We applied each approach to the COIL-100 dataset which contains images of 100 objects from 72 different angles. We set the number of anchor points to 100 for EMR. Figure 9 depicts the results of the case studies where "Connected" represents the direct neighbors of the query nodes in the k-NN graph. That is, "Connected" corresponds to the results of k-nearest neighbor search since the k-NN graph is used in Manifold Ranking [3].

This figure indicates that, although the semantically different images can be connected in the given graph, our ap-

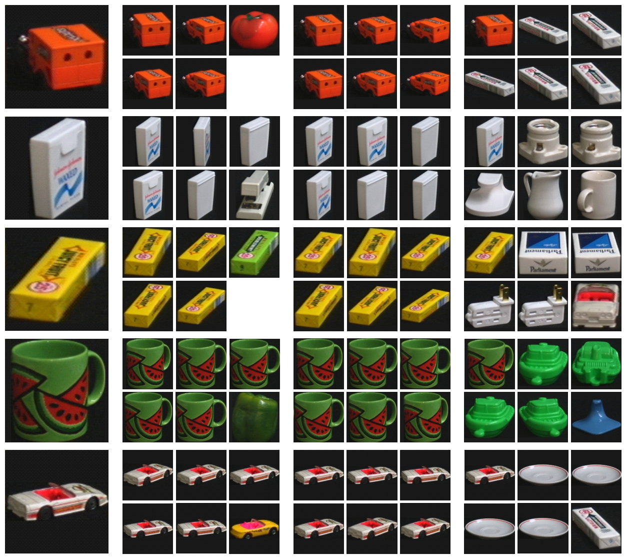


Figure 9: Retrieval on COIL-100 dataset.

proach can effectively find images similar to the query node. For example, in the first case of Figure 9, the color of the query image is orange and the shape is square. However, the query image is connected the semantically different image of tomato. This indicates that the semantically different images can be contained in the results of k-nearest neighbor search. On the other hand, as shown in the result of our approach for the first query, Mogul can effectively find semantically similar images; the color and shape of the answer images are the same as those of the query node. Furthermore, the results of our approach in the first case reveal that the query is not an image of orange square but orange cargo truck; Mogul can be used to more effectively understand the semantics of the query image. In addition, Figure 9 shows that EMR has difficulty in finding semantically similar images. For example, in the first case of Figure 9, the results of EMR are all the same shape (square) but different objects. In the other cases shown in Figure 9, the results of EMR are different in terms of color and/or shape.

Since the existing approaches identified images semantically different from the query images, they suffer from the semantic gap problem in image retrieval. Even though the existing approaches can enhance the search speed for Manifold Ranking, they lose the benefit of applying Manifold Ranking for image retrieval. On the other hand, the results of our approach are reasonable, and consistent with our intuition. While our approach achieves high accuracy, it is significantly more efficient than the previous approaches. This indicates that our approach is another option for the research community in utilizing Manifold Ranking.

6. CONCLUSIONS

We have solved the key problems of Manifold Ranking that prevented it from being used efficiently to find the top-k nodes. The proposed approach, Mogul, is based on two advances: (1) It efficiently computes the approximate scores by utilizing Incomplete Cholesky factorization, and (2) It prunes unnecessary approximate computations in finding the top-k nodes by estimating the upper bounding scores. The search cost of our approach is $O(n)$ which is significantly smaller than the $O(n^3)$ of the inverse matrix approach. Experiments confirmed that Mogul is much faster than the previous approaches on real-world datasets.

7. REFERENCES

[1] J. Bu, S. Tan, C. Chen, C. Wang, H. Wu, L. Zhang, and X. He. Music Recommendation by Unified Hypergraph: Combining Social Media Information and Music Content. In *ACM Multimedia*, pages 391–400, 2010.

[2] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image Retrieval: Ideas, Influences, and Trends of the New Age. *ACM Comput. Surv.*, 40(2), 2008.

[3] Y. Fujiwara and G. Irie. Efficient Label Propagation. In *ICML*, pages 784–792, 2014.

[4] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka. Efficient Ad-hoc Search for Personalized PageRank. In *SIGMOD*, pages 445–456, 2013.

[5] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka. Fast and Exact Top-k Algorithm for PageRank. In *AAAI*, 2013.

[6] J. He, M. Li, H. Zhang, H. Tong, and C. Zhang. Manifold-ranking Based Image Retrieval. In *ACM Multimedia*, pages 9–16, 2004.

[7] J. He, M. Li, H. Zhang, H. Tong, and C. Zhang. Generalized Manifold-ranking-based Image Retrieval. *IEEE Transactions on Image Processing*, 15(10):3170–3177, 2006.

[8] R. He, Y. Zhu, and W. Zhan. Fast Manifold-ranking for Content-based Image Retrieval. In *CCCM*, pages 299–302, 2009.

[9] H. Jégou, M. Douze, and C. Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.

[10] K.-H. Kim and S. Choi. Walking on Minimax Paths for k-NN Search. In *AAAI*, 2013.

[11] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and Simile Classifiers for Face Verification. In *ICCV*, pages 365–372, 2009.

[12] D. G. Lowe. Distinctive Image Features from Scale-invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[13] M. Nakatsuji, Y. Fujiwara, A. Tanaka, T. Uchiyama, and T. Ishida. Recommendations over Domain Specific User Graphs. In *ECAI*, pages 607–612, 2010.

[14] S. A. Nene, S. K. Nayar, and H. Murase. Columbia Object Image Library (COIL-100). Technical report, Feb 1996.

[15] J. Nocedal and S. Wrigh. *Numerical Optimization.* Springer, 2006.

[16] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition.* Cambridge University Press, 2007.

[17] H. Shiokawa, Y. Fujiwara, and M. Onizuka. Fast Algorithm for Modularity-based Graph Clustering. In *AAAI*, 2013.

[18] J. Song, Y. Yang, Y. Yang, Z. Huang, and H. T. Shen. Inter-media Hashing for Large-scale Retrieval from Heterogeneous Data Sources. In *SIGMOD Conference*, pages 785–796, 2013.

[19] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and Efficiency in High Dimensional Nearest Neighbor Search. In *SIGMOD Conference*, pages 563–576, 2009.

[20] J. Weston, R. Kuang, C. S. Leslie, and W. S. Noble. Protein Ranking by Semi-supervised Network Propagation. *BMC Bioinformatics*, 7(S-1), 2006.

[21] B. Xu, J. Bu, C. Chen, D. Cai, X. He, W. Liu, and J. Luo. Efficient Manifold Ranking for Image Retrieval. In *SIGIR*, pages 525–534, 2011.

[22] M. Yannakakis. Computing the Minimum Fill-in is NP-complete. *SIAM. J. on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

[23] X. Yuan, X.-S. Hua, M. Wang, and X. Wu. Manifold-ranking Based Video Concept Detection on Large Database and Feature Pool. In *ACM Multimedia*, pages 623–626, 2006.

[24] D. Zhang, D. Agrawal, G. Chen, and A. K. H. Tung. Hashfile: An Efficient Index Structure for Multimedia Data. In *ICDE*, pages 1103–1114, 2011.

[25] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with Local and Global Consistency. In *NIPS*, 2003.

[26] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on Data Manifolds. In *NIPS*, 2003.