Fast and Scalable Approximate Spectral Graph Matching for Correspondence Problems

U Kang[a,1], Martial Hebert[b], Soonyong Park[c,1,*]

[a] *Computer Science Department, Carnegie Mellon University, Pittsburgh, PA*
[b] *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA*
[c] *Future IT Research Center, Samsung Advanced Institute of Technology (SAIT), South Korea*

Abstract

Establishing consistent correspondences between two sets of features is a fundamental problem in computer vision. This problem can be well formulated as graph matching in which nodes and edges represent feature points and pairwise relationships between feature points, respectively. Spectral matching [19] is the state-of-the-art eigenvector-based method for graph matching. The spectral matching algorithm has been used successfully for small data, but its heavy memory requirement limited the maximum data sizes and contexts it can be used. In this paper, we propose FASM, a fast and scalable approximate spectral matching algorithm. The main ideas are two-fold. First, we exploit the redundancy in the data generation process to approximate the affinity matrix with the linear combination of Kronecker products between bases and index matrices. The bases and index matrices are highly compressed representation of the approximated affinity matrix, requiring much smaller memory than in previous works which store the whole affinity matrix. Second, we compute the eigenvector of the approximated affinity matrix using the small bases and index matrices without explicitly materializing the approximated matrix. Experimental results show that our proposed method is up to 33$\times$ faster, requiring up to 645$\times$ smaller memory than the exact algorithm, with little or no loss of accuracy.

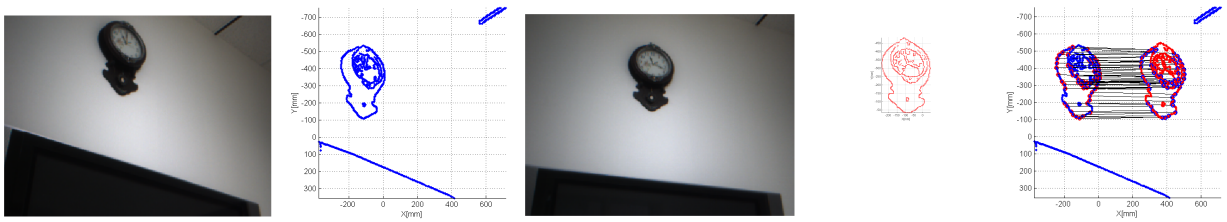
Keywords: spectral graph matching, correspondence problem, approximation algorithm

1. Introduction

Establishing consistent correspondences between two sets of features is a fundamental problem in computer vision. Those include object recognition [5], object category discovery [20], robot localization [23, 26], 2D/3D registration [4, 32], shape matching [1], wide baseline stereo [25], and feature tracking [27]. Given two sets of features P and Q,

*Corresponding author. Tel: +82-31-280-1579
Email addresses: ukang@cs.cmu.edu (U Kang), hebert@ri.cmu.edu (Martial Hebert), soonyong.park@gmail.com (Soonyong Park)

[1] Soonyong Park and U Kang made equal contributions to this paper.



(a) Query image (b) Query points (c) Model image (d) Model points (e) Matching

Figure 1: Example of the correspondence problem. Given a query image (a), query features (points) are extracted as in (b). The query features are then compared with model features (d), which are extracted from a model image (c), to find corresponding pairs as in (e), where horizontal lines indicate matches.

each having n_P and n_Q features respectively, the correspondence problem is to find a set C of corresponding pairs (i, i') where $i \in P$ and $i' \in Q$. Fig. 1 shows an example of the correspondence problem. For instance, a robot navigating inside a building extracts features from the current image that it sees, and compares the extracted features with the feature sets in the object database to identify the nature of the object (e.g., door, desk, clock, etc). The correspondence problem can be formulated as graph matching in which nodes can represent the feature points and edges can encode their pairwise relationships such as distances or angles. Correspondences between the nodes of two graphs are established by considering how well both the unary information on the nodes and the pairwise relationships associated with the edges are satisfied.

Spectral matching [19] is the state-of-the-art eigenvector-based graph matching algorithm successfully used in various tasks [15, 20, 24]. This method builds the affinity matrix M of a graph whose nodes represent the potential correspondences $a = (i, i')$ while the weights on the edges measure the similarity between pairs of potential correspondences. The principal eigenvector of the matrix M is then computed to find corresponding pairs. The spectral matching method, however, has the problem of scalability: the matrix M is very large, and thus the construction of M, as well as the eigenvector computation takes very long. In this paper, we address the research challenge of *scalability* - we show how to perform efficient spectral matching on a very large matrix with billions of nonzero elements by our proposed FASM (FAst Spectral Matching) algorithm. Our contributions are the followings:

1. We observe that there exists high redundancy in the affinity matrix used for the spectral matching. Based on the observation, we approximate the matrix by the linear combination of the Kronecker product of 'bases' and index matrices which are highly compressed representation of the approximated affinity matrix, requiring much smaller memory than in previous works.
2. We develop an efficient algorithm to compute the principal eigenvector of the approximated affinity matrix using the compressed representation without explicitly materializing the approximated matrix.

Symbol	Definition
P	Set of n_P features (points) from the query image.
Q	Set of n_Q features (points) from the model image.
M	$n_P n_Q \times n_P n_Q$ original affinity matrix whose element $M(a,b)$ represents how compatible the pair $a=(i,i')$ is with the pair $b=(j,j')$.
M_{ij}	$n_Q \times n_Q$ submatrix of M, containing the rows $(i-1)n_Q+1 : i \cdot n_Q$ and the columns $(j-1)n_Q+1 : j \cdot n_Q$.
$\hat{M}$	Approximated matrix of M by FASM.
d_{ij}	Distance of two points i and j.
$\hat{d}_{ij}$	Approximated distance of d_{ij} by FASM.
w	Bin width used in FASM.
σ_d	Parameter to control the sensitivity of matching in spectral matching algorithms.

Table 1: Table of symbols.

3. We perform extensive experiments, and show that our FASM is up to $33\times$ faster, requires up to $645\times$ smaller memory than the exact algorithm, with little or no loss of accuracy.

The rest of this paper is organized as follows. Section 2 reviews the standard spectral matching [19]. Section 3 describes our proposed method of approximating the affinity matrix for spectral matching with few bases and index matrices, and explains our proposed method of computing the principal eigenvector using the matrices. Section 4 provides experimental evaluations and comparisons. After reviewing the related works in Section 5, we conclude in Section 6.

To enhance the readability of this paper, we listed the symbols frequently used in this paper in Table 1.

