Fast Algorithm for Anchor Graph Hashing

Yasuhiro Fujiwara
NTT Communication Science Laboratories
yasuhiro.fujiwara.kh@hco.ntt.co.jp

Sekitoshi Kanai
NTT Software Innovation Center
Keio University
sekitoshi.kanai.fu@hco.ntt.co.jp

Yasutoshi Ida
NTT Software Innovation Center
yasutoshi.ida.yc@hco.ntt.co.jp

Atsutoshi Kumagai
NTT Software Innovation Center
atsutoshi.kumagai.ht@hco.ntt.co.jp

Naonori Ueda
NTT Communication Science Laboratories
naonori.ueda.fr@hco.ntt.co.jp

ABSTRACT

Anchor graph hashing is used in many applications such as cancer detection, web page classification, and drug discovery. It computes the hash codes from the eigenvectors of the matrix representing the similarities between data points and anchor points; anchors refer to the points representing the data distribution. In performing an approximate nearest neighbor search, the hash codes of a query data point are determined by identifying its closest anchor points. Anchor graph hashing, however, incurs high computation cost since (1) the computation cost of obtaining the eigenvectors is quadratic to the number of anchor points, and (2) the similarities of the query data point to all the anchor points must be computed. Our proposal, *Tridiagonal hashing*, increases the efficiency of anchor graph hashing because of its two advances: (1) we apply a graph clustering algorithm to compute the eigenvectors from the tridiagonal matrix obtained from the similarities between data points and anchor points, and (2) we detect anchor points closest to the query data point by using a dimensionality reduction approach. Experiments show that our approach is several orders of magnitude faster than the previous approaches. Besides, it yields high search accuracy than the original anchor graph hashing approach.

PVLDB Reference Format:
Yasuhiro Fujiwara, Sekitoshi Kanai, Yasutoshi Ida, Atsutoshi Kumagai, and Naonori Ueda. Fast Algorithm for Anchor Graph Hashing. PVLDB, 14(6): 916 - 928, 2021.
doi:10.14778/3447689.3447696

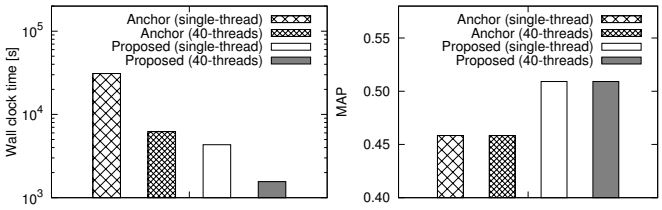
1 INTRODUCTION

Massive sets of high-dimensional data are now being stored day after day with the rapid development of database systems. This imposes a fundamental challenge to efficiently and accurately process millions of records of different data types such as images, audio, and video [8, 10, 11, 23, 24]. Nearest neighbor search on high-dimensional data is a fundamental research topic [9, 42]. One conventional approach to address the problem uses tree-based schemes for nearest neighbor search, such as R-tree [15], K-D tree [2], and SR-tree [26], which were proposed in the 1980s and 1990s. They need $O(\log n)$ query time where n is the number of data points. Although these approaches yield accurate results, they are not time-efficient for high-dimensional data due to the curse of dimensionality [18]. Specifically, when the dimensionality exceeds about ten, they are slower than the brute-force, linear-scan approach [51]. In these approaches, most of the query cost is spent on verifying a data point as a real nearest neighbor [4]. Since the linear-scan approach's cost increases linearly with dataset size, the query cost should be constant or sub-linear [47]. Approximate nearest neighbor search improves efficiency by relaxing the precision of verification, and several approaches were proposed such as Clindex [28], MEDRANK [6], and SASH [20] in the database community in the 2000s. However, these approaches do not offer sub-linear growth of query cost in the worst case.

Hashing techniques transform data records into short fixed-length codes to reduce storage costs and offer approximate nearest neighbor search with constant or sub-linear time complexity. The approximate nearest neighbor search consists of two phases: offline and online. The offline phase computes hash codes from the data. The online phase finds the nearest neighbors to the query data point by hashing (compressing) the query. Locality Sensitive Hashing (LSH) is one of the most popular hashing techniques [13]. It uses random projection to map data points into hash codes, and it is more efficient than a tree-based scheme in searching for nearest neighbors as it needs fewer disk accesses [13]. In the 2010s, several database researchers proposed LSH variants such as C2LSH [12], SRS [46], and QALSH [22]. However, the LSH-based approaches yield inadequate search accuracy since LSH employs random projections [30]. As a result, they require long hash codes (say, 1024 bits) to achieve satisfactory search accuracy [14]. This is because the collision probability of two codes having the same hash codes falls as code length increases [37]. However, long codes increase search and storage costs [29, 35].

Anchor graph hashing was proposed to improve the search efficiency and accuracy [36]; anchors refer to the points representing the data distribution in the high-dimensional space. Technically, it is based on spectral hashing that computes eigendecomposition for the similarity matrix among data points to obtain hash codes. Although spectral hashing achieves higher search accuracy than LSH, it is difficult to apply to large-scale data since it requires $O(n^2d)$ time and $O(n^2)$ space to obtain all pairwise similarities

Proceedings of the VLDB Endowment, Vol. 14, No. 6 ISSN 2150-8097.
doi:10.14778/3447689.3447696



(1-1) Offline phase time (1-2) Search accuracy

Figure 1: Offline phase time and MAP for wesad. When a single thread is used, our approach ("Proposed (single-thread)") is much faster than the original approach of anchor graph hashing ("Anchor (single-thread)"), and our approach's efficiency improves with the use of multiple threads ("Proposed (40-threads)"). Moreover, our approach yields higher search accuracy than the original approach. We will describe the parameter settings detailed in Section 5.

where d is the number of dimensions. To overcome this drawback, anchor graph hashing computes similarities between data points by exploiting the low-rank property of the approximate adjacency matrix computed from the similarity between data points and anchor points. As shown in previous studies [31, 34, 36], LSH offers lower search accuracy than other hashing techniques, and anchor graph hashing outperforms recent hashing techniques such as binary reconstruction embeddings [27], discrete graph hashing [34], and binary autoencoder [3] in terms of efficiency. Due to its effectiveness, anchor graph hashing can be used in many applications such as cancer detection [32], web page classification [54], and drug discovery [53]. However, its computation time is high for large-scale data. In the offline phase, to obtain hash codes, it computes eigenvectors of the matrix obtained from the similarities between data points and anchor points. However, it needs $O(nm^2)$ time to obtain the eigenvectors where m is the number of anchor points; the computation cost is quadratic to the number of anchor points. The online phase finds the closest anchor points to a query data point from all anchor points to perform the nearest neighbor search. Therefore, anchor graph hashing suffers high computation cost.

This paper proposes *Tridiagonal hashing*, a novel and efficient approach to anchor graph hashing. Its advantage is that it can improve the efficiency and search accuracy of anchor graph hashing. The offline phase of our approach efficiently computes the eigenvectors by applying a graph clustering algorithm to the tridiagonal matrix obtained from a similarity matrix between data points and anchor points. Moreover, the online phase computes approximate distances using a dimensionality reduction approach to identify anchor points closest to the query data point efficiently. Note that our approach is easily parallelized to improve efficiency. Besides, we can improve the search accuracy because our approach recursively reduces the number of bits in hash codes in finding similar data points. Our approach is theoretically guaranteed to find all data points found by the original anchor graph hashing; it is guaranteed to have no false negatives in the search results.

Figure 1 shows the offline phase time for the wesad dataset, a dataset of stress-affect lab study, including over sixty million data points. To assess search accuracy, the figure also shows Mean Average Precision (MAP) [1] for finding data points of the same label as the query data point. In this figure, "Proposed (single-thread)" and "Anchor (single-thread)" indicate our approach and the original anchor graph hashing using a single thread, respectively. "Proposed (40-thread)" corresponds to our approach using 40 threads, and "Anchor (40-thread)" is the original approach parallelized by the proposed approach. As shown in Figure 1-1, our approach is more efficient than the original approach, and it can increase efficiency by using multiple threads. As a result, our approach can compute hash codes within a half-hour. On the other hand, the original approach with a single thread takes over eight hours, and it takes nearly two hours even if it uses multiple threads. As shown in Figure 1-2, our approach yields higher search accuracy than the original approach since it guarantees no false negatives in the search results. These results indicate that our approach enhances the efficiency of anchor graph hashing while achieving high search accuracy. In summary, the main contributions of this paper are as follows:

- We propose an efficient approach to anchor graph hashing that computes eigenvectors using a graph clustering algorithm and prunes similarity computations in finding the closest anchor points by using approximate distances.
- The proposed approach yields higher search accuracy than the original approach of anchor graph hashing. This is because the proposed approach is theoretically designed to yield the search results of no false negatives.
- Experiments confirm that our approach achieves higher search accuracy while reducing the time to compute the hash codes compared to the previous approaches; it is up to 3195.7 times faster than the existing approaches.

In the remainder of this paper, we give preliminaries in Section 2, introduce our approach in Section 3, describe related work in Section 4, show experimental results in Section 5, and provide our conclusions in Section 6.

Table 1: Definitions of main symbols.

Symbol	Definition
n	Number of data points
d	Number of dimensions
m	Number of anchor points
r	Number of bits in hash codes
s	Number of closest anchor points
$\mathbb{X}$	Set of data points
$\mathbb{U}$	Set of anchor points
σ_i	i-th eigenvalue to compute hash codes
$\mathbf{x}_i$	i-th data point vector of length d
$\mathbf{u}_i$	i-th anchor point vector of length d
$\mathbf{v}_i$	i-th eigenvector to compute hash codes
$\mathbf{Y}$	Embedding matrix of size $n \times r$
$\mathbf{Z}$	Similarity matrix of size $n \times m$
$\mathbf{A}$	Approximate adjacency matrix of size $n \times n$
$\mathbf{M}$	Symmetric matrix of size $m \times m$
$\mathbf{T}$	Tridiagonal matrix of size $m \times m$

2 PRELIMINARIES

We introduce the background to this paper. Table 1 lists the main symbols. Let n and d be the number of data points and dimensions, respectively. Let row vector $\mathbf{x}_i = [x_i[1], \ldots, x_i[d]]$ correspond to

the i-th data point; a set of data points is given as $\mathbb{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. If m ($m < n$) is the number of anchor points, $\mathbb{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ is a set of anchor points. Anchor points are randomly sampled from the data points or centers of k-means clusters for the data points [33]. If r is the number of bits in hash codes and $\mathbf{Y} \in \{1, -1\}^{n \times r}$ is an embedding of n data points, anchor graph hashing computes the embedding by minimizing

$$\begin{aligned} &\min_{\mathbf{Y}} \tfrac{1}{2} \textstyle\sum_{i,j=1}^{n} \|\mathbf{y}_i - \mathbf{y}_j\|^2 A[i][j] = \operatorname{tr}(\mathbf{Y}^\top(\mathbf{D} - \mathbf{A})\mathbf{Y}) \\ &\text{s.t. } \mathbf{Y} \in \{1, -1\}^{n \times r}, \mathbf{1}^\top \mathbf{Y} = \mathbf{0}, \mathbf{Y}^\top \mathbf{Y} = n\mathbf{I} \end{aligned} \tag{1}$$

In this equation, $\mathbf{y}_i$ is the i-th row of $\mathbf{Y}$ representing the r-bits code for data point $\mathbf{x}_i$, $\|\cdot\|$ is the L_2-norm of a vector, $\mathbf{A}$ is the $n \times n$ adjacency matrix where $A[i][j]$ corresponds to the similarity between data point $\mathbf{x}_i$ and $\mathbf{x}_j$, $\mathbf{D}$ is a diagonal matrix given as $\mathbf{D} = \operatorname{diag}(\mathbf{A1})$ with $\mathbf{1} = [1, \ldots, 1]^\top \in \mathcal{R}^n$, and $\mathbf{I}$ is the identity matrix. The constraint $\mathbf{1}^\top \mathbf{Y} = \mathbf{0}$ is imposed to maximize each bit's information, and $\mathbf{Y}^\top \mathbf{Y} = n\mathbf{I}$ forces r bits to be mutually uncorrelated to minimize redundancy among bits. Intuitively, anchor graph hashing embeds the data points in a low-dimensional Hamming space such that the neighbors in the original space remain neighbors.

Equation (1) is an integer problem, and solving the problem is equivalent to balanced graph partitioning even for a single bit; this is known to be NP-hard [52]. One approach to relaxing it uses the spectral method to drop the integer constraint of $\mathbf{Y} \in \{1, -1\}^{n \times r}$ and allow $\mathbf{Y} \in \mathcal{R}^{n \times r}$. The relaxed problem's solution consists of the r eigenvectors of $\mathbf{A}$ with minimal eigenvalues except for the eigenvalue of zero. Since these eigenvectors are orthogonal to each other, they satisfy the orthogonal constraint after multiplication by the scale $\sqrt{n}$, that is, $\mathbf{Y}^\top \mathbf{Y} = n\mathbf{I}$ holds. Besides, they satisfy the constraint of $\mathbf{1}^\top \mathbf{Y} = \mathbf{0}$. This is because the excluded eigenvector with the eigenvalue of zero is $\mathbf{1}$ and all the other eigenvectors are orthogonal to it. To obtain the binary codes, the binarization operation with threshold zero is performed over the solution $\mathbf{Y}$, which forms a Hamming embedding.

Anchor graph hashing approximately computes adjacency matrix $\mathbf{A}$ by using anchor points. If $\mathbf{Z}$ is an $n \times m$ similarity matrix between data points and anchor points, it computes matrix $\mathbf{Z}$ as

$$Z[i][j] = \begin{cases} \frac{\exp(-D^2(\mathbf{x}_i, \mathbf{u}_j)/t)}{\sum_{k \in \langle i \rangle} \exp(-D^2(\mathbf{x}_i, \mathbf{u}_k)/t)}, & \forall j \in \langle i \rangle \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

In this equation, $\langle i \rangle$ are the indices of the s ($s \ll m$) closest anchor points of $\mathbf{x}_i$ in $\mathbb{U}$ according to l_2 distance function $D(\cdot)$; t ($t > 0$) is the bandwidth parameter. Using $\mathbf{Z}$, approximate adjacency matrix $\mathbf{A}$ can be computed as $\mathbf{A} = \mathbf{Z}\mathbf{\Lambda}^{-1}\mathbf{Z}^\top$ where $\mathbf{\Lambda} = \operatorname{diag}(\mathbf{Z}^\top \mathbf{1})$ [33]. Each row of $\mathbf{Z}$ contains s nonzero elements, which sum to 1 from Equation (2). By following the original paper [36], we assume that all data points are connected in the graph given by $\mathbf{A}$.

The offline phase computes the largest eigenvalues σ_i ($i \geq 0$) of the following $m \times m$ symmetric matrix $\mathbf{M}$ to obtain hash codes:

$$\mathbf{M} = \mathbf{\Lambda}^{-\frac{1}{2}} \mathbf{Z}^\top \mathbf{Z} \mathbf{\Lambda}^{-\frac{1}{2}} \tag{3}$$

This is because the minimal eigenvalues of $\mathbf{A}$ correspond to the maximal eigenvalues of $\mathbf{M}$. If all the data points are connected, we have $\sigma_0 = 1$; the largest eigenvalue of $\mathbf{M}$ is 1. However, $\sigma_0 = 1$ yields the eigenvector of $\mathbf{1}$, and such the eigenvector is not useful in computing hash codes. Therefore, anchor graph hashing computes r largest eigenvector-eigenvalue pairs $\{(\mathbf{v}_i, \sigma_i)\}_{i=1}^{r}$ such that $1 > \sigma_1 \geq \ldots \geq \sigma_r > 0$ by ignoring the largest eigenvalue $\sigma_0 = 1$ [36]. If $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_r] \in \mathcal{R}^{m \times r}$, $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_r) \in \mathcal{R}^{r \times r}$, and $\mathbf{W} = \sqrt{n}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}\mathbf{\Sigma}^{-\frac{1}{2}}$, it obtains embedding matrix $\mathbf{Y}$ as follows [36]:

$$\mathbf{Y} = \operatorname{sgn}(\mathbf{ZW}) \tag{4}$$

where $\operatorname{sgn}(\cdot)$ is the sign function.

In the online phase, for query data point $\mathbf{x}$, it computes column vector $\mathbf{z}(\mathbf{x})$ of length m as follows:

$$\mathbf{z}(\mathbf{x}) = \frac{\left[\delta_1 \exp(-D^2(\mathbf{x}, \mathbf{u}_1)/t), \ldots, \delta_m \exp(-D^2(\mathbf{x}, \mathbf{u}_m)/t)\right]^\top}{\sum_{i=1}^{m} \delta_i \exp(-D^2(\mathbf{x}, \mathbf{u}_i)/t))} \tag{5}$$

where $\delta_i \in \{1, 0\}$ and $\delta_i = 1$ if and only if $\mathbf{u}_i$ is one of the s closest anchor points of $\mathbf{x}$ in $\mathbb{U}$; $\delta_i = 0$ otherwise. Let $\mathbf{y}$ be a row vector of length r that corresponds to r-bits codes for data point $\mathbf{x}$, it is computed as follows [36]:

$$\mathbf{y} = \operatorname{sgn}(\mathbf{z}(\mathbf{x})\mathbf{W}) \tag{6}$$

Even though anchor graph hashing can effectively compute hash codes, it incurs high computation cost. The offline phase needs $O(nmd + nm^2 + nsr)$ time to compute hash codes for the given anchor points described in [36]; it is quadratic to the number of anchor points. This is because it needs $O(nmd)$ time to compute $\mathbf{Z}$, $O(nm^2)$ time to compute $\mathbf{M}$, $O(m^3)$ time to compute its eigenvectors, $O(nsr)$ time to compute $\mathbf{ZW}$, and $O(nr)$ time to compute $\mathbf{Y}$. As a result, in terms of computation cost, the offline phase has a bottleneck in computing eigenvectors from $\mathbf{M}$. In the online phase, anchor graph hashing takes $O(md + sr)$ to compute hash codes for data point $\mathbf{x}$. This is because it takes $O(md)$ time to identify the closest anchor points for obtaining $\mathbf{z}(\mathbf{x})$ and $O(sr)$ time to compute $\mathbf{y}$. As described in [36], computation cost in the online phase is dominated by the construction of $\mathbf{z}(\mathbf{x})$. Consequently, the computation cost of anchor graph hashing is high when handling large-scale data.

3 PROPOSED METHOD

This section explains the proposed approach. Section 3.1 shows how our approach computes eigenvectors of the tridiagonal matrix obtained from similarities between data points and anchor points in the offline phase. Section 3.2 details how our approach finds the closest anchor points using a dimensionality reduction approach. Section 3.3 details our algorithms and their properties. Section 3.4 proposes implementation-oriented approaches to improve our approach's efficiency and search accuracy. We show proofs of lemmas and theorems in this section in the Appendix.

3.1 Offline Phase Computation

In this section, we show our approach to computing r eigenvector-eigenvalue pairs to obtain hash codes in the offline phase. As mentioned in Section 2, anchor graph hashing needs high computation cost to obtain eigenvectors of the largest eigenvalues. Although the power method is a well-known approach to computing eigenvalues, we do not use the approach since it is computationally slow; it needs $O(m^3)$ time to solve the eigenproblem [41]. Instead, we exploit the bisection method since it can efficiently compute just a subset of approximate eigenvalues; other approaches such as the QR algorithm and the divide-and-conquer algorithm compute all the eigenvalues [7]. We compute the tridiagonal matrix directly from

$\mathbf{Z}$ used in the bisection method; we do not compute $\mathbf{M}$ to obtain eigenvalues. From approximate eigenvalues, we accurately compute eigenvalues and eigenvectors by using inverse iteration [44]. Although our approach does not compute $\mathbf{M}$, we can effectively perform approximate nearest neighbor searches. This is because the tridiagonal matrix has the same eigenvalues as $\mathbf{M}$ [7].

In Section 3.1.1, we describe the approach to compute the tridiagonal matrix. Section 3.1.2 shows our approach to compute approximate eigenvalues by the bisection method for the tridiagonal matrix. Section 3.1.3 describes the approach to compute eigenvectors from the obtained approximate eigenvalues by inverse iteration.

3.1.1 Tridiagonal Matrix Computation. This section describes the approach to compute the tridiagonal matrix, which has the same eigenvalues as $\mathbf{M}$. In our approach, we use the Lanczos method [7] since it is an effective way of computing a tridiagonal matrix from a symmetric matrix. If $\mathbf{P}$ is an $m \times m$ orthonormal matrix, the Lanczos method decomposes $\mathbf{M}$ as follows:

$$\mathbf{M} = \mathbf{P}\mathbf{T}\mathbf{P}^\top \tag{7}$$

where $\mathbf{T}$ is an $m \times m$ symmetric tridiagonal matrix given by

$$\mathbf{T} = \begin{bmatrix} \alpha_1 & \beta_1 & & & 0 \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \alpha_{m-1} & \beta_{m-1} \\ 0 & & & \beta_{m-1} & \alpha_m \end{bmatrix} \tag{8}$$

If $\mathbf{p}_i$ is the i-column vector of $\mathbf{P}$, we have

$$\alpha_i = \mathbf{p}_i^\top \mathbf{M}\mathbf{p}_i \tag{9}$$

$$\beta_i = \|\mathbf{M}\mathbf{p}_i - \beta_{i-1}\mathbf{p}_{i-1} - \alpha_i\mathbf{p}_i\| \tag{10}$$

$$\mathbf{p}_{i+1} = (\mathbf{M}\mathbf{p}_i - \beta_{i-1}\mathbf{p}_{i-1} - \alpha_i\mathbf{p}_i)/\beta_i \tag{11}$$

where $\mathbf{p}_0 = \mathbf{0}$ and $\mathbf{p}_1$ is a random vector such that $\|\mathbf{p}_1\| = 1$. We have $\beta_i \geq 0$ from Equation (10). Since $\mathbf{P}$ is an orthonormal matrix, $\mathbf{T}$ and $\mathbf{M}$ have the same eigenvalue σ_i [7]. Besides, the eigenvector of $\mathbf{M}$ is obtained as $\mathbf{v}_i = \mathbf{P}\mathbf{v}'_i$ from Equation (7) if $\mathbf{v}'_i$ is the eigenvector of $\mathbf{T}$ corresponding to σ_i. Although we can obtain eigenvector-eigenvalue pair $(\mathbf{v}_i, \sigma_i)$ of $\mathbf{M}$ from $\mathbf{T}$, it takes $O(m^3)$ time to obtain $\mathbf{T}$, whose size is $m \times m$, from Equation (9), (10), and (11).

We introduce column vector $\mathbf{b}_i = \mathbf{Z}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{p}_i$ of length n to efficiently compute $\mathbf{T}$. It needs $O(ns)$ time to compute $\mathbf{b}_i$ since the number of nonzero elements in $\mathbf{Z}$ is ns and $\mathbf{b}_i$ can be obtained through matrix-vector multiplications. Note that $\mathbf{\Lambda}$ is a diagonal matrix of size $m \times m$. From Equation (3), we can rewrite α_i as

$$\alpha_i = \mathbf{p}_i^\top \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{Z}^\top\mathbf{Z}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{p}_i = \mathbf{b}_i^\top\mathbf{b}_i = \|\mathbf{b}_i\|^2 \tag{12}$$

As for β_i, we have the following equation:

$$\begin{aligned}(\beta_i)^2 =& (\mathbf{M}\mathbf{p}_i - \beta_{i-1}\mathbf{p}_{i-1} - \alpha_i\mathbf{p}_i)^\top(\mathbf{M}\mathbf{p}_i - \beta_{i-1}\mathbf{p}_{i-1} - \alpha_i\mathbf{p}_i) \\ =& \mathbf{p}_i^\top\mathbf{M}^2\mathbf{p}_i - \beta_{i-1}\mathbf{p}_i^\top\mathbf{M}^\top\mathbf{p}_{i-1} - \alpha_i\mathbf{p}_i^\top\mathbf{M}^\top\mathbf{p}_i \\ & - \beta_{i-1}\mathbf{p}_{i-1}^\top\mathbf{M}\mathbf{p}_i + \beta_{i-1}^2\mathbf{p}_{i-1}^\top\mathbf{p}_{i-1} + \alpha_i\beta_{i-1}\mathbf{p}_{i-1}^\top\mathbf{p}_i \\ & - \alpha_i\mathbf{p}_i^\top\mathbf{M}\mathbf{p}_i + \alpha_i\beta_{i-1}\mathbf{p}_i^\top\mathbf{p}_{i-1} + \alpha_i^2\mathbf{p}_i^\top\mathbf{p}_i\end{aligned} \tag{13}$$

Since $\mathbf{P}$ is an orthonormal matrix, we have $\mathbf{p}_i^\top\mathbf{p}_i = 1$ and $\mathbf{p}_i^\top\mathbf{p}_j = 0$ if $i \neq j$. Therefore,

$$\begin{aligned}(\beta_i)^2 =& \mathbf{p}_i^\top\mathbf{M}^2\mathbf{p}_i - \beta_{i-1}\mathbf{p}_i^\top\mathbf{M}^\top\mathbf{p}_{i-1} - \beta_{i-1}\mathbf{p}_{i-1}^\top\mathbf{M}\mathbf{p}_i - \alpha_i^2 + \beta_{i-1}^2 \\ =& \mathbf{b}_i^\top\mathbf{Z}\mathbf{\Lambda}^{-1}\mathbf{Z}^\top\mathbf{b}_i - \beta_{i-1}\mathbf{b}_i^\top\mathbf{b}_{i-1} - \beta_{i-1}\mathbf{b}_{i-1}^\top\mathbf{b}_i - \alpha_i^2 + \beta_{i-1}^2 \\ =& \|\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{Z}^\top\mathbf{b}_i\|^2 - 2\beta_{i-1}\langle\mathbf{b}_i, \mathbf{b}_{i-1}\rangle - \alpha_i^2 + \beta_{i-1}^2\end{aligned} \tag{14}$$

where $\langle\cdot\rangle$ represents the inner product of two vectors. Therefore, β_i can be recast as follows:

$$\beta_i = \sqrt{\|\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{Z}^\top\mathbf{b}_i\|^2 - 2\beta_{i-1}\langle\mathbf{b}_i, \mathbf{b}_{i-1}\rangle - \alpha_i^2 + \beta_{i-1}^2} \tag{15}$$

$\mathbf{p}_{i+1}$ is also rewritten as follows:

$$\mathbf{p}_{i+1} = (\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{Z}^\top\mathbf{b}_i - \beta_{i-1}\mathbf{p}_{i-1} - \alpha_i\mathbf{p}_i)/\beta_i \tag{16}$$

Equations (12), (15), and (16) indicate that we can compute α_i, β_i, and $\mathbf{p}_i$ from $\mathbf{b}_i$. We have the following lemma for computation cost:

Lemma 1 (Tridiagonal Matrix Computation). *It takes $O(nms)$ time to compute $\mathbf{T}$ and $\mathbf{P}$ from Equation (12), (15), and (16).*

Note that the original approach of the Lanczos method needs $O(nm^2)$ time to compute $\mathbf{M}$ [36] and $O(m^3)$ time to compute $\mathbf{T}$ [41]. Therefore, this lemma indicates that we can efficiently compute $\mathbf{T}$ that has the same eigenvalues as $\mathbf{M}$ from $\mathbf{Z}$ without computing $\mathbf{M}$.