2. Preliminaries; Spectral Matching

In this section, we review the standard spectral matching [19] whose fast solution will be proposed in Section 3.

We consider two sets of features P and Q, each having n_P and n_Q features respectively. Following the original work on spectral matching [19], we use the 2D or 3D coordinates of points as features, and the pairwise distances between feature points as the pairwise relationships in this paper, but extending our work to other pairwise relationships (e.g. angles [15, 20, 24]) is straightforward. The spectral matching first considers all possible correspondence candidates ($i \in P, i' \in Q$), and builds the affinity matrix $M^{n_P n_Q \times n_P n_Q}$ whose element $M(a,b)$ represents how compatible the pair $a=(i,i')$ is with the pair $b=(j,j')$. The pairwise score $M(a,b)$, representing the amount of deformation between two candidate correspondences $a=(i,i')$ and $b=(j,j')$, is determined by

$$M(a,b) = \begin{cases} 4.5 - \frac{(d_{ij} - d_{i'j'})^2}{2\sigma_d^2}, & \text{if } |d_{ij} - d_{i'j'}| < 3\sigma_d; \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

where d_{ij} and $d_{i'j'}$ are the distances between the points i and j, and between their candidate matches i' and j', respectively. σ_d is a parameter to control the sensitivity of the matching: larger σ_d allows more deformations of the data, and more pairwise relationships to have positive scores in the matrix M.

Note that M is a symmetric matrix, and all of its elements are in the range of 0 and 4.5. Intuitively, $M(a,b)$ is big if the two pairs a and b are consistent. For example, in Figs. 2a and 2b, the pairs $(1,A)$ and $(4,D)$ have high value in M since 1 and A, as well as 4 and D, are exact corresponding pairs, making the difference between d_{14} and d_{AD} to almost 0. However, the pairs $(1,A)$ and $(4,C)$ have low value in M since 4 doesn't correspond to C, making the difference between d_{14} and d_{AC} to a nontrivial value. Fig. 2c shows the affinity matrix M of the example graphs in Figs. 2a and 2b.

Given the matrix M, we want to find the best cluster C^* of assignments (i,i') that maximizes the matching score S:

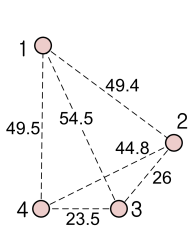
$$S = \sum_{a,b \in C^*} M(a,b) = x^T M x, \quad (2)$$

where x is a binary indicator vector with an element for each correspondence pair $a = (i,i')$ such that $x(a) = 1$ if i in P is matched with i' in Q, and 0 otherwise. The vector x^* maximizing (2) is given by the solution of an Integer Quadratic Program (IQP):

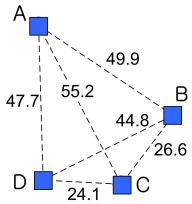
$$x^* = argmax(x^T M x). \quad (3)$$

Relaxing the integral constraints on x by interpreting the value of $x(a)$ as the association of a with the best cluster, and requiring the norm of x to be 1, the solution of (3) is given by the eigenvector of M associated with the largest eigenvalue according to the Rayleigh quotient theorem [17]. Since M contains only nonnegative elements, the elements of x^* will be in $[0,1]$ by Perron-Frobenius theorem [14]. This is the relaxed and continuous vector, so we need to binarize x^* to find discrete assignments. The binarization is performed by repeating the following greedy steps:

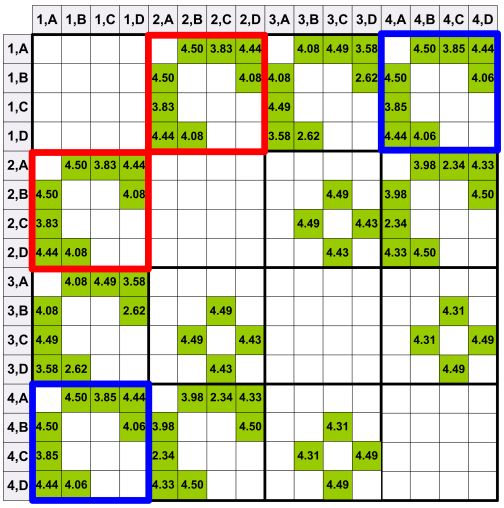
1. Find $a^* = argmax_{a \in L}(x^*(a))$, where L is the set of all tentative correspondences. If $x^*(a^*) = 0$, stop the binarization. Otherwise set $x(a^*) = 1$ and remove a^* from L.



(a) Query (scene)



(b) Model (object in DB)



(c) Affinity matrix

Figure 2: [Best viewed in color.] Example feature sets and their affinity matrix. (a-b): both the query and the model have four points (features) whose pairwise distances are annotated on edges between them. (c): M is a symmetric matrix, with each of its row or column (i, i') representing a pair derived from the two feature sets. Note that submatrices with the red bounding rectangles contain almost the same values as the submatrices with the blue bounding rectangles, reflecting the main idea of Lemma 1.

2. Remove from L all potential correspondences which conflict with $a^* = (i, i')$. These have the form of $(i, *)$ or $(*, i')$.
3. If L is empty, return the solution x. Otherwise go back to step 1.

There is a scalability issue, however, in computing the eigenvector of M. Notice that M is a matrix with $n_P n_Q$ rows and columns, which means that the number of nonzero elements in M can grow in proportion to n_Q^4 at maximum (see Section 3.3 for more details), assuming $n_P \sim n_Q$, which is typically the case in real world data as described in Section 4. Thus, constructing M explicitly in memory is prohibitive. In the next section, we show how to solve this problem.

3. Proposed Method

In this section, we propose FASM (FAst Spectral Matching), a fast, scalable, and accurate approximation method for spectral matching. FASM comprises two steps.

1. **Affinity Matrix Approximation.** Compute the approximated affinity matrix $\hat{M}$.
2. **Eigenvector Computation.** Compute the principal eigenvector of $\hat{M}$.

We first describe our proposed method of approximating the affinity matrix used for the spectral matching, and the fast method for computing the eigenvector of the affinity matrix.

3.1. Affinity Matrix Approximation

As described in Section 2, explicitly materializing M and computing the first eigenvector of M are hopeless due to the heavy storage requirement of M. Our proposed idea to solve this problem is to exploit the redundancy pattern in the affinity matrix. We observe that some areas of the matrix M have almost the same elements which we formally describe as follows.