3.1.2 Approximate Eigenvalues Computation. This section shows how to compute approximate eigenvalues by using the bisection method. The bisection method can find the i-th approximate eigenvalue $\tilde{\sigma}_i$ corresponding to σ_i in the interval $[\underline{\sigma}_i, \overline{\sigma}_i]$; $\underline{\sigma}_i$ and $\overline{\sigma}_i$ are, respectively, left and right bounds of $\tilde{\sigma}_i$ such that $\underline{\sigma}_i \leq \tilde{\sigma}_i \leq \overline{\sigma}_i$ [7]. The bisection method computes an approximate eigenvalue by using $a(\lambda)$; $a(\lambda)$ is the number of agreements in sign between consecutive numbers of the sequence $(f_0(\lambda), f_1(\lambda), \ldots, f_m(\lambda))$ where $f_i(\lambda)$ is given as follows [7]:

$$f_i(\lambda) = \begin{cases} 1 & \text{if } i = 0 \\ \alpha_1 - \lambda & \text{if } i = 1 \\ (\alpha_i - \lambda)f_{i-1}(\lambda) - \beta_{i-1}^2 f_{i-2}(\lambda) & \text{otherwise} \end{cases} \tag{17}$$

Here, agreement corresponds to $f_i(\lambda)f_{i-1}(\lambda) > 0$ and, for example, if signs of the sequence are $(+, -, -, +)$, we have $a(\lambda) = 1$; moreover, $a(\lambda) = 2$ holds if $(+, -, -, -)$[1]. It requires $O(m)$ time to compute sequence $(f_0(\lambda), f_1(\lambda), \ldots, f_m(\lambda))$ from Equation (17). The bisection method finds eigenvalues in an interval based on the property that $a(\lambda)$ is equal to the number of eigenvalues not smaller than λ [7].

As described in Section 2, we need to compute the $r+1$ largest eigenvalues of $\mathbf{M}$ to obtain hash codes. The largest eigenvalue of $\mathbf{M}$ is 1 (i.e., $\sigma_0 = 1$) and the smallest eigenvalue is larger than 0 [33, 50]. Therefore, to obtain eigenvalue σ_1, a naive approach based on the bisection method is to set $\overline{\sigma}_1 = 1$ and $\underline{\sigma}_1 = 0$, and then iteratively update $\overline{\sigma}_1$ and $\underline{\sigma}_1$ following $(\overline{\sigma}_1 + \underline{\sigma}_1)/2$ until it has $a(\underline{\sigma}_1) = 2$ and $a(\overline{\sigma}_1) = 1$. After convergence, we accurately compute σ_1 by exploiting inverse iteration, where we set $\tilde{\sigma}_1 = (\overline{\sigma}_1 + \underline{\sigma}_1)/2$ as described in the next section. Similarly, we can compute σ_i ($2 \leq i \leq r$) by initializing $\overline{\sigma}_i = \underline{\sigma}_{i-1}$ and $\underline{\sigma}_i = 0$. However, this naive approach needs a large number of iterations to update $\overline{\sigma}_i$ and $\underline{\sigma}_i$. This is because all eigenvalues used in hash codes are larger than 0, and thus $\underline{\sigma}_i = 0$ is not so tight as the left bound for σ_i, whereas we

[1] If $f_i(\lambda) = 0$, assuming that $f_i(\lambda)$ has the same sign as $f_{i-1}(\lambda)$.

can effectively obtain the right bound by setting $\overline{\sigma}_i = \underline{\sigma}_{i-1}$ since $\underline{\sigma}_{i-1} > \sigma_i$ and $\underline{\sigma}_{i-1} \approx \sigma_i$ in practice.

To initialize $\underline{\sigma}_i$, we use a graph clustering algorithm in computing eigenvalues in the order of $\sigma_1, \sigma_2, \ldots, \sigma_r$. Since σ_i is the $(i+1)$-th largest eigenvalue of $\mathbf{T}$, if $\mathbf{V}'_i$ is an $m \times (i+1)$ matrix of eigenvectors, $\mathbf{V}'_i$ gives the optimal solution for the following problem [19];

$$\max_{\mathbf{V}'_i} \operatorname{tr}\left((\mathbf{V}'_i)^\top \mathbf{T}\mathbf{V}'_i\right) \quad \text{s.t. } (\mathbf{V}'_i)^\top \mathbf{V}_i = \mathbf{I} \tag{18}$$

In this equation, we have $\operatorname{tr}\left((\mathbf{V}'_i)^\top \mathbf{T}\mathbf{V}_i\right) = \sum_{j=0}^{i} \sigma_j$ where $\sigma_0 = 1$. To obtain $\underline{\sigma}_i$, we compute $m \times (i+1)$ matrix $\mathbf{H}_i$ such that $\mathbf{H}_i^\top \mathbf{H}_i = \mathbf{I}$ by using a graph clustering algorithm for the set of data points $\mathbb{X}$. Specifically, we compute $\underline{\sigma}_i$ as follows:

Definition 1 (Left Bound). *If $\mathbb{X}$ is partitioned into $i+1$ clusters $\mathbb{X}_1, \ldots, \mathbb{X}_{i+1}$ by a graph clustering algorithm, the j-th column vector $\mathbf{h}_j$ of matrix $\mathbf{H}_i$ is given as follows:*

$$h_j[k] = \begin{cases} 1/\sqrt{|\mathbb{X}_j|} & \text{if } \mathbf{x}_k \in \mathbb{X}_j \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

where $|\mathbb{X}_j|$ is the number of data points included in cluster $\mathbb{X}_j$ and the columns in $\mathbf{H}_i$ are orthonormal to each other; $\mathbf{H}_i^\top \mathbf{H}_i = \mathbf{I}$. We compute left bound $\underline{\sigma}_i$ as follows:

$$\underline{\sigma}_i = \operatorname{tr}\left(\mathbf{H}_i^\top \mathbf{T}\mathbf{H}_i\right) - \sum_{j=0}^{i-1} \sigma_j \tag{20}$$

We have the following property for left bound $\underline{\sigma}_i$:

Lemma 2 (Left Bound). *For tridiagonal matrix $\mathbf{T}$, $\underline{\sigma}_i \leq \sigma_i$ holds.*

This lemma indicates that we can compute left bound $\underline{\sigma}_i$ by using a graph clustering algorithm to compute $\tilde{\sigma}_i$.

While several graph clustering approaches have been proposed, such as METIS [25] and the Girvan-Newman algorithm [38], we adopt the ratio cut algorithm [16] since its object function equals $\operatorname{tr}\left(\mathbf{H}_i^\top \mathbf{T}\mathbf{H}_i\right)$ [50]. Specifically, if R_i is the objective function of the ratio cut algorithm used to obtain $i+1$ clusters, we have $R_i = \operatorname{tr}\left(\mathbf{H}_i^\top \mathbf{T}\mathbf{H}_i\right)$. As a result, we can compute $\underline{\sigma}_i$ by using R_i instead of $\operatorname{tr}\left(\mathbf{H}_i^\top \mathbf{T}\mathbf{H}_i\right)$ in Equation (20). Intuitively, the ratio cut algorithm partitions a graph by cutting edges in the graph so that edges between different clusters have low weights, and edges within clusters have high weights. The objective function of the ratio cut algorithm to obtain $i+1$ clusters is given as follows:

$$R_i = \sum_{j=1}^{i+1} \frac{\operatorname{cut}(\mathbb{X}_\mathrm{j}, \mathbb{X} \backslash \mathbb{X}_\mathrm{j})}{|\mathbb{X}_j|} \tag{21}$$

In this equation, $\operatorname{cut}(\mathbb{X}_\mathrm{j}, \mathbb{X} \backslash \mathbb{X}_\mathrm{j})$ corresponds to the sum of edge weights of data points between $\mathbb{X}_j$ and $\mathbb{X} \backslash \mathbb{X}_j$ given as

$$\operatorname{cut}(\mathbb{X}_\mathrm{j}, \mathbb{X} \backslash \mathbb{X}_\mathrm{j}) = \frac{1}{2} \sum_{\mathbf{x}_\mathrm{k} \in \mathbb{X}_\mathrm{j}, \mathbf{x}_{\mathrm{k}'} \in \mathbb{X} \backslash \mathbb{X}_\mathrm{j}} \mathrm{T}[\mathrm{k}][\mathrm{k}'] \tag{22}$$

The original ratio cut algorithm finds the cut that minimizes the objective function since clusters should be large groups of data points [16]. However, we modify the algorithm to find the cut that maximizes the objective function. This is because we can effectively obtain tight bound $\underline{\sigma}_i$ by using a large value of R_i from Equation (20).

We efficiently compute clusters based on the observation that tridiagonal matrix $\mathbf{T}$ corresponds to the chain graph of Figure 2, where β_j is the edge weight between data points. In obtaining $i+1$ clusters, we find a cut from the chain graph of i clusters; we cut edges in the chain graph one by one to obtain clusters. In particular, if $\Delta R_i(\beta_j)$ is an increase in the objective function by cutting the edge that corresponds to β_j, we compute the cut by identifying the edge that maximizes $\Delta R_i(\beta_j)$ in the chain graph of i clusters. As shown in Figure 3, if $\mathbb{X}_j$ is the cluster that includes the edge corresponding to β_j, β_k is edge weight between cluster $\mathbb{X}_j$ and $\mathbb{X}_{j-1}$, and $\beta_{k'}$ is edge weight between cluster $\mathbb{X}_j$ and $\mathbb{X}_{j+1}$, $\Delta R_i(\beta_j)$ is computed as follows from Equation (21):

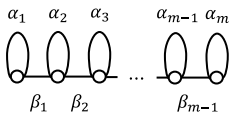


Figure 2: Chain graph of tridiagonal matrix T.

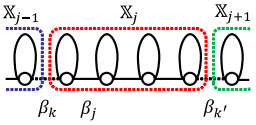


Figure 3: Clusters in the chain graph.

$$\Delta R_i(\beta_j) = \frac{\beta_k + \beta_j}{\operatorname{left}(\beta_\mathrm{j})} + \frac{\beta_j + \beta_{k'}}{\operatorname{right}(\beta_\mathrm{j})} - \frac{\beta_k + \beta_{k'}}{|\mathbb{X}_j|} \tag{23}$$

In this equation, $\operatorname{left}(\beta_\mathrm{j})$ and $\operatorname{right}(\beta_\mathrm{j})$ are the number of data points in the left and right sides of the edge corresponding to β_j in cluster $\mathbb{X}_j$, respectively. In the case of Figure 3, $\operatorname{left}(\beta_\mathrm{j}) = 1$ and $\operatorname{right}(\beta_\mathrm{j}) = 3$ since $|\mathbb{X}_j| = 4$. Note that we can incrementally update $\operatorname{left}(\beta_\mathrm{j})$ and $\operatorname{right}(\beta_\mathrm{j})$ in $O(|\mathbb{X}_j|)$ time from the clustering results. It also takes $O(m-i)$ time to find the cut that maximizes $\Delta R_i(\beta_j)$ from Equation (23). Since $\beta_j \geq 0$ from Equation (10), $\Delta R_i(\beta_j) \geq 0$ from Equation (23). We can incrementally compute left bound $\underline{\sigma}_i$ based on the following lemma:

Lemma 3 (Incremental computation). *Left bound $\underline{\sigma}_i$ can be computed in $O(1)$ time as follows:*

$$\underline{\sigma}_i = \underline{\sigma}_{i-1} + \Delta R_i(\beta_j) - \sigma_{i-1} \tag{24}$$

This lemma indicates that we can efficiently compute $\underline{\sigma}_i$ from Equation (24) instead of Equation (20). Note that $\underline{\sigma}_0 = 1$ since $\sigma_0 = 1$. The next section shows our algorithm based on the bisection method with inverse iteration.

3.1.3 Eigenvector Computation. We use inverse iteration to accurately compute the eigenvector-eigenvalue pair $(\mathbf{v}'_i, \sigma_i)$ of $\mathbf{T}$ from approximate eigenvalue $\tilde{\sigma}_i$ obtained by the bisection method. Inverse iteration is a variant of the power method based on the property that inverse matrix $(\mathbf{T} - \tilde{\sigma}_i \mathbf{I})^{-1}$ has eigenvector-eigenvalue pair $(\mathbf{v}'_i, \frac{1}{\sigma_i - \tilde{\sigma}_i})$ [44]. Since $\tilde{\sigma}_i$ is an approximation of σ_i such that $\underline{\sigma}_i \leq \tilde{\sigma}_i \leq \overline{\sigma}_i$, $\frac{1}{\sigma_i - \tilde{\sigma}_i}$ is larger than $\frac{1}{\sigma_j - \tilde{\sigma}_i}$ for any eigenvalue $\sigma_j \neq \sigma_i$. Therefore, $\frac{1}{\sigma_i - \tilde{\sigma}_i}$ is the largest eigenvalue of $(\mathbf{T} - \tilde{\sigma}_i \mathbf{I})^{-1}$. In general, inverse iteration computes eigenvector-eigenvalue pair $(\mathbf{v}'_i, \sigma_i)$ by applying the power method to $(\mathbf{T} - \tilde{\sigma}_i \mathbf{I})^{-1}$ [7].

Although matrix $\mathbf{T} - \tilde{\sigma}_i \mathbf{I}$ is sparse, it requires $O(m^3)$ time to compute its inverse since $(\mathbf{T} - \tilde{\sigma}_i \mathbf{I})^{-1}$ is a dense matrix [19]. We compute the LU decomposition of $\mathbf{T} - \tilde{\sigma}_i \mathbf{I}$ to obtain eigenvector-eigenvalue pairs since it has desirable properties, as revealed in this section. LU decomposition factorizes a matrix into the product of a lower triangular matrix and an upper triangular matrix [41]. Specifically, if $\mathbf{T}' = \mathbf{T} - \tilde{\sigma}_i \mathbf{I}$, we compute $\mathbf{T}' = \mathbf{L}\mathbf{U}$ where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices, respectively. We have the following property for LU decomposition of $\mathbf{T}'$:

Lemma 4 (Zero Elements). *In the i-th row of $\mathbf{L}$ where $i \geq 3$, we have $L[i][k] = 0$ if $1 \leq k \leq i-2$ holds. Besides, in the j-th column of $\mathbf{U}$ where $j \geq 3$, we have $U[k][j] = 0$ if $1 \leq k \leq j-2$ holds.*

Algorithm 1 Eigenvector Computation

Input: tridiagonal matrix $\mathbf{T}$, orthonormal matrix $\mathbf{P}$
Output: eigenvector-eigenvalue pairs $\{(\mathbf{v}_i, \sigma_i)\}_{i=1}^{r}$

1: $\underline{\sigma}_0 = 1$;
2: $\mathbb{E} = \{e(\beta_i)\}_{i=1}^{m-1}$;
3: **for** $i = 1$ to r **do**
4: **for** each $e(\beta_j) \in \mathbb{E}$ **do**
5: compute $\Delta R_i(\beta_j)$ from Equation (23);
6: $e(\beta_k) = \text{argmax}_{\mathbb{E}}\{\Delta R_i(\beta_j)\}$;
7: cut $e(\beta_k)$ in the chain graph;
8: subtract $e(\beta_k)$ from $\mathbb{E}$;
9: $\overline{\sigma}_i = \underline{\sigma}_{i-1}$;
10: $\underline{\sigma}_i = \underline{\sigma}_{i-1} + \Delta R_i(\beta_j) - \sigma_{i-1}$;
11: **repeat**
12: $\lambda = \frac{\overline{\sigma}_i + \underline{\sigma}_i}{2}$;
13: **if** $a(\lambda) \leq i + 1$ **then**
14: $\underline{\sigma}_i = \lambda$;
15: **else**
16: $\overline{\sigma}_i = \lambda$;
17: **until** $a(\lambda) = i + 1$
18: $\tilde{\sigma}_i = \frac{\overline{\sigma}_i + \underline{\sigma}_i}{2}$;
19: pick a random column vector $\mathbf{w}$ of length m;
20: $\mathbf{w} = \mathbf{w}/\|\mathbf{w}\|$;
21: compute $\mathbf{L}$ and $\mathbf{U}$ from $\mathbf{T} - \tilde{\sigma}_i\mathbf{I}$;
22: **repeat**
23: solve $\mathbf{L}\mathbf{w}' = \mathbf{w}$ for $\mathbf{w}'$ by forward substitution;
24: solve $\mathbf{U}\mathbf{v}' = \mathbf{w}'$ for $\mathbf{v}'$ by back substitution;
25: $\mathbf{v}' = \mathbf{v}'/\|\mathbf{v}'\|$;
26: $\mathbf{w} = \mathbf{v}'$;
27: **until** $\mathbf{v}'$ reaches convergence
28: $\mathbf{v}_i = \mathbf{P}\mathbf{v}'$;
29: $\sigma_i = (\mathbf{v}')^\top\mathbf{T}\mathbf{v}'$;

Since $\mathbf{L}$ is a lower triangular matrix, $L[i][k] = 0$ if $k \geq i+1$ for its i-th row. Therefore, Lemma 4 indicates that $L[i][k]$ can be nonzero only if $i - 1 \leq k \leq i$ in the i-th row; $\mathbf{L}$ has at most two nonzero elements in each row. As a result, the number of nonzero elements in $\mathbf{L}$ of size $m \times m$ is $O(m)$. Similarly, the number of nonzero elements in $\mathbf{U}$ is $O(m)$. From Lemma 4, we have the following property;

Lemma 5 (LU Decomposition). *It takes $O(m)$ time to compute $\mathbf{L}$ and $\mathbf{U}$ from $\mathbf{T}'$.*

In general, it takes $O(m^3)$ time to compute LU decomposition [41]. However, this lemma shows that we can efficiently compute the LU decomposition of $\mathbf{T}'$.

Algorithm 1 shows how to compute eigenvector-eigenvalue pair $(\mathbf{v}_i, \sigma_i)$ of $\mathbf{M}$ based on the bisection method and inverse iteration. Algorithm 1 consists of three parts; graph clustering, bisection method, and inverse iteration. The graph clustering part (lines 4-8), partitions the chain graph into $i + 1$ clusters. The bisection method part computes approximate eigenvalues (lines 9-18). The inverse iteration part computes eigenvector-eigenvalue pairs (lines 19-29).

Algorithm 1 first initializes $\underline{\sigma}_0 = 1$ and $\mathbb{E} = \{e(\beta_i)\}_{i=1}^{m-1}$ where $e(\beta_i)$ is an edge in the chain graph corresponding to β_i (lines 1-2). The graph clustering part computes $\Delta R_i(\beta_j)$ for each edge (lines 4-5) and cuts an edge in the chain graph based on $\Delta R_i(\beta_j)$ (lines 6-7). The bisection method part initializes $\overline{\sigma}_i$ and $\underline{\sigma}_i$ (lines 9-10), and updates them until only a single eigenvalue lies in the interval $[\underline{\sigma}_i, \overline{\sigma}_i]$ (lines 11-17). It then computes $\tilde{\sigma}_i = \frac{\overline{\sigma}_i + \underline{\sigma}_i}{2}$ (line 18). The inverse iteration part initializes random vector $\mathbf{w}$ whose L_2-norm is 1 (line 19-20), and computes the LU decomposition of $\mathbf{T} - \tilde{\sigma}_i\mathbf{I}$ (line 21). It iteratively uses forward and back substitutions to compute eigenvector $\mathbf{v}'$ (lines 22-27); forward and back substitutions iteratively solve a system of linear algebraic equations by using lower and upper triangular matrices, respectively [41]. It then computes the eigenvector-eigenvalue pair $(\mathbf{v}_i, \sigma_i)$ of $\mathbf{M}$ from eigenvector $\mathbf{v}'$ (lines 28-29). We have the following property for Algorithm 1;

Lemma 6 (Eigenvector Computation). *If t_B and t_I are the numbers of iterations in the bisection method and inverse iteration, respectively, Algorithm 1 requires $O(m(t_B+t_I+m))$ time to compute the eigenvector-eigenvalue pair $(\mathbf{v}_i, \sigma_i)$ of matrix $\mathbf{M}$ from tridiagonal matrix $\mathbf{T}$ and its orthonormal matrix $\mathbf{P}$.*

Unlike the power method that takes $O(m^3)$ time, we can efficiently compute the eigenvectors of $\mathbf{M}$ as shown in Lemma 6. Therefore, we can effectively reduce the computation cost of the offline phase of anchor graph hashing.

3.2 Online Phase Computation

This section describes our approach for the online phase. As described in Section 2, we need to compute vector $\mathbf{z}(\mathbf{x})$ for query data point $\mathbf{x}$ in the online phase. The i-th element of vector $\mathbf{z}(\mathbf{x})$ has a nonzero element if the i-th anchor point is one of the closest anchor points of $\mathbf{x}$; it has a zero element otherwise. However, it needs a high computation cost of $O(md)$ time to find the closest anchor points due to the data points' high dimensionality.

In finding the closest anchor points, we use SVD (Singular Value Decomposition) to approximate distances since it gives the smallest error in approximating high-dimensional data [45]. For query data point $\mathbf{x} = [x[1], \ldots, x[d]]$ with d dimensions, we can use SVD of rank d' to obtain its approximation $\tilde{\mathbf{x}} = [\tilde{x}[1], \ldots, \tilde{x}[d']]$ of d' dimensions ($d' \leq d$). We apply SVD to the matrix of m anchor points $[\mathbf{u}_1, \ldots, \mathbf{u}_m]^\top$, which takes $O(md \log d' + (m+d)(d')^2)$ time [17]. If $\mathbf{Q}$ is a $d \times d'$ orthonormal matrix obtained by SVD, $\tilde{\mathbf{x}}$ is computed as $\tilde{\mathbf{x}} = \mathbf{x}\mathbf{Q}$. Similarly, if $\tilde{\mathbf{u}}_i$ is the approximate vector of length d' for anchor point $\mathbf{u}_i$, $\tilde{\mathbf{u}}_i$ is computed as $\tilde{\mathbf{u}}_i = \mathbf{u}_i\mathbf{Q}$. We compute the approximate distance as follows:

Definition 2 (Approximate Distance). *If we have $\|\mathbf{x}'\| = \sqrt{\|\mathbf{x}\|^2 - \|\tilde{\mathbf{x}}\|^2}$ and $\|\mathbf{u}_i'\| = \sqrt{\|\mathbf{u}_i\|^2 - \|\tilde{\mathbf{u}}_i\|^2}$, approximate distance $\tilde{D}(\mathbf{x}, \mathbf{u}_i)$ between query data point $\mathbf{x}$ and anchor point $\mathbf{u}_i$ is given by*

$$\tilde{D}(\mathbf{x}, \mathbf{u}_i) = \sqrt{\|\mathbf{x}\|^2 + \|\mathbf{u}_i\|^2 - 2\langle\tilde{\mathbf{x}}, \tilde{\mathbf{u}}_i\rangle - 2\|\mathbf{x}'\|\|\mathbf{u}_i'\|\cos(\theta_{\mathbf{u}_i',\mathbf{a}'} - \theta_{\mathbf{x}',\mathbf{a}'})} \quad (25)$$

In this equation, $\theta_{\mathbf{u}_i',\mathbf{a}'}$ and $\theta_{\mathbf{x}',\mathbf{a}'}$ are given as follows:

$$\theta_{\mathbf{u}_i',\mathbf{a}'} = \cos^{-1}\frac{\langle\mathbf{u}_i,\mathbf{a}\rangle - \langle\tilde{\mathbf{u}}_i,\tilde{\mathbf{a}}\rangle}{\|\mathbf{u}_i'\|\|\mathbf{a}'\|}, \; \theta_{\mathbf{x}',\mathbf{a}'} = \cos^{-1}\frac{\langle\mathbf{x},\mathbf{a}\rangle - \langle\tilde{\mathbf{x}},\tilde{\mathbf{a}}\rangle}{\|\mathbf{x}'\|\|\mathbf{a}'\|} \quad (26)$$

where $\mathbf{a} \in \mathbb{U}$ is a reference anchor point, $\tilde{\mathbf{a}}$ is an approximate vector of $\mathbf{a}$, and $\|\mathbf{a}'\| = \sqrt{\|\mathbf{a}\|^2 - \|\tilde{\mathbf{a}}\|^2}$

Note that we subject anchor points to SVD in the offline process. Similarly, in the offline process, we compute $\theta_{\mathbf{u}_i',\mathbf{a}'}$ by setting $\mathbf{a} = \mathbf{u}_1, \ldots, \mathbf{u}_m$ for anchor points in $O(m^2 d)$ time and compute $\|\mathbf{u}_i\|$ for each anchor point in $O(md)$ time. Therefore, it takes $O(d')$ time to

Algorithm 2 Closest Anchor Points

Input: query data point $\mathbf{x}$, set of anchor points $\mathbb{U}$, orthonormal matrix $\mathbf{Q}$
Output: set of the s closest anchor points $\mathbb{C}$

1: $\mathbb{C} = \emptyset$;
2: add s dummy anchor points to $\mathbb{C}$;
3: $\mathbf{a} = \mathbf{u}_0$;
4: $\tilde{\mathbf{x}} = \mathbf{x}\mathbf{Q}$;
5: compute $\|\mathbf{x}\|$, $\|\mathbf{x}'\|$, and $\theta_{\mathbf{x}',\mathbf{a}'}$;
6: **for** $i = 1$ to m **do**
7: compute $\tilde{D}(\mathbf{x}, \mathbf{u}_i)$ for $\mathbf{u}_i$;
8: **if** $\tilde{D}(\mathbf{x}, \mathbf{u}_i) \leq \max_{\mathbb{C}}\{D(\mathbf{x}, \mathbf{u}_j)\}$ **then**
9: compute $D(\mathbf{x}, \mathbf{u}_i)$ for $\mathbf{u}_i$;
10: **if** $D(\mathbf{x}, \mathbf{u}_i) \leq \max_{\mathbb{C}}\{D(\mathbf{x}, \mathbf{u}_j)\}$ **then**
11: add $\mathbf{u}_i$ to $\mathbb{C}$;
12: subtract $\mathbf{u}_j = \text{argmax}_{\mathbb{C}}\{D(\mathbf{x}, \mathbf{u}_j)\}$ from $\mathbb{C}$;
13: **if** $\theta_{\mathbf{u}'_i,\mathbf{x}'} \leq \theta_{\mathbf{a}',\mathbf{x}'}$ **then**
14: $\mathbf{a} = \mathbf{u}_i$;
15: compute $\theta_{\mathbf{x}',\mathbf{a}'}$;

compute approximated distance $\tilde{D}(\mathbf{x}, \mathbf{u}_i)$ in the online phase. We introduce the following lemma for the approximate distance:

Lemma 7 (Approximate Distance). *We have the following property for the approximate distance:*

$$\tilde{D}(\mathbf{x}, \mathbf{u}_i) \leq D(\mathbf{x}, \mathbf{u}_i) \tag{27}$$

This lemma indicates that $\tilde{D}(\mathbf{x}, \mathbf{u}_i)$ has the lower bounding property for $D(\mathbf{x}, \mathbf{u}_i)$, which enables us to exactly compute the closest anchor points for query data point $\mathbf{x}$ [45].