Lemma 1 (Redundancy in Affinity Matrix)**.** *Let i_1, i_2, j_1, j_2 be the points in P, with the pairwise distances following the condition $d_{i_1 j_1} = d_{i_2 j_2}$. The four points are distinct, with the exception that i_1 and i_2 can be the same. Let i' and j' be the arbitrary points in Q, satisfying $i' \neq j'$. Let $a_k = (i_k, i')$ and $b_k = (j_k, j')$, for $k = 1, 2$. Then, the submatrix of M involving i_1 and j_1 is identical to the submatrix of M involving i_2 and j_2. That is, $M(a_1, b_1) = M(a_2, b_2)$.*

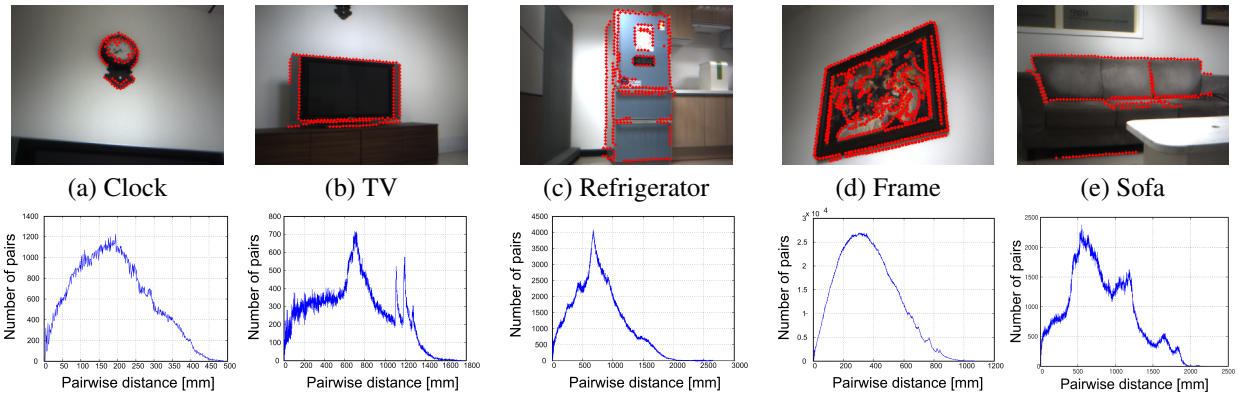
Proof: Let $|d_{i_1 j_1} - d_{i' j'}| < 3\sigma_d$. Then,

$$\begin{aligned} M(a_1, b_1) &= 4.5 - \frac{(d_{i_1 j_1} - d_{i' j'})^2}{2{\sigma_d}^2} \\ &= 4.5 - \frac{(d_{i_2 j_2} - d_{i' j'})^2}{2{\sigma_d}^2} \\ &= M(a_2, b_2) \end{aligned}$$

where we used the assumption $d_{i_1 j_1} = d_{i_2 j_2}$ in the second equality. The same analysis applies to the case when $|d_{i_1 j_1} - d_{i' j'}| \geq 3\sigma_d$. □

Lemma 1 says that if there are two pairs of points in P whose pairwise distances are the same, then the submatrices of M involving the two pairs are exactly the same. Furthermore, if $|d_{i_1 j_1} - d_{i_2 j_2}|$ is very small, Lemma 1 implies that $M(a_1, b_1) \sim M(a_2, b_2)$. Thus, if we have many pairs of points whose pairwise distances are the same or close, then we have many redundancies in the matrix M. For example, the distance between the points 1 and 2 is almost the same as that between the points 1 and 4 in Figs. 2a and 2b, which resulted in the submatrices with the red bounding rectangles containing almost the same values as the submatrices with the blue bounding rectangles in Fig. 2c.

To make the observation into something useful, it is necessary to check whether there are many pairs of points with same pairwise distances in real world data. For the purpose, we analyze the distance distribution from real world data in Fig. 3. We see that the distribution is highly skewed, and many pairs of points have same or similar pairwise distances.



(a) Clock (b) TV (c) Refrigerator (d) Frame (e) Sofa

(f) Pairwise distance distribution of clock (g) Pairwise distance distribution of TV (h) Pairwise distance distribution of refrigerator (i) Pairwise distance distribution of frame (j) Pairwise distance distribution of sofa

Figure 3: [Best viewed in color.] Objects and their distance distributions. (a-e): red dots indicate feature points sampled from edges extracted with an edge detector [6]. (f-j): x-axis: the pairwise distance, y-axis: the number of pairs with such distance. We use 1mm for the bin width.

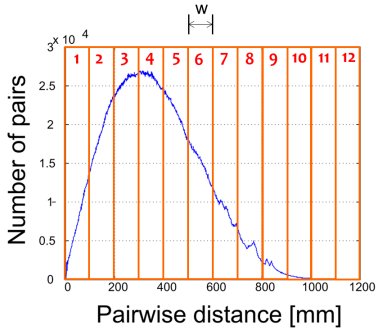


Figure 4: Distance distribution of the frame object shown in Fig. 3d, divided by bins whose width w is 100. All the distances within a bin are approximated to the center distance of the bin: for example, all the distances from 200 to 300 mm (the rectangle 3) are approximated to 250 mm. In real applications, the bin width is much smaller than 100 (typically, 5 mm).

Given this observation, our proposed idea is to approximate the elements of M using approximate distances $\hat{d}_{ij}$ instead of exact distances d_{ij}. Let d_{max} be the maximum distance of the points in P. We divide the total distance range into $k = \frac{d_{max}}{w}$ bins, where w is the width of a bin. Then, all the distances within the range from $w \cdot i$ to $w(i+1)$ is approximated by $\hat{d}_{ij} = \frac{w(2i+1)}{2}$. That is, d_{ij} is approximated by

$$\hat{d}_{ij} = w(\lfloor \frac{d_{ij}}{w} \rfloor + \frac{1}{2}), \tag{4}$$

where the function $\lfloor x \rfloor$ maps a real number x to the largest integer not greater than x.

For example, see Fig. 4 for the distance distribution of the frame object shown in Fig. 3d. We divide the total

distance range into 12 bins of width 100. All the distances within a bin are approximated to the center distance of the bin: e.g., all the distances from 200 to 300 mm (the rectangle 3) are approximated to 250 mm.

How does this approximation of d_{ij} by $\hat{d}_{ij}$ change the affinity matrix M? Let M_{ij} be the n_Q by n_Q submatrix of M, containing the rows $(i-1)n_Q+1 : i \cdot n_Q$ and the columns $(j-1)n_Q+1 : j \cdot n_Q$. If we think of filling M with n_Q by n_Q blocks, then M_{ij} is the area of M covered with the block with the block row id i and the block column id j. That is, M can be represented by

$$M = \begin{bmatrix} 0 & M_{12} & \cdots & M_{1n_P} \\ M_{21} & 0 & \cdots & M_{2n_P} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n_P1} & M_{n_P2} & \cdots & 0 \end{bmatrix}.$$

Recall from (1) that for two points $i, j \in P$ whose pairwise distance is d_{ij}, the (i', j')th element $M_{ij}(i', j')$ of M_{ij} is determined by

$$M_{ij}(i', j') = \begin{cases} 4.5 - \frac{(d_{ij} - d_{i'j'})^2}{2{\sigma_d}^2}, & \text{if } |d_{ij} - d_{i'j'}| < 3\sigma_d; \\ 0 & \text{otherwise.} \end{cases}$$

Our idea is to approximate M_{ij} with $\hat{M}_{ij}$ whose (i', j')th element $\hat{M}_{ij}(i', j')$ is determined by

$$\hat{M}_{ij}(i', j') = \begin{cases} 4.5 - \frac{(\hat{d}_{ij} - d_{i'j'})^2}{2{\sigma_d}^2}, & \text{if } |\hat{d}_{ij} - d_{i'j'}| < 3\sigma_d; \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Since there are $(k = \frac{d_{max}}{w})$ distinct $\hat{d}_{ij}$s, the approximation by (5) creates k distinct $n_Q \times n_Q$ submatrices of $\hat{M}$. Let $B = \{B_0, ..., B_{k-1}\}$ be the set of such matrices, where B_i is created using the approximated distance $w(i + \frac{1}{2})$. Let H_i be the $n_P \times n_P$ index matrix containing only the index of the matrix B_i in $\hat{M}$. That is, H_i's (j, k)th element $H_i(j, k)$ is 1 if $\hat{M}_{jk} = B_i$, and 0 otherwise. Then, $\hat{M}$ can be represented by the linear combination of the Kronecker product of H_i and B_i, for $0 \leq i \leq k-1$:

$$\hat{M} = \sum_{i=0}^{k-1} H_i \otimes B_i, \tag{6}$$

where the Kronecker product of two matrices H_i and B_i of dimensions $n_P \times n_P$ and $n_Q \times n_Q$, respectively, is a matrix with dimensions $n_P n_Q \times n_P n_Q$ given by

$$H_i \otimes B_i = \begin{bmatrix} H_i(1,1)B_i & H_i(1,2)B_i & \cdots & H_i(1,n_P)B_i \\ H_i(2,1)B_i & H_i(2,2)B_i & \cdots & H_i(2,n_P)B_i \\ \vdots & \vdots & \ddots & \vdots \\ H_i(n_P,1)B_i & H_i(n_P,2)B_i & \cdots & H_i(n_P,n_P)B_i \end{bmatrix}.$$

For example, when we use the bin width 10 to approximate the matrix in Fig. 2, $\hat{M}$ becomes

$$\hat{M} = \begin{bmatrix} 0 & B_4 & B_5 & B_4 \\ B_4 & 0 & B_2 & B_4 \\ B_5 & B_2 & 0 & B_2 \\ B_4 & B_4 & B_2 & 0 \end{bmatrix}$$

with H_2, H_4, and H_5 given by

$$H_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, H_4 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}, and\ H_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Note that H_i and B_i matrices are highly compressed representation of the $\hat{M}$ matrix. Storing H_i and B_i require much smaller space than explicitly storing the $\hat{M}$ matrix, as we will see in Section 3.3.

A remaining question is, how accurate is the approximation of $M_{ij}(i',j')$ by $\hat{M}_{ij}(i',j')$? Since d_{ij} and $\hat{d}_{ij}$ are close, the matrix elements $M_{ij}(i',j')$ and $\hat{M}_{ij}(i',j')$ are expected to be close. We provide the bound of the approximation as follows.

Lemma 2 (Bound of Approximation)**.** *Assume* $|d_{ij} - d_{i'j'}| < 3\sigma_d - \frac{w}{2}$. *Then,* $|\hat{M}_{ij}(i',j') - M_{ij}(i',j')| \leq \frac{1.5w}{\sigma_d}$.

Proof: $|\hat{M}_{ij}(i',j') - M_{ij}(i',j')|$
$= |(4.5 - \frac{(d_{ij}-d_{i'j'})^2}{2{\sigma_d}^2}) - (4.5 - \frac{(\hat{d}_{ij}-d_{i'j'})^2}{2{\sigma_d}^2})|$
$= |\frac{(d_{ij}+\hat{d}_{ij}-2d_{i'j'})(d_{ij}-\hat{d}_{ij})}{2{\sigma_d}^2}| \leq \frac{6\sigma_d \frac{w}{2}}{2{\sigma_d}^2} = \frac{1.5w}{\sigma_d}$,
where the inequality is derived from the fact that $|d_{ij} - \hat{d}_{ij}| \leq \frac{w}{2}$ *which follows from the definition of* $\hat{d}_{ij}$ *in* (4). □

Lemma 2 states that the approximated matrix does not differ much from the original matrix, under the mild assumption. We verify the accuracy of the approximation with the experiments in Section 4.

3.2. Eigenvector Computation

The standard method to compute the principal eigenvector of a matrix is called the power method [13]. In the power method, we multiply the matrix with a randomly initialized vector repeatedly until the normalized vector converges. The converged vector is the principal eigenvector of the matrix.

A naive method to compute the principal eigenvector of $\hat{M}$ is to explicitly materialize $\hat{M}$ and run a standard power method to compute the principal eigenvector. However, such method lacks scalability since the materialized matrix $\hat{M}$ is very large. Our proposed idea, called bases power method, is to use the compressed representation of $\hat{M}$ for the power method. Recall that we expressed $\hat{M}$ using the linear combinations of Kronecker Products of H_i and B_i matrices in (6). The key observation is that we can perform the power method on $\hat{M}$ using H_i and B_i. Applying the power method on $\hat{M}$ requires the matrix-vector multiplication $\hat{M}v$ for a vector v. Since $\hat{M} = \sum_{i=0}^{k-1} H_i \otimes B_i$, we can instead compute

$$\hat{M}v = (\sum_{i=0}^{k-1} H_i \otimes B_i)v = \sum_{i=0}^{k-1}(H_i \otimes B_i)v$$

for computing $\hat{M}v$.