Algorithm 2 shows the approach to finding the closest anchor points. It first initializes the set of the closest anchor points by adding s dummy anchor points whose distances to $\mathbf{x}$ are ∞ (line 1-2). It sets the reference anchor point as $\mathbf{a} = \mathbf{u}_0$ such that $\cos(\theta_{\mathbf{u}'_i,\mathbf{u}'_0} - \theta_{\mathbf{x}',\mathbf{u}'_0}) = 1$ for each anchor point (line 3), and computes approximate vector $\tilde{\mathbf{x}}$ (line 4). It then computes $\|\mathbf{x}\|$, $\|\mathbf{x}'\|$, and $\theta_{\mathbf{x}',\mathbf{a}'}$ (line 5). It picks up anchor points one by one to compute approximate distances (lines 6-7). If an anchor point can be the closest anchor points, it accurately computes the distance for the anchor point (lines 8-9), and it updates the closest anchor points if necessary (lines 10-12). Since the approximation quality of $\tilde{D}(\mathbf{x}, \mathbf{u}_i)$ improves as the angle between vector $\mathbf{x}'$ and $\mathbf{a}'$ decreases from Equation (25), it changes the reference anchor point based on the angle to the query data point (lines 13-15). For Algorithm 2, we have

Lemma 8 (Closest Anchor Points). *If c is the ratio of the anchor points used to compute the exact distances, Algorithm 2 requires $O((m+d)d'+cmd)$ time to identify the closest anchor point to $\mathbf{x}$.*

Lemma 8 indicates that we can reduce the online phase's computation cost by pruning distance computations when identifying the closest anchor points. Note that we can exploit Algorithm 2 in computing $\mathbf{Z}$ in the offline phase, as shown in the next section.

3.3 Hashing Algorithm

Algorithms 3 and 4 give full descriptions of our offline and online phases. In the offline phase, Algorithm 3 computes SVD of matrix $[\mathbf{u}_1, \ldots, \mathbf{u}_m]^\top$ used in determining the approximate distances (line 1). It then computes the norms of anchor points and angles between

Algorithm 3 Offline Phase of Tridiagonal Hashing

Input: set of data points $\mathbb{X}$, set of anchor points $\mathbb{U}$, number of bits in hash codes r, target rank d'
Output: hash code matrix $\mathbf{Y}$

1: compute rank-d' SVD of matrix $[\mathbf{u}_1, \ldots, \mathbf{u}_m]^\top$;
2: **for** $i = 1$ to m **do**
3: compute $\|\mathbf{u}_i\|$;
4: **for** $j = i$ to m **do**
5: compute $\theta_{\mathbf{u}'_i,\mathbf{a}'}$;
6: **for** $i = 1$ to n **do**
7: compute $\mathbb{C}_i$ of $\mathbf{x}_i$ by Algorithm 2;
8: compute $\mathbf{Z}$ by Equation (2);
9: $\mathbf{\Lambda} = \text{diag}(\mathbf{Z}^\top\mathbf{1})$;
10: compute $\mathbf{T}$ and $\mathbf{P}$ by Equation (12), (15), and (16);
11: compute $\{(\mathbf{v}_i, \sigma_i)\}_{i=1}^r$ by Algorithm 1;
12: $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_r]$;
13: $\mathbf{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_r)$;
14: $\mathbf{W} = \sqrt{n}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}\mathbf{\Sigma}^{-\frac{1}{2}}$;
15: $\mathbf{Y} = \text{sgn}(\mathbf{Z}\mathbf{W})$;

Algorithm 4 Online Phase of Tridiagonal Hashing

Input: query data point $\mathbf{x}$, set of anchor points $\mathbb{U}$, orthonormal matrix $\mathbf{Q}$, hash code matrix $\mathbf{Y}$, matrix $\mathbf{W}$
Output: hash code vector $\mathbf{y}$

1: compute $\mathbb{C}$ of $\mathbf{x}$ by Algorithm 2;
2: compute $\mathbf{z}(\mathbf{x})$ by Equation (5);
3: $\mathbf{y} = \text{sgn}(\mathbf{z}(\mathbf{x})\mathbf{W})$;

anchor points (line 2-5). If $\mathbb{C}_i$ is the set of the closest anchor points to $\mathbf{x}_i$, it computes $\mathbb{C}_i$ of each data points by using Algorithm 2 and computes $\mathbf{Z}$ and $\mathbf{\Lambda}$ from the closest anchor points (lines 6-9). It computes $\mathbf{T}$ and $\mathbf{P}$ and obtains eigenvector-eigenvalue pairs by Algorithm 1 (line 10-11). It then computes $\mathbf{W}$ from $\mathbf{V}$ and $\mathbf{\Sigma}$ (lines 12-14), and computes $\mathbf{Y}$ from $\mathbf{W}$ (line 15). In the online phase, Algorithm 4 computes the closest anchor points to the query data point using Algorithm 2 (line 1). It then computes $\mathbf{z}(\mathbf{x})$ and obtains $\mathbf{y}$ (lines 2-3). The proposed approach has the following properties:

Theorem 1 (Computation Cost). *Our offline phase requires $O(dd'(n+d')+m(cnd+ns+nr+rt_B+rt_I))$ time. Besides, our online phase needs $O(d'(m+d)+m(cd+r))$ time.*

Theorem 2 (Hashing Result). *The proposed approach yields the same results as the original anchor graph hashing algorithm if the fixed-number of bits is used in hash codes to perform an approximate nearest neighbor search.*

Theorem 1 and 2 theoretically indicate that the proposed approach can improve the efficiency of anchor graph hashing while guaranteeing search results' equivalence.

3.4 Extension

This section proposes implementation-oriented approaches to improve our approach's efficiency and search accuracy.

3.4.1 Parallelization. Algorithms 3 and 4 implicitly assume a single thread. However, they are easily parallelized. Let K be the number of threads. To find the closest anchor points, we randomly divide

the anchor points into K sets and identify each set's closest anchor points in parallel. We can efficiently obtain the closest anchor points from the identified anchor points. Moreover, we use block partitioning to perform matrix multiplication efficiently. Specifically, in computing $\mathbf{X} = \mathbf{BC}$, we perform block partitioning on the input matrix $\mathbf{B}$ and $\mathbf{C}$ as follows:

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} \tag{28}$$

Then, we have

$$\mathbf{X} = \begin{bmatrix} \mathbf{B}_{11}\mathbf{C}_{11} + \mathbf{B}_{12}\mathbf{C}_{21} & \mathbf{B}_{11}\mathbf{C}_{12} + \mathbf{B}_{12}\mathbf{C}_{22} \\ \mathbf{B}_{21}\mathbf{C}_{11} + \mathbf{B}_{22}\mathbf{C}_{21} & \mathbf{B}_{21}\mathbf{C}_{12} + \mathbf{B}_{22}\mathbf{C}_{22} \end{bmatrix} \tag{29}$$

Note that the smaller block matrices $\mathbf{B}_{11}, \mathbf{B}_{12}, \ldots, \mathbf{C}_{11}, \mathbf{C}_{12}, \ldots$ can be further partitioned recursively. Since the recursive matrix multiplications and summations can be performed in parallel, efficiency is significantly improved by using multiple threads.

3.4.2 Hash Codes. As shown in Theorem 2, our approach yields the same search results as the original approach if we fix the number of bits in hash codes. To improve search accuracy, we recursively reduce the number of bits in the offline phase. In hashing, the similarity of data points of the same hash codes becomes more similar as code length increases. However, since hash space size exponentially increases as the number of bits increases, we can fail to find similar data points if we use long codes. To alleviate this problem, we adaptively use several different bit numbers in finding similar data points. Specifically, we first perform an approximate nearest neighbor search using hash codes of r bits. If we do not find any data points, we reduce hash code length by c bits, which corresponds to the smallest eigenvalues used in anchor graph hashing. We iteratively perform this procedure until we find a data point, or the number of bits is smaller than c. We have the following property for the search result of the recursive approach:

Theorem 3 (No False Negative). *Let $\mathbb{S}[\mathbf{x}]$ and $\mathbb{S}'[\mathbf{x}]$ be sets of similar data points for query data point $\mathbf{x}$ output by the original and recursive approaches, respectively. We have $\mathbb{S}'[\mathbf{x}] \supseteq \mathbb{S}[\mathbf{x}]$.*

This theorem indicates that, if a data point is found by the original approach, the data point must be found by the recursive approach; this guarantees no false negatives in the search results. Note that, only if the original approach cannot find any data point do we have $\mathbb{S}'[\mathbf{x}] \supseteq \mathbb{S}[\mathbf{x}]$; otherwise $\mathbb{S}'[\mathbf{x}] = \mathbb{S}[\mathbf{x}]$.

4 RELATED WORK

Hashing techniques can be classified into data-independent or data-dependent. LSH is the representative data-independent method as it computes hash codes by random projections [13]. However, LSH needs long hash codes to achieve high search accuracy [14]. Recently, data-dependent methods have begun to attract more attention. Spectral hashing is one of the most popular data-dependent methods [52]. It constructs the similarity matrix of data points and solves the hashing problem via spectral decomposition, which is inefficient for large-scale data. The anchor graph can efficiently compute the similarity matrix [33], and it is used in recent hashing techniques. For example, large graph hashing with spectral rotation is a recent anchor graph-based hashing technique [31]. To compute hash codes, it transforms the solution to spectral relaxation of the hashing problem by using an orthonormal matrix and then updates the orthonormal matrix from hash codes by using SVD. This approach iteratively performs these procedures to determine hash codes of query data points from the eigenvectors of the adjacency matrix. Discrete spectral hashing computes the spectral solution with the rotation technique and generates hash codes under the required binary constraint of hash codes [21]. It iteratively computes hash codes by using generalized power iteration [39], where it recursively transforms a matrix obtained from the anchor graph by using an orthonormal matrix obtained by SVD.

As described in Section 3, the divide-and-conquer algorithm computes all eigenvalues of a tridiagonal matrix. It recursively divides the tridiagonal matrix into submatrices, and performs eigendecomposition by multiplying the orthonormal matrices obtained from the submatrices. The major computation cost of the approach is the matrix multiplication as it needs $O(m^3)$ time [7]. Coakley et al. proposed to use the fast multiple method (FMM) of one dimension to approximately compute the matrix multiplication in $O(m \log m)$ time [5]. However, it is difficult to use their approach rather than our approach. Our approach can accurately compute eigenvalue σ_i since $\frac{1}{\sigma_i - \tilde{\sigma}_i}$ is the largest eigenvalue of inverse matrix $(\mathbf{T} - \tilde{\sigma}_i \mathbf{I})^{-1}$ based on the property that $\underline{\sigma}_i \leq \tilde{\sigma}_i \leq \overline{\sigma}_i$, as described in Section 3.1.3. However, approximated eigenvalues obtained by the FMM-based approach do not have this property. Therefore, the FMM-based approach cannot be used to compute eigenvalues accurately.

The Nyström approach is a technique widely used to approximate eigendecomposition. Locally linear landmarks is a recent Nyström approach [48]. It projects all points by using a locally linear function of nearest sampled points where the locally linear function is obtained from $\mathbf{M}$. Variational Nyström is the state-of-the-art Nyström approach that uses samples of points similar to locally linear landmarks [49]. It computes the eigendecomposition by using the columns of the matrix corresponding to sampled points.

5 EXPERIMENTAL EVALUATION

We performed experiments to show the effectiveness of our approach. In this section, "Proposed", "Anchor", "VN" "LGHSR", and "DSH" represent our approach, the original approach of anchor graph hashing [36], variational Nyström-based approach [49], large graph hashing with spectral rotation [31], and discrete spectral hashing [21], respectively. VN is the state-of-the-art Nyström approach, and we used it to compute hash codes of anchor graph hashing since it can approximately compute eigendecomposition. LGHSR and DSH are state-of-the-art anchor graph hashing techniques as described in the previous section.

We used the connect-4, epsilon, mnist, kitsune, and wesad datasets[2]. connect-4 is a dataset obtained from Connect Four games, a two-player game in which the players take turns dropping colored disks onto the 42 grid spaces formed by seven-columns and six-rows. The objective of the game is to form a line of four same color disks. Each feature corresponds to each grid space occupied by a player or empty; the number of dimensions is 126. The number of data points is 67557, and classes correspond to win, loss, or draw for the first player; the number of classes is three. epsilon is a dataset used in the Pascal large scale learning challenge in 2008. The goal of this

[2] https://www.csie.ntu.edu.tw/~cjlin/libsvm/, https://archive.ics.uci.edu/ml/index.php

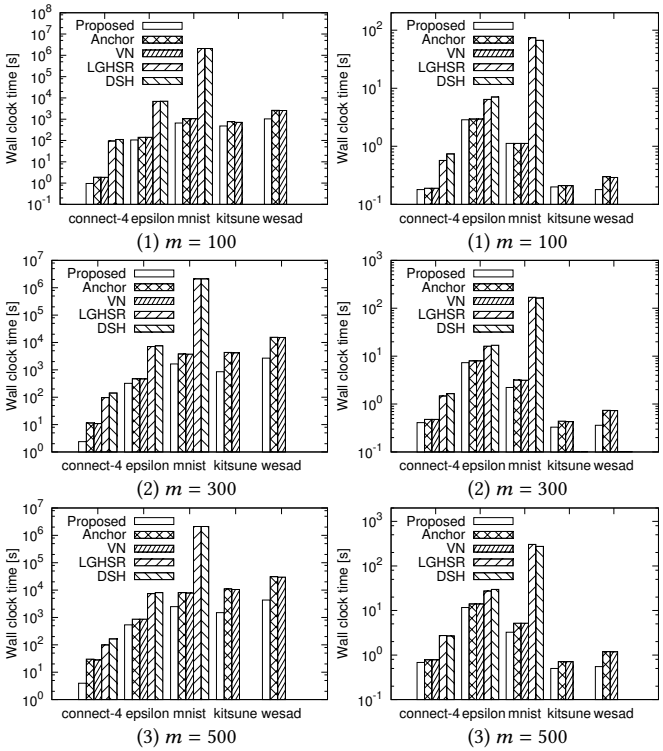


Figure 4: Offline phase time of each approach.

Figure 5: Online phase time of each approach.

challenge is to recognize objects from a number of visual object classes. This dataset includes images obtained from the Flickr website. It contains 500000 objects 2000 numerical features. The number of classes is two in this dataset. mnist is a dataset of handwritten digits. This dataset is a collection of 8100000 grayscale images with 784 features. Each image is one digit from "0" to "9", so the number of classes is ten. kitsune is a cybersecurity dataset containing different network attacks on a commercial IP-based surveillance system and an IoT network. Each data point represents a behavioral snapshot of the hosts and protocols and consists of 115 traffic statistics captured in a temporal window. This dataset includes several network attacks, such as DoS, reconnaissance, man-in-the-middle, and botnet attacks. The numbers of data points and classes are 21017596 and ten, respectively. wesad contains data of subjects during a stress-affect lab study while wearing physiological and motion sensors. It includes sensor modalities such as blood volume pulse, respiration, body temperature, and three-axis acceleration. To effectively perform stress-affect detection, we used the dataset's polynomial features [40]. In particular, we created second-order polynomial features by following the method of [43]. As a result, the number of dimensions is 176. The numbers of data points and classes are 60807600 and eight, respectively.

In each dataset, we randomly sampled 10000 data points to form the query set used in the online phase, and we used the other data points to compute hash codes in the offline phase. We set the number of bits in hash codes, r, to 32 and the number of reduced bit, c, to 8. Besides, we set the bandwidth parameter $t = m^2$ and the number of closest anchor points to $s = 2$ by following a previous

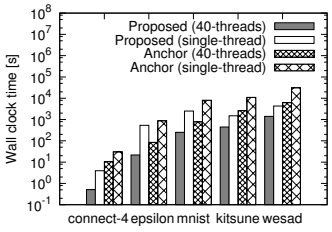


Figure 6: Offline phase time with parallelization.

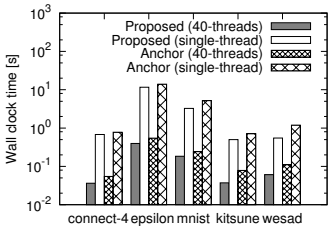


Figure 7: Online phase time with parallelization.

study [36]. For the proposed approach, we set the target rank of SVD to $d' = 30$. We set the number of iterations used in inverse iteration to three by following a previous study [7]. We used the power method to compute eigenvalues except for the proposed approach by setting tolerance to 0.001. For LGHSR and DSH, we set the number of SVD computations to five. We also set the number of generalized power iterations to five for DSH. The number of sampled data points in VN was set to $0.5m$.

All the approaches examined were implemented in C++ with Eigen library. Eigen is a popular C++ template library for linear algebra[3]. Moreover, to parallelize our approach, we implemented it in OpenMP, a multi-threaded coding interface[4]. We conducted all experiments on a Linux server with two Intel(R) Xeon(R) E5-2650 v3 CPUs with 2.30 GHz processors. Each CPU has ten physical cores with hyper-threading, and thus the server has 40 logical cores. The server has 314GB of memory and a 1TB hard disk. We conducted the experiments using a single thread unless otherwise stated.

5.1 Processing Time

We evaluated the processing time. Figure 4 shows the results of the offline phase, and Figure 5 plots the online phase results. The number of anchor points was set to $m = 100$, 300, and 500. Note that the number of anchor points was set to $m = 300$ in the original paper [36]. For kitsune and wesad, we omit the results of LGHSR and DSH since they failed to complete the hash codes within three months. To evaluate the scalability offered by our parallelization approach, we evaluated the times taken by the offline and online phases using 40 threads, see Figure 6 and 7, respectively. In these figures, "Proposed (40-threads)" is for our approach with 40-threads, and "Anchor (40-threads)" is for the original approach parallelized by our approach. "Proposed (single-threads)" and "Anchor (single-threads)" are results using a single thread. In this experiment, we set the number of anchor points to $m = 500$. Figure 1 of Section 1 evaluates each approach with $m = 500$.

As shown in Figure 4 and 5, our approach offers higher efficiency than the previous approaches. Specifically, in the offline phase, it is up to 7.6, 7.0, 3191.4, and 3195.7, times faster than Anchor, VN, LGHSR, and DSH, respectively. Moreover, it is up to 2.2, 2.1, 93.6, and 84.7 times faster than Anchor, VN, LGHSR, and DSH, respectively, in the online phase. As described in Section 2, Anchor needs to compute the eigenvalues of $\mathbf{M}$ in the offline phase and closest anchor points in the online phase. Since it takes times of $O(nm^2)$

[3] http://eigen.tuxfamily.org
[4] http://www.openmp.org/

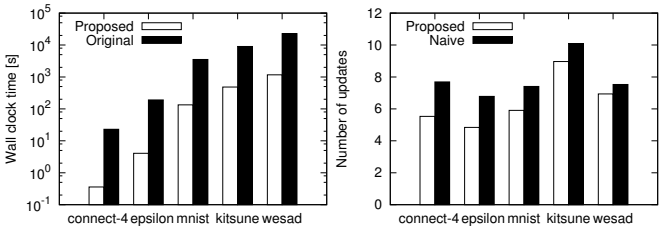


Figure 8: Computation time to obtain tridiagonal matrix.

Figure 9: Number of updates in bisection method.

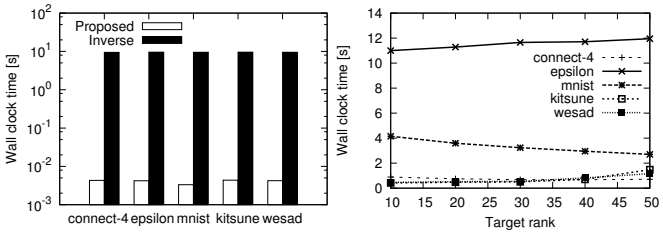


Figure 10: Computation time of inverse iteration.

Figure 11: Online phase time vs. target rank.

and $O(md)$ to compute eigenvalues and closest anchor points, respectively [36, 41], Anchor is slower than our approach. Although VN can efficiently compute eigenvectors, it still needs $O(nm^2)$ time to compute $\mathbf{M}$. Therefore, VN is slower than our approach. As described in Section 4, LGHSR needs to compute the eigendecomposition of the $n \times n$ adjacency matrix of the anchor graph in the offline phase. In the online phase, it computes hash codes from the $n \times r$ dense matrix of obtained eigenvalues. Therefore, LGHSR incurs significantly high computation times compared to our approach. The offline phase of DSH computes the eigendecomposition of the adjacency matrix similar to LGHSR, as described in Section 4. The online phase of DSH computes the hash codes of each query data point using the $n \times r$ dense matrix of the obtained hash codes in the offline phase. Therefore, DSH is much slower than our approach. On the other hand, to compute hash codes, we efficiently compute eigenvectors by applying a graph clustering algorithm to the tridiagonal matrix, as described in Section 3.1. We also compute lower bounds of distances to efficiently identify the closest anchor points for each query data point, as described in Section 3.2. Consequently, our approach is much faster than previous approaches.

As described in Section 3.4.2, our approach can be parallelized by using multiple threads. As shown in Figure 6 and 7, our parallelization approach with 40-thread execution is faster than the proposed approach using a single thread; up to 24.8 times in the offline phase and 29.5 times faster in the online phase. This is because we can efficiently find the closest anchor points by dividing the anchor points into several sets and performing matrix multiplication using block partitioning. Since our approach can be easily parallelized, we can improve its efficiency by using multiple threads.

5.2 Lanczos Method

As described in Section 3.1.1, we compute $\mathbf{T}$ following the Lanczos method by using $\mathbf{b}_i = \mathbf{Z}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{p}_i$. This experiment examined the time taken to compute $\mathbf{T}$ where $m = 500$. In Figure 8, "Original" denotes the original Lanczos method, which gets $\mathbf{T}$ by computing $\mathbf{M}$.

Figure 8 shows that our approach achieves shorter computation times than the original Lanczos method. In particular, it is up to 63.8 times faster than the original Lanczos method because it takes $O(nm^2)$ time to compute $\mathbf{M}$ [36]. The original Lanczos method needs $O(m^3)$ time to compute $\mathbf{T}$ from Equation (9), (10), and (11), as described in Section 3.1.1. On the other hand, we compute $\mathbf{T}$ from $\mathbf{b}_i$ by using Equation (12), (15), and (16) in $O(nms)$ time as shown in Lemma 1 without computing $\mathbf{M}$. As a result, our approach can more efficiently compute $\mathbf{T}$ than the original Lanczos method.

5.3 Bisection Method

As mentioned in Section 3.1.2, we initialize left bound $\underline{\sigma}_i$ by using the ratio cut algorithm to reduce the number of updates for $\underline{\sigma}_i$ needed in the bisection method. We evaluated the average number of updates needed in computing approximate eigenvalue $\tilde{\sigma}_i$ ($1 \leq i \leq r$); we evaluated t_B in this experiment. In Figure 9, "Naive" indicates the results of the approach described in Section 3.1.2, which sets $\underline{\sigma}_i = 0$ based on the property of $\sigma_i > 0$. In each approach, the process of computing $\tilde{\sigma}_i$ sets the right bound to $\overline{\sigma}_i = \underline{\sigma}_{i-1}$. We set the number of anchor points to $m = 500$.

Figure 9 shows that our approach reduced the number of updates by up to 28.6% from the naive approach. Since the smallest eigenvalue of $\mathbf{T}$ is larger than 0 [33, 50], it is theoretically valid to set $\underline{\sigma}_i = 0$. However, since in practice $\sigma_i \approx \underline{\sigma}_{i-1}$, it is difficult to obtain tight left bound $\underline{\sigma}_i$ by setting $\underline{\sigma}_i = 0$. Therefore, the naive approach needs a larger number of updates to produce $\underline{\sigma}_i$. To reduce update computations, we apply the ratio cut algorithm to the chain graph corresponding to $\mathbf{T}$, and effectively compute the left bound for σ_i as shown in Figure 9. Our approach needs $O(mt_B)$ time to compute $\tilde{\sigma}_i$ where t_B is a small number, as shown in Figure 9. Therefore, we can efficiently compute approximate eigenvalues.

5.4 Inverse Iteration

As described in Section 3.1.3, we use inverse iteration to compute the eigenvectors by using $(\mathbf{T} - \tilde{\sigma}_i\mathbf{I})^{-1}$ ($1 \leq i \leq r$). We compute the LU decomposition of $\mathbf{T}' = \mathbf{T} - \tilde{\sigma}_i\mathbf{I}$ to improve efficiency. This experiment examined the processing times needed for inverse iteration where $m = 500$. In Figure 10, "Inverse" is the approach that directly computes $(\mathbf{T} - \tilde{\sigma}_i\mathbf{I})^{-1}$ for obtaining eigenvectors.

As shown in Figure 10, our approach is up to 2867.0 times faster than the comparable approach. Even though $\mathbf{T} - \tilde{\sigma}_i\mathbf{I}$ has a sparse structure, its inversion, $(\mathbf{T} - \tilde{\sigma}_i\mathbf{I})^{-1}$, has a dense structure [19]. Therefore, it takes $O(m^3)$ time to compute $(\mathbf{T} - \tilde{\sigma}_i\mathbf{I})^{-1}$; the comparable approach incurs high computation cost. On the other hand, since the number of nonzero elements is $O(m)$ in the lower and upper triangular matrices obtained by LU decomposition from Lemma 4, we can compute LU decomposition of $\mathbf{T}'$ in $O(m)$ time as shown in Lemma 5. Consequently, we can efficiently compute eigenvectors.