Algorithm 1 describes our proposed bases power method. In line 2, a random vector with norm 1 is initialized. Then, the vector is multiplied with the $\hat{M}$ repeatedly until convergence, in lines 4 to 11. Note that neither $\hat{M}$ nor $H_i \otimes B_i$, for $i = 0..k-1$, are never explicitly constructed: only the bases matrices B_i and index matrices H_i are used to compute the matrix-vector multiplication. To compute $(H_i \otimes B_i)v$, for each nonzero elements $H_i(x, y)$ we

Algorithm 1: Bases Power Method

Input : Bases matrices B_i and index matrices H_i, for $0 \leq i \leq k-1$
Output: Principal eigenvector $\hat{x}$ of $(\hat{M} = \sum_{i=0}^{k-1} H_i \otimes B_i)$

1 **begin**
2 $v^{(0)} \leftarrow$ random vector with $||v^{(0)}|| = 1$;
3 $j \leftarrow 0$;
4 **while** $v^{(j)}$ *does not converge* **do**
5 $j \leftarrow j + 1$;
6 $z \leftarrow 0$;
7 **for** *i=0..k-1* **do**
8 $z \leftarrow z + (H_i \otimes B_i)v^{(j-1)}$;
9 **end**
10 $v^{(j)} \leftarrow z/||z||$;
11 **end**
12 $\hat{x} \leftarrow v^{(j)}$;
13 **end**

multiply B_i with $v((y-1)n_Q+1:y\cdot n_Q)$ and add the resulting vector to the $v((x-1)n_Q+1:x\cdot n_Q)$th rows of the output vector. This method avoids constructing $H_i \otimes B_i$ and thus requires much less storage than the original spectral matching (see Section 3.3 for a detailed analysis). Note that one can even decrease the number of floating point operations by reusing the multiplication result among the nonzero elements of H_i with the same column id, since the multiplication results are the same for $H_i(x1, y1)$ and $H_i(x2, y2)$ if $y1 = y2$. Also note that the B_i and H_i matrices are accessed sequentially (not randomly) in line 7-9 of Algorithm 1, making the algorithm suitable for large scale, distributed platform like MAPREDUCE [9] which allows only sequential accesses to files.

3.3. Discussion

We discuss the storage, time of FASM, and the perturbation analysis of the principal eigenvector of the affinity matrix.

Storage Estimation. In this section, we estimate the required memory in the standard spectral matching and in our approximation. From the definition of M in 1, the matrix M, as well as its submatrices M_{ij}, are symmetric. Thus, the maximum number of nonzero elements inside a block is given by $\sum_{i=1}^{n_Q-1} i = \frac{n_Q(n_Q-1)}{2}$, if we save only upper-diagonal elements. Applying the same analysis to M, the maximum number of nonzero elements in the matrix M to store in memory is given by

$$\text{Maximum nonzeros in } M = \frac{n_P(n_P-1)}{2}\frac{n_Q(n_Q-1)}{2}.$$

Let ρ be the density of the matrix M, meaning the number of edges divided by the maximum possible number of edges. Assuming $n_P \sim n_Q$, it follows that

$$\text{Nonzeros in } M = \rho\frac{n_P(n_P-1)}{2}\frac{n_Q(n_Q-1)}{2} \propto O(n_Q^4).$$

In contrast, storing $\hat{M}$ requires ($k = \frac{d_{max}}{w}$) distinct bases matrices B_i with dimensions $n_Q \times n_Q$, and index matrices H_i with dimensions $n_P \times n_P$, for $0 \le i \le k-1$. Notice that the number of nonzero elements in all the index matrices, in total, is at most n_P^2. Thus, the maximum number of nonzero elements to store $\hat{M}$ is given by

$$\text{Maximum nonzeros to store } \hat{M} = \frac{d_{max}}{w}\frac{n_Q(n_Q-1)}{2} + n_P^2.$$

Assuming $\hat{M}$ has the same density as M, and $n_P \sim n_Q$,

$$\text{Nonzeros to store } \hat{M} \le \rho\frac{d_{max}}{w}\frac{n_Q(n_Q-1)}{2} + n_P^2 \propto O(n_Q^2).$$

Thus, the quadratic growth rate of our approximation is much smaller than the quartic growth rate of the standard spectral matching. We verify this growth rate in Section 4.3.

Time Analysis. FASM runs faster than the standard spectral matching for two reasons. First, FASM requires much less memory than the standard spectral matching, thereby benefits much more from the cache functionality of modern computer system. Especially, computing $(H_i \otimes B_i)v$ for each i (line 8 of Algorithm 1) requires the repetitive use of B_i some (or all) of which can be contained in the cache. Second, FASM allows an optimization of reducing the number of floating point operations by reusing multiplication results, as described in the last paragraph of Section 3.2. We will see that these two advantages lead to the running time differences up to $33\times$, in Section 4.

Perturbation Analysis. An interesting question is how the principal eigenvector $\hat{x}$ of the approximated matrix $\hat{M}$ differs from the principal eigenvector x of the original matrix M. Let the matrix $\Delta = \hat{M} - M$ be the difference of the two matrices $\hat{M}$ and M, and γ_1 be the eigengap (difference of the first and the second largest eigenvalues) of M. From the standard matrix perturbation theory [22, 30], the difference of the eigenvectors is bounded by

$$||\hat{x} - x||_2 \leq \frac{4||\Delta||_F}{\gamma_1 - \sqrt{2}||\Delta||_F}. \tag{7}$$

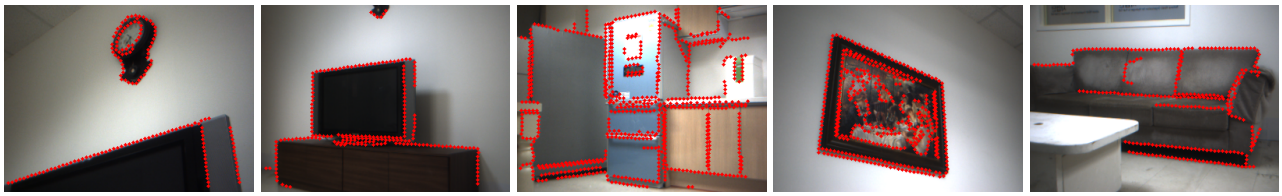
where the eigenvector changes little if the eigengap is large and the perturbation is small. As we will see in Section 4, the left and the right hand term of (7) is small for real data.

4. Experiments

We present experimental results to answer the following questions:

- **Q1 Accuracy.** What is the accuracy of our FASM compared to the exact algorithm?
- **Q2 Memory and Running Time.** How much does our FASM reduce the required memory and the running time?
- **Q3 Scalability.** How does our FASM scale-up with data sizes?
- **Q4 Parameter.** What is the trade-off between the accuracy and the running time/memory size on different values of the bin width parameter in FASM?
- **Q5 Perturbation.** How does the principal eigenvector change due to the approximation by FASM?
- **Q6 Practical Implementation.** How can we make our FASM faster and reduce the required memory even more?

For the experiments, we use discrete sets of 3D points of image features sampled from the external and internal contours on objects in Figs. 3 and 5, whose sizes are listed in Table 2. These data are available in `http://cs.cmu.edu/~ukang/FASM`. In order to extract the image features on the objects, we first obtain the edge map from



(a) Clock (b) TV (c) Refrigerator (d) Frame (e) Sofa

Figure 5: [Best viewed in color.] Query images including clock, TV, refrigerator, frame, and sofa. Red dots indicate feature points sampled from edges extracted with an edge detector [6].

Object	Features in Model (Figure 3)	Features in Query (Figure 5)	Number of matrix nonzeros in SM	Number of matrix nonzeros in FASM
Clock	79	155	1,928,048	30,794
TV	167	204	11,702,742	138,602
Refrigerator	313	701	315,033,384	488,248
Frame	419	567	1,153,604,736	2,060,794
Sofa	1390	1567	2,094,849,871	4,825,635

Table 2: Size of the data and its affinity matrix.

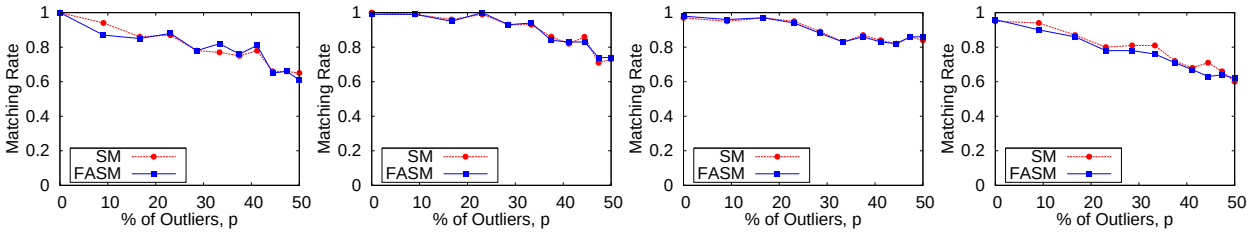
the Canny edge detector [6]. Next, we remove spurious edges by grouping the edge elements into connected contour fragments [28]. Then we select image features by evenly sampling the edge contours. We use $\sigma_d = 5$ for all the experiments, and bin width $w = 5$ for the experiments from Section 4.1 to 4.3 and 4.6, and vary the bin width for the experiments from Section 4.4 to 4.5. We use SM to refer to the standard exact spectral matching algorithm. All experiments are performed in a machine with Intel Core i5 CPU 661 (3.33Ghz) and 8GB DDR3 memory.

4.1. Accuracy

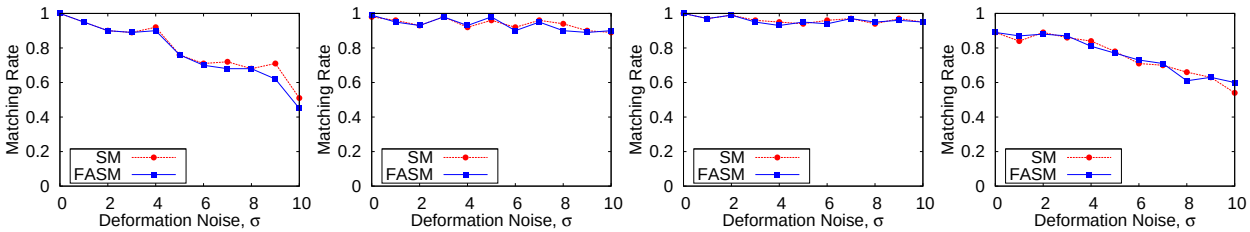
How accurate is our approximated FASM algorithm, compared to the exact SM algorithm? For the evaluation, we run FASM and SM on the model objects in Table 2, and the perturbed version of the objects to measure the matching rate defined as the ratio of the correctly matched pairs and the total matched pairs. We use the following perturbations:

- *Varying outliers.* We add Gaussian noise $N(0, 5^2)$ to the model points, and rotate/translate them randomly. Then, we choose $p\%$ of the points randomly, where we vary p, and swap their x, y, z coordinates to make them outliers.
- *Varying noise.* We choose 20% of the model points randomly, and swap their x, y, z coordinates to make them outliers. Then, we add Gaussian noise $N(0, \sigma^2)$, where we vary σ, and rotate/translate them randomly.

In Figs. 6a $\sim$ 6d, we see that FASM has little or no loss of accuracy compared to SM on varying the percentage p of outliers. Varying the deformation Gaussian noise σ leads to the same conclusion as shown in Figs. 6e $\sim$ 6h: FASM



(a) Accuracy w.r.t. outliers in clock (b) Accuracy w.r.t. outliers in TV (c) Accuracy w.r.t. outliers in refrigerator (d) Accuracy w.r.t. outliers in frame



(e) Accuracy w.r.t. noises in clock (f) Accuracy w.r.t. noises in TV (g) Accuracy w.r.t. noises in refrigerator (h) Accuracy w.r.t. noises in frame

Figure 6: [Best viewed in color.] Accuracy comparison between SM (standard spectral matching) and FASM (our proposed method). FASM and SM are compared on the model objects in Table 2, and the perturbed version of the objects to measure the matching rate defined as the ratio of the correctly matched pairs and the total matched pairs. FASM has little or no loss of accuracy compared to SM.

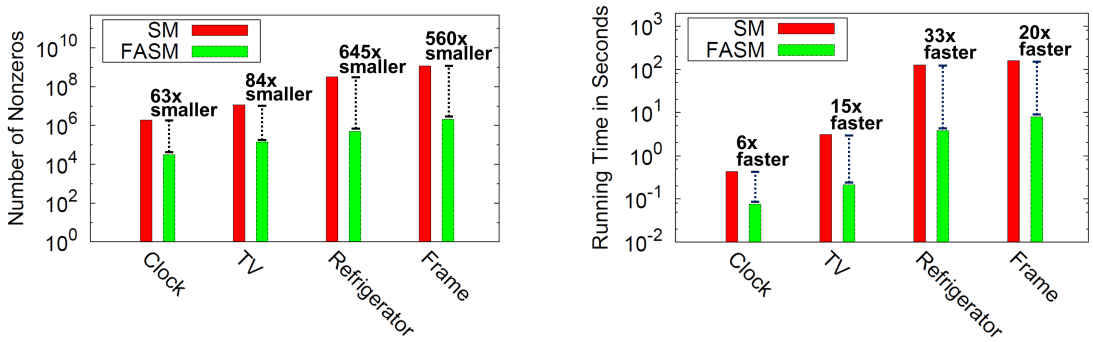
provides little or no loss of accuracy compared to SM.