5.5 Singular Value Decomposition

As mentioned in Section 3.2, we exploit the SVD of rank d' for matrix $[\mathbf{u}_1, \ldots, \mathbf{u}_m]^\top$ to compute approximated distances to improve the online phase's efficiency. Since we avoid exact computations of

Table 2: MAE vs. target rank.

d'	connect-4	epsilon	mnist	kitsune	wesad
10	6.53×10^2	3.17×10^2	4.57×10^5	6.15×10^1	3.83×10^2
20	3.78×10^2	3.05×10^2	3.48×10^5	1.50×10^1	1.04×10^2
30	2.40×10^2	2.93×10^2	2.76×10^5	3.22×10^0	2.11×10^1
40	1.20×10^2	2.83×10^2	2.35×10^5	7.01×10^{-1}	5.56×10^0
50	6.17×10^1	2.72×10^2	1.97×10^5	1.49×10^{-1}	1.36×10^0

Table 3: MAP of each approach.

Dataset	m	Proposed	Anchor	VN	LGHSR	DSH
connect-4	100	0.536	0.392	0.335	0.533	0.123
	300	0.551	0.411	0.361	0.541	0.128
	500	0.556	0.412	0.371	0.543	0.149
epsilon	100	0.479	0.249	0.208	0.377	0.192
	300	0.500	0.270	0.249	0.383	0.195
	500	0.505	0.340	0.324	0.390	0.237
mnist	100	0.579	0.578	0.548	0.582	0.345
	300	0.689	0.686	0.662	0.709	0.371
	500	0.732	0.730	0.705	0.759	0.390
kitsune	100	0.798	0.719	0.690	–	–
	300	0.807	0.776	0.767	–	–
	500	0.808	0.793	0.775	–	–
wesad	100	0.443	0.317	0.304	–	–
	300	0.498	0.440	0.433	–	–
	500	0.509	0.458	0.452	–	–

distances in identifying the closest anchor point by using approximated distances, the target rank of SVD is expected to impact the efficiency of our approach. Figure 11 evaluates the processing time of the online phase for different target rank settings. We also evaluated approximation errors of approximated distances by setting different target rank values. Table 2 shows the results using Mean Absolute Error (MAE) as the metric [1]. In these experiments, we set the number of anchor points to $m = 500$.

As shown in Figure 11, our approach's processing time did not vary much with SVD target rank; it is not so sensitive to rank d'. Table 2 shows that the mean absolute errors of approximated distances decrease as the target rank increases. As described in Section 3.2, it takes $O(d')$ time to compute approximated distance by SVD. Therefore, if d' is small, we can efficiently compute approximated distances. On the other hand, we can reduce the number of exact distance computations when d' is large. This is because we can improve the approximation quality by increasing the target rank, as shown in Table 2. Due to this trade-off, derived from target rank d', our approach is not so sensitive to the target rank of SVD.

5.6 Search Accuracy

This section shows that the proposal yields an effective approximate nearest neighbor search in terms of accuracy. Table 3 shows search accuracy when using a hash lookup to find data points with the same label as the query data point. By following the previous study of [36], we used Mean Average Precision (MAP) as the metric of search accuracy; MAP is the mean of the average precision scores for each query data point in finding data points of the same label [1]. Moreover, we evaluated the MAP of our approach for several bit numbers, r, in the hash codes with $m = 500$. In Figure 1 of Section 1, MAP was evaluated by setting $m = 500$ and $r = 32$.

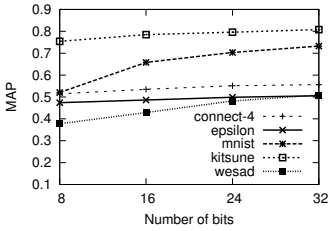


Figure 12: MAP vs. the number of bits.

Table 3 indicates that the search accuracy of our approach improves with the number of anchor points. This is because anchor points represent the data distribution in the high-dimensional space. We note that our approach yields higher search accuracy than the original approach of anchor graph hashing. This is because we use the recursive technique to iteratively reduce hash codes, as described in Section 3.4.2. As shown in Theorem 3, there are no false negatives in the recursive approach's search results. Specifically, if the original approach finds any data points, we have $\mathbb{S}'[\mathbf{x}] = \mathbb{S}[\mathbf{x}]$; the proposed approach finds the same data points. Furthermore, even if the original approach cannot find any data points, our approach can find similar data points since $\mathbb{S}'[\mathbf{x}] \supseteq \mathbb{S}[\mathbf{x}]$ holds. As a result, our approach can more accurately find similar data points than the original approach and indeed other previous approaches. As shown in Figure 12, our approach can more accurately find similar data points as the number of bits increase. The results in Table 3, as well as those in Figure 4 and 5, indicate that the proposed approach increases the efficiency of anchor graph hashing while improving search accuracy. Moreover, our approach's search accuracy can be enhanced by increasing the number of anchor points without a significant increase in computation time.

6 CONCLUSIONS

This paper addressed the problem of reducing the processing time of anchor graph hashing. Our approach computes eigenvectors from the tridiagonal matrix by using the ratio cut algorithm. The proposed approach also uses SVD to compute the closest anchor points efficiently. Experiments show that our approach is significantly faster than existing approaches with improved search accuracy.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant Number 19H04117.

APPENDIX

We provide the proofs omitted from Section 3.

Proof of Lemma 1. Since we can compute $\mathbf{b}_i$ in $O(ns)$ time and α_i is obtained from the L_2-norm of $\mathbf{b}_i$ from Equation (12), we can compute α_i in $O(ns)$ time. In Equation (15) and (16), we can compute $\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{Z}^\top\mathbf{b}_i$ in $O(ns)$ time since the number of nonzero elements in $\mathbf{Z}^\top$ is ns. It also requires $O(n)$ time to compute the inner product $\langle \mathbf{b}_i, \mathbf{b}_{i-1} \rangle$ since the length of $\mathbf{b}_i$ is n. Therefore, we can compute β_i and $\mathbf{p}_i$ in $O(ns)$ time from Equation (15) and (16). As a result, it takes $O(nms)$ time to compute $\mathbf{T}$ and $\mathbf{P}$. □

Proof of Lemma 2. Since $\mathbf{V}'_i$ gives the optimal solution for the problem of Equation (18), we have $\mathrm{tr}\,(\mathbf{H}_i^\top\mathbf{T}\mathbf{H}_i) \leq \mathrm{tr}\,((\mathbf{V}'_i)^\top\mathbf{T}\mathbf{V}_i)$ and

$\mathrm{tr}\left((\mathbf{V}'_i)^\top \mathbf{T}\mathbf{V}_i\right) = \sum_{j=0}^{i} \sigma_j$. Therefore, we have $\sigma_i = \mathrm{tr}\left((\mathbf{V}'_i)^\top \mathbf{T}\mathbf{V}'_i\right) - \sum_{j=0}^{i-1} \sigma_j \geq \mathrm{tr}\left(\mathbf{H}_i^\top \mathbf{T}\mathbf{H}_i\right) - \sum_{j=0}^{i-1} \sigma_j$, which completes the proof. □

Proof of Lemma 3. Since $R_i = \mathrm{tr}\left(\mathbf{H}_i^\top \mathbf{T}\mathbf{H}_i\right)$ holds, left bound $\underline{\sigma}_i$ is computed as $\underline{\sigma}_i = \mathrm{tr}\left(\mathbf{H}_i^\top \mathbf{T}\mathbf{H}_i\right) - \sum_{j=0}^{i-1} \sigma_j = R_i - \sum_{j=0}^{i-1} \sigma_j$. Since $\Delta R_i(\beta_j)$ is the increase in the objective functions by cutting the edge corresponding to β_j, we have $\underline{\sigma}_i - \underline{\sigma}_{i-1} = R_i - R_{i-1} - \sigma_{i-1} = \Delta R_i(\beta_j) - \sigma_{i-1}$. As a result, we can compute $\underline{\sigma}_i$ in $O(1)$ time. □

Proof of Lemma 4. Each element of the lower and upper triangular matrices of LU decomposition of $\mathbf{T}'$ is given as follows [41]:

$$L[i][j] = \begin{cases} 0 & \text{if } i < j \\ 1 & \text{if } i = j \\ \frac{T'[i][j] - \sum_{l=1}^{j-1} L[i][l]U[l][j]}{U[j][j]} & \text{otherwise} \end{cases} \tag{30}$$

$$U[i][j] = \begin{cases} 0 & \text{if } i > j \\ T'[i][j] & \text{if } i = 1 \\ T'[i][j] - \sum_{l=1}^{i-1} L[i][l]U[l][j] & \text{otherwise} \end{cases} \tag{31}$$

We prove Lemma 4 by using mathematical induction.

Initial step: show the statement above holds in the case of $k = 1$. If $k = 1$, we have $L[i][k] = \frac{T'[i][k]}{U[k][k]}$ from Equation (30), and $T'[i][k] = 0$ since $\mathbf{T}' = \mathbf{T} - \tilde{\sigma}_i\mathbf{I}$ and $T[i][k] = 0$ from Equation (8). Therefore, $L[i][k] = 0$. Similarly, if $k = 1$, we have $U[k][j] = T'[k][j] = 0$ from Equation (31).

Inductive step: assume that $L[i][k'] = 0$ and $U[k'][j] = 0$ hold for the k'-th element such that $1 \leq k' \leq k-1$. From Equation (30), we have $L[i][k] = \frac{T'[i][k]}{U[k][k]}$ since $L[i][k'] = 0$ holds for all k' such that $1 \leq k' \leq k-1$. Since $T'[i][k] = 0$ holds, we have $L[i][k] = 0$. Similarly, we have $U[j'][j] = 0$ from Equation (31) since $U[k'][j] = 0$ for all k' such that $1 \leq k' \leq k-1$, which completes the inductive step. □

Proof of Lemma 5. From Lemma 4, $\mathbf{L}$ and $\mathbf{U}$ is computed as

$$L[i][j] = \begin{cases} \frac{T'[i][j]}{U[j][j]} & \text{if } i = j+1 \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{32}$$

$$U[i][j] = \begin{cases} T'[i][j] & \text{if } i = j-1 \\ T'[i][j] - L[i][j-1]U[i-1][j] & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{33}$$

where we assume $L[1][0] = U[0][1] = 0$ in computing $U[1][1]$. Equation (32) and (33) indicate that it takes $O(1)$ time to compute each element of $\mathbf{L}$ and $\mathbf{U}$. Since the number of nonzero elements in $\mathbf{L}$ and $\mathbf{U}$ is $O(m)$, it needs $O(m)$ time to compute $\mathbf{L}$ and $\mathbf{U}$. □

Proof of Lemma 6. From Lemma 3, we can compute $\underline{\sigma}_i$ in $O(1)$ time. Since it takes $O(m)$ to compute $a(\lambda)$, we need $O(mt_B)$ time to compute $\tilde{\sigma}_i$ using the bisection method. From Lemma 5, it takes $O(m)$ time to compute LU decomposition of $\mathbf{T} - \tilde{\sigma}_i\mathbf{I}$. Since the number of nonzero elements in $\mathbf{L}$ and $\mathbf{U}$ is $O(m)$, we need $O(mt_I)$ time to compute $\mathbf{v}'$ using the inverse method. Since the size of $\mathbf{P}$ is $m \times m$, it requires $O(m^2)$ time to compute $\mathbf{v}$ from equation of $\mathbf{v}_i = \mathbf{P}\mathbf{v}'_i$. Since the number of nonzero elements in $\mathbf{T}$ is $O(m)$ as shown in Equation (8), it takes $O(m)$ time to compute σ_i from equation of $\sigma_i = (\mathbf{v}')^\top \mathbf{T}\mathbf{v}'$. As a result, we can compute an eigenvector-eigenvalue pair $(\mathbf{v}_i, \sigma_i)$ of $\mathbf{M}$ in $O(m(t_B+t_I+m))$ time from $\mathbf{T}$. □

Proof of Lemma 7. We first prove that $\tilde{D}(\mathbf{x}, \mathbf{u}_i) \geq 0$ holds. Since $\langle \tilde{\mathbf{x}}, \tilde{\mathbf{u}}_i \rangle \leq \|\tilde{\mathbf{x}}\| \|\tilde{\mathbf{u}}_i\|$ and $\cos(\theta_{\mathbf{u}'_i,\mathbf{a}'} - \theta_{\mathbf{x}',\mathbf{a}'}) \leq 1$, we have

$$\begin{aligned} &\|\mathbf{x}\|^2 + \|\mathbf{u}_i\|^2 - 2\langle \tilde{\mathbf{x}}, \tilde{\mathbf{u}}_i \rangle - 2\|\mathbf{x}'\| \|\mathbf{u}'_i\| \cos(\theta_{\mathbf{u}'_i,\mathbf{a}'} - \theta_{\mathbf{x}',\mathbf{a}'}) \\ \geq &\|\tilde{\mathbf{x}}\|^2 + \|\mathbf{x}'\|^2 + \|\tilde{\mathbf{u}}_i\|^2 + \|\mathbf{u}'_i\|^2 - 2\|\tilde{\mathbf{x}}\| \|\tilde{\mathbf{u}}_i\| - 2\|\mathbf{x}'\| \|\mathbf{u}'_i\| \\ = &(\|\tilde{\mathbf{x}}\| - \|\tilde{\mathbf{u}}_i\|)^2 + (\|\mathbf{x}'\| - \|\mathbf{u}'_i\|)^2 \geq 0 \end{aligned} \tag{34}$$