4.2. Memory and Running Time

How much does our FASM save the required memory and the running time, compared to SM? We compare the required memory and the running time of FASM and SM using the objects in Table 2. As shown in Fig. 7a, FASM requires 63× to 645× smaller memory than SM. Furthermore, FASM is 6× to 33× faster than SM, as shown in Fig. 7b.

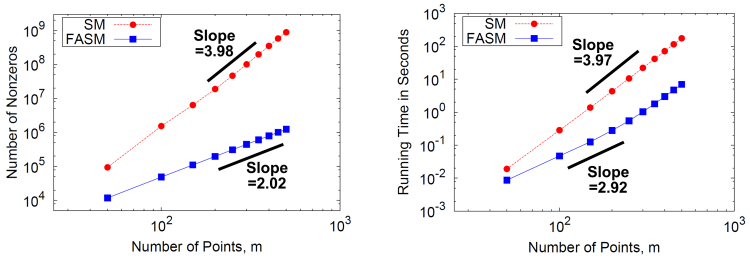
4.3. Scalability

How does our FASM scale-up with growing data sizes? To compare the scalability of FASM and SM, we measure the running time and memory size as the data sizes grow. Specifically, we randomly take m points from the frame object with 567 points, to make a model object. Then, we add Gaussian noise $N(0, 5^2)$ to the m points, and swap the x, y, z coordinates of the 20% of the m points to make the query object. As we see in Fig. 8a, the required memory in FASM grows much slowly than in SM. The growth rates 2.02 and 3.98 of the memory sizes of FASM and SM, respectively, are very close to our estimated quadratic and quartic growth rate in Section 3.3.



(a) Memory usage comparison of FASM and SM. (b) Running time comparison of FASM and SM.

Figure 7: Memory usage and running time comparison between SM and FASM. The y-axis is in log scale. (a): FASM requires 63× to 645× smaller memory than SM. (b): FASM is 6× to 33× faster than SM.



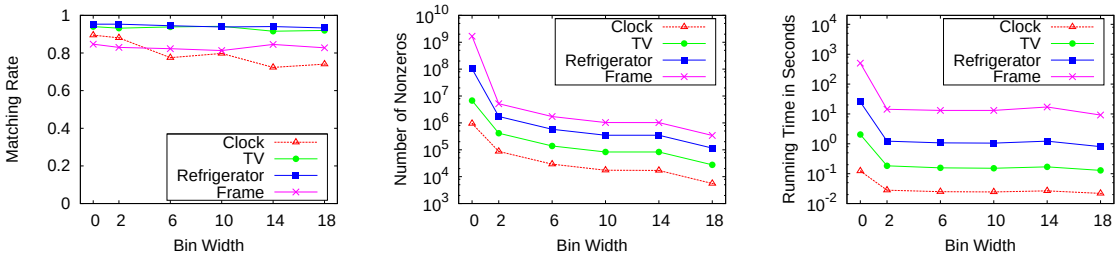
(a) Memory vs. data size (b) Running time vs. data size

Figure 8: Scalability comparison of SM and FASM, on the frame object shown in Fig. 5d. The growth rates 2.02 and 3.98 of the memory sizes of FASM and SM, respectively, are very close to our estimated quadratic and quartic growth rate in Section 3.3. The difference of the growth rate of the running times is smaller than that of the memory sizes, since the bases power method in FASM still requires similar number of floating point operations, compared to the power method in SM.

The result in Fig 8b shows that the running time of FASM grows slower than SM. However, the difference of the growth rate of the running times is smaller than that of the memory sizes, since the bases power method in FASM still requires similar number of floating point operations, compared to the power method in SM.

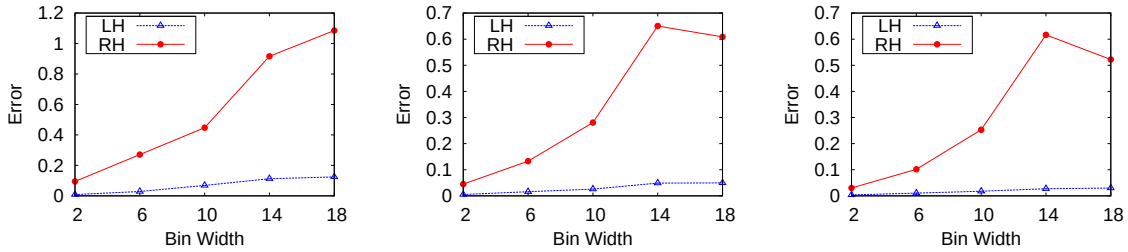
4.4. Parameter

How does the bin width parameter affect the performance of FASM? For the evaluation, we run FASM on the model objects in Table 2 and the perturbed version of the objects to measure the matching rate, the required memory, and the running time. We perturb the model points with Gaussian noise $N(0, 5^2)$, and add outliers by swapping the x, y, z coordinates of the 20% of the model points. Fig. 9 shows the trade-off results with regard to the bin width parameter on the objects shown in Fig. 3. The result on the bin width 0 is from the standard spectral matching algorithm SM, and all other results are from the FASM. We see that increasing the bin width slightly decrease the



(a) Accuracy (b) Memory size (c) Running time

Figure 9: The effect of the bin width parameter on the performance of FASM tested on the objects shown in Fig. 3. The result on the bin width 0 is from the standard spectral matching algorithm SM, and all other results are from the FASM. Increasing the bin width slightly decrease the accuracy, while decreasing the memory requirement dramatically. The running time also decreases as the bin width increase, but not as quickly as the rate of the memory size decrease since the number of floating point operations didn't change much.



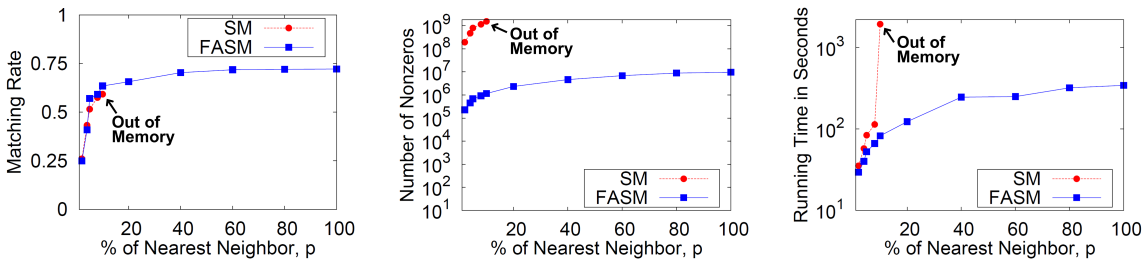
(a) 40 points (b) 100 points (c) 167 points

Figure 10: Perturbation analysis on the TV object shown in Fig. 3b. (c) is the full dataset, while (a) and (b) are randomly sampled dataset from the full dataset. LH=$||\hat{x} - x||_2$, and RH=$\frac{4||\Delta||_F}{\gamma_1 - \sqrt{2}||\Delta||_F}$ of (7). Note that the actual difference (labeled 'LH') of the eigenvector is very small (at most 0.1241 for the largest bin width 18). Also note that the actual difference is much smaller than the theoretical upper bound (labeled 'RH'), which explains the high accuracy of the experiments shown in Section 4.1.

accuracy, while decreasing the memory requirement dramatically. The running time also decreases as the bin width increase, but not as quickly as the rate of the memory size decrease since the number of floating point operations does not change much. Fig. 9 suggests how to choose the bin width parameter. Increasing the bin width has advantages for decreasing the memory and running time, but it has the disadvantage of decreasing the accuracy. Thus, the bin width parameter should be determined based on the desired accuracy, memory size, and the running time.

4.5. Perturbation

How does the principal eigenvector change due to the approximation by FASM? We obtain the ideal matrix M and the approximated matrix $\hat{M}$ experimentally, in matching sets of feature points from the TV object shown in Fig. 3b. We rotate and translate a set of feature points, without adding Gaussian noise and outliers, to obtain the other set.



(a) Accuracy (b) Memory Size (c) Running Time

Figure 11: Performance analysis on the effect of the number of nearest neighbors for building the pairwise relationships per feature point, on the sofa object. The accuracy of FASM quickly increases up to 10%, while slightly increases from 10% to 100%. The memory size and running time of FASM linearly increase as the number of nearest neighbors increase. SM failed to show the matching results when the number of nearest neighbors is larger than 10% since the required memory to store the affinity matrix of SM exceeds the physical memory size of our machine.

We then compute the difference of the eigenvector ($||\hat{x} - x||_2$), and the upper bound of the difference $\frac{4||\Delta||_F}{\gamma_1 - \sqrt{2}||\Delta||_F}$ of Equation (7). Fig. 10 shows that the actual difference of the eigenvector (labeled 'LH') is very small (at most 0.1241 for the largest bin width 18). We also see that the actual difference is much smaller than the theoretical upper bound (labeled 'RH'), which explains the high accuracy of FASM in practice as shown in Fig 6.

4.6. Practical Implementation

How can we make our FASM method faster and reduce the required memory even more? When the number of feature points is very large, it is very time consuming to use all the pairwise relationships of feature points. A simple and practical solution for this problem is to build the pairwise relationships per feature point with its *nearest neighbors*. To do that we first store the model and query feature points in a kd-tree [2]. Then we only find and sample the n nearest neighbors per feature point in both model and query objects, and compute their pairwise distances.

How does the number of nearest neighbors affect the performance of our FASM method, compared to the SM method? For the evaluation, we run FASM and SM on the model object of sofa in Table 2 and the perturbed version of the object to measure the matching rate, the required memory, and the running time. We disturbed the model points with Gaussian noise $N(0, 5^2)$, and add outliers by swapping the x, y, z coordinates of the 20% of the model points. Fig. 11 shows the evaluation results with respect to the number of nearest neighbors on the sofa object shown in Fig. 3e. We determine the number of nearest neighbors as $p\%$ of the model points. We see that increasing the number of nearest neighbors increase the accuracy, memory, and running time. As we see in Fig. 11a, the accuracy of FASM quickly increases until p is 10%, while slightly increases when p is from 10% to 100%. The memory and running time of FASM linearly increase as shown in Figs. 11b and 11c. We also see that SM failed to show the matching results when the number of nearest neighbors is larger than 10% since the required memory to store the affinity matrix of SM

exceeds the physical memory size of our machine. The above experimental results show that SM cannot establish and complete the matching process for huge data, while our FASM method is able to complete its matching process even if the number of feature points is very large.

5. Related Work

In this section, we review related works on graph matching-based correspondence problems.

Correspondence problems have been formulated by graph matching problems: each feature set is considered as a clique graph whose edges represent the pairwise relationship (e.g. pairwise distance) of two features.

The goal of graph matching is to find correspondences between the nodes of two feature graphs such that both the unary information on the nodes and the pairwise information associated with the edges are preserved as much as possible. Graph matching can be formulated as the optimization problem of a certain cost function that includes both the node-to-node (or unary, first-order) [16, 21] as well as the edge-to-edge (or pairwise, second-order) [3, 8, 19, 20] similarities. Most recently, the hyper-graph matching problem has also been studied [10, 18, 31], by considering higher-order relationships between tuples of features. The Integer Quadratic Programming (IQP) problem formulated as (2) and (3) is a recent and generalized definition of graph matching, which can be applied to almost all types of graph matching problems. Since the IQP problem is NP-hard [11], various relaxation methods have been studied to solve the problem. Gold and Rangarajan [12] presented one of the first of second-order matching methods solving the IQP problem by the spectral relaxation with power iteration, in which Sinkhorn's bistochastic normalization [29] is employed to satisfy two-way assignment constraints for one-to-one mapping. Maciel and Costeira [21] proposed a global optimization solution for an IQP problem by maximizing all possible correlation computed with unary information, in which a concave objective function is constructed and the discrete search domain is relaxed into its convex hull. Berg et al. [3] presented fast-approximate techniques to address NP-hard problem of the IQP whose cost function embeds the similarity of both unary and pairwise information, in which the minimum of a linear bounding problem is first found and then a locally minimal assignment is detected with a manner of local gradient descent. Leordeanu and Hebert [19] used similar cost function to [3] while they approximated the IQP problem as an eigenvector problem, such as described in Section 2. Cour et al. [8] presented similar spectral matching algorithm to [19], in which they used the modified cost function $x^T Mx/x^T x$ and imposed affine constraints on the relaxed solution. Cho et al. [7] proposed a random walk