Therefore, $\tilde{D}(\mathbf{x}, \mathbf{u}_i) \geq 0$ holds from Equation (25). Since SVD is an orthonormal transformation, we have

$$\begin{aligned} &D^2(\mathbf{x}, \mathbf{u}_i) = \sum_{j=1}^{d} (x[j] - u_i[j])^2 \\ = &\|\mathbf{x}\|^2 + \|\mathbf{u}_i\|^2 - 2\sum_{j=1}^{d'} \tilde{x}[j]\tilde{u}_i[j] - 2\sum_{j=d'+1}^{d} \tilde{x}[j]\tilde{u}_i[j] \\ = &\|\mathbf{x}\|^2 + \|\mathbf{u}_i\|^2 - 2\langle \tilde{\mathbf{x}}, \tilde{\mathbf{u}}_i \rangle - 2\|\mathbf{x}'\| \|\mathbf{u}'_i\| \cos\theta_{\mathbf{u}'_i,\mathbf{x}'} \end{aligned} \tag{35}$$

where $\theta_{\mathbf{u}'_i,\mathbf{x}'}$ is the angle between $\mathbf{u}'_i$ and $\mathbf{x}'$. As shown in Equation (26), $\theta_{\mathbf{u}_i,\mathbf{a}'}$ is the angle between $\mathbf{u}'_i$ and $\mathbf{a}'$, and $\theta_{\mathbf{x}',\mathbf{a}'}$ is the angle between $\mathbf{x}'$ and $\mathbf{a}'$. Since we have $\cos\theta_{\mathbf{u}'_i,\mathbf{x}'} \leq \cos(\theta_{\mathbf{u}'_i,\mathbf{a}'} - \theta_{\mathbf{x}',\mathbf{a}'})$, we have $\tilde{D}(\mathbf{x}, \mathbf{u}_i) \leq D(\mathbf{x}, \mathbf{u}_i)$ from Equation (25) and (35). □

Proof of Lemma 8. Since the size of $\mathbf{Q}$ is $d \times d'$, it takes $O(dd')$ time to compute $\tilde{\mathbf{x}} = \mathbf{x}\mathbf{Q}$. It also needs $O(d)$ time to compute $\|\mathbf{x}\|$ and $\|\mathbf{x}'\|$. Therefore, it requires $O((m+d)d')$ time to compute approximate distances for the anchor points. It also takes $O(cmd)$ time to compute $D(\mathbf{x}, \mathbf{u}_i)$ after pruning anchor points by using approximate distances. Therefore, it needs $O((m+d)d'+cmd)$ time to find the closest anchor points to query data point $\mathbf{x}$. □

Proof of Theorem 1. In the offline phase, it takes $O(md \log d' + (m+d)(d')^2)$ time to compute SVD. It takes $O(md)$ and $O(m^2 d)$ time to compute the anchor points' norms and angles between anchor points, respectively. It requires $O((m+d)nd'+cnmd)$ time to obtain $\mathbf{Z}$ and $\mathbf{\Lambda}$ from the closest anchor points. From Lemma 1, we can compute $\mathbf{T}$ and $\mathbf{P}$ in $O(nms)$ time. We can compute r eigenvector-eigenvalue pairs in $O(mr(t_B+t_I+m))$ time from Lemma 6. Since the size of $\mathbf{W}$ is $m \times r$, it needs $O(nmr)$ time to compute $\mathbf{Y}$ of size $n \times r$. As a result, we need $O(dd'(n+d')+m(cnd+ns+nr+rt_B+rt_I))$ time in the offline phase. The online phase takes $O((m+d)d'+cmd)$ time to obtain the anchor points closest to the query data point from Lemma 8. It takes $O(mr)$ time to compute $\mathbf{y}$. Therefore, the online phase of our approach needs $O(d'(m+d)+m(cd+r))$ time. □

Proof of Theorem 2. Our approach uses approximate distances with the lower bounding property in computing the closest anchor points, as shown in Lemma 7. Therefore, it can exactly obtain the closest anchor points [45]. It computes σ_i of $\mathbf{T}$ by inverse iteration from $\tilde{\sigma}_i$ obtained by the bisection method. The corresponding eigenvector $\mathbf{v}_i$ to σ_i can be computed from $\mathbf{P}$. Since $\mathbf{T}$ and $\mathbf{M}$ have the same eigenvalues [7], our approach can accurately compute the r largest eigenvector-eigenvalue pairs $\{(\mathbf{v}_i, \sigma_i)\}_{i=1}^{r}$ of $\mathbf{M}$ from $\mathbf{T}$. □

Proof of Theorem 3. If the original approach finds data points similar to query data points using r bits, the recursive approach also finds the same data points using r bits; $\mathbb{S}[\mathbf{x}] = \mathbb{S}'[\mathbf{x}]$. Otherwise, since the original approach fails to find similar data points, we have $\mathbb{S}[\mathbf{x}] = \emptyset$. Since the recursive approach iterates the approximate nearest neighbor search, $\mathbb{S}'[\mathbf{x}] \supseteq \emptyset = \mathbb{S}[\mathbf{x}]$ holds. □

REFERENCES

[1] S. Margret Anouncia and Uffe Kock Wiil. 2018. *Knowledge Computing and its Applications: Knowledge Computing in Specific Domains: Volume II.* Springer.

[2] Jon Louis Bentley. 1990. K-d Trees for Semidynamic Point Sets. In *SoCG*. 187–197.

[3] Miguel Á. Carreira-Perpiñán and Ramin Raziperchikolaei. 2015. Hashing with Binary Autoencoders. In *CVPR*. 557–566.

[4] Paolo Ciaccia and Marco Patella. 2000. PAC Nearest Neighbor Queries: Approximate and Controlled Search in High-Dimensional and Metric Spaces. In *ICDE*. 244–255.

[5] Ed S. Coakley and Vladimir Rokhlin. 2013. A fast divide-and-conquer algorithm for computing the spectra of real symmetric tridiagonal matrices. *Applied and Computational Harmonic Analysis* 34, 3 (2013), 379 – 414.

[6] Ronald Fagin, Ravi Kumar, and D. Sivakumar. 2003. Efficient Similarity Search and Classification via Rank Aggregation. In *SIGMOD*. 301–312.

[7] William Ford. 2014. *Numerical Linear Algebra with Applications: Using MATLAB*. Academic Press.

[8] Yasuhiro Fujiwara, Yasutoshi Ida, Junya Arai, Mai Nishimura, and Sotetsu Iwamura. 2016. Fast Algorithm for the Lasso based L1-Graph Construction. *Proc. VLDB Endow.* 10, 3 (2016), 229–240.

[9] Yasuhiro Fujiwara, Go Irie, Shari Kuroyama, and Makoto Onizuka. 2014. Scaling Manifold Ranking Based Image Retrieval. *Proc. VLDB Endow.* 8, 4 (2014), 341–352.

[10] Yasuhiro Fujiwara, Atsutoshi Kumagai, Sekitoshi Kanai, Yasutoshi Ida, and Naonori Ueda. 2020. Efficient Algorithm for the b-Matching Graph. In *KDD*. 187–197.

[11] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Yasutoshi Ida, and Machiko Toyoda. 2015. Adaptive Message Update for Fast Affinity Propagation. In *KDD*. 309–318.

[12] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive Hashing Scheme Based on Dynamic Collision Counting. In *SIGMOD*. 541–552.

[13] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *VLDB*. 518–529.

[14] Yuchen Guo, Guiguang Ding, Li Liu, Jungong Han, and Ling Shao. 2017. Learning to Hash With Optimized Anchor Embedding for Scalable Retrieval. *IEEE Trans. Image Processing* 26, 3 (2017), 1344–1354.

[15] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. 47–57.

[16] Lars W. Hagen and Andrew B. Kahng. 1992. New Spectral Methods for Ratio Cut Partitioning and Clustering. *IEEE Trans. on CAD of Integrated Circuits and Systems* 11, 9 (1992), 1074–1085.

[17] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.* 53, 2 (2011), 217–288.

[18] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. 2012. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory of Computing* 8, 1 (2012), 321–350.

[19] David A. Harville. 2008. *Matrix Algebra From a Statistician's Perspective*. Springer.

[20] Michael E. Houle and Jun Sakuma. 2005. Fast Approximate Similarity Search in Extremely High-Dimensional Data Sets. In *ICDE*. 619–630.

[21] Di Hu, Feiping Nie, and Xuelong Li. 2019. Discrete Spectral Hashing for Efficient Similarity Retrieval. *IEEE Trans. Image Process.* 28, 3 (2019), 1080–1091.

[22] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 9, 1 (2015), 1–12.

[23] Yasutoshi Ida, Yasuhiro Fujiwara, and Hisashi Kashima. 2019. Fast Sparse Group Lasso. In *NeurIPS*. 1700–1708.

[24] Yasutoshi Ida, Sekitoshi Kanai, Yasuhiro Fujiwara, Tomoharu Iwata, Koh Takeuchi, and Hisashi Kashima. 2020. Fast Deterministic CUR Matrix Decomposition with Accuracy Assurance. In *ICML*. 4594–4603.

[25] George Karypis and Vipin Kumar. 1999. Parallel Multilevel Series k-Way Partitioning Scheme for Irregular Graphs. *SIAM Rev.* 41, 2 (1999), 278–300.

[26] Norio Katayama and Shin'ichi Satoh. 1997. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *SIGMOD*. 369–380.

[27] Brian Kulis and Kristen Grauman. 2009. Kernelized Locality-sensitive Hashing for Scalable Image Search. In *ICCV*. 2130–2137.

[28] Chen Li, Edward Y. Chang, Hector Garcia-Molina, and Gio Wiederhold. 2002. Clustering for Approximate Similarity Search in High-Dimensional Spaces. *IEEE Trans. Knowl. Data Eng.* 14, 4 (2002), 792–808.

[29] Mingjie Li, Ying Zhang, Yifang Sun, Wei Wang, Ivor W. Tsang, and Xuemin Lin. 2020. I/O Efficient Approximate Nearest Neighbour Search based on Learned Functions. In *ICDE*. 289–300.

[30] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *IEEE Trans. Knowl. Data Eng.* 32, 8 (2020), 1475–1488.

[31] Xuelong Li, Di Hu, and Feiping Nie. 2017. Large Graph Hashing with Spectral Rotation. In *AAAI*. 2203–2209.

[32] Jingjing Liu, Shaoting Zhang, Wei Liu, Xiaofan Zhang, and Dimitris N. Metaxas. 2014. Scalable Mammogram Retrieval Using Anchor Graph Hashing. In *ISBI*. 898–901.

[33] Wei Liu, Junfeng He, and Shih-Fu Chang. 2010. Large Graph Construction for Scalable Semi-Supervised Learning. In *ICML*. 679–686.

[34] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. 2014. Discrete Graph Hashing. In *NIPS*. 3419–3427.

[35] Wanqi Liu, Hanchen Wang, Ying Zhang, Wei Wang, and Lu Qin. 2019. I-LSH: I/O Efficient c-Approximate Nearest Neighbor Search in High-Dimensional Space. In *ICDE*. 1670–1673.

[36] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2011. Hashing with Graphs. In *ICML*. 1–8.

[37] Kejing Lu, Hongya Wang, Wei Wang, and Mineichi Kudo. 2020. VHP: Approximate Nearest Neighbor Search via Virtual Hypersphere Partitioning. *Proc. VLDB Endow.* 13, 9 (2020), 1443–1455.

[38] M.E.J. Newman. 2003. Fast Algorithm for Detecting Community Structure in Networks. *Physical Review E* 69 (2003).

[39] Feiping Nie, Rui Zhang, and Xuelong Li. 2017. A Generalized Power Iteration Method for Solving Quadratic Problem on the Stiefel Manifold. *Sci. China Inf. Sci.* 60, 11 (2017), 112101:1–112101:10.

[40] Paul Pavlidis, Jason Weston, Jinsong Cai, and William Noble Grundy. 2001. Gene functional classification from heterogeneous data. In *RECOMB*. 249–255.

[41] William H. Press, Saul A. Teukolsky, William T Vetterling, and Brian P. Flannery. 2007. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press.

[42] Parikshit Ram and Kaushik Sinha. 2019. Revisiting kd-tree for Nearest Neighbor Search. In *KDD*. 1378–1388.

[43] Volker Roth and Bernd Fischer. 2008. The Group-Lasso for Generalized Linear Models: Uniqueness of Solutions and Efficient Algorithms. In *ICML*. 848–855.

[44] Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics.

[45] Dennis Shasha and Yunyue Zhu. 2004. *High Performance Discovery In Time Series: Techniques And Case Studies*. SpringerVerlag.

[46] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: Solving c-Approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index. *Proc. VLDB Endow.* 8, 1 (2014), 1–12.

[47] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2009. Quality and Efficiency in High Dimensional Nearest Neighbor Search. In *SIGMOD*. 563–576.

[48] Max Vladymyrov and Miguel Á. Carreira-Perpiñán. 2013. Locally Linear Landmarks for Large-Scale Manifold Learning. In *ECML PKDD*. 256–271.

[49] Max Vladymyrov and Miguel Á. Carreira-Perpiñán. 2016. The Variational Nystrom Method for Large-scale Spectral Problems. In *ICML*. 211–220.

[50] Ulrike von Luxburg. 2007. A Tutorial on Spectral Clustering. *Stat. Comput.* 17, 4 (2007), 395–416.

[51] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*. 194–205.

[52] Yair Weiss, Antonio Torralba, and Robert Fergus. 2008. Spectral Hashing. In *NIPS*. 1753–1760.

[53] Yi Zheng, Hui Peng, Xiaocai Zhang, Xiaoying Gao, and Jinyan Li. 2018. Predicting Drug Targets from Heterogeneous Spaces using Anchor Graph Hashing and Ensemble Learning. In *IJCNN*. 1–7.

[54] Y. Zheng, C. Sun, C. Zhu, X. Lan, X. Fu, and W. Han. 2015. LWCS: A Large-scale Web Page Classification System Based on Anchor Graph Hashing. In *ICSESS*. 90–94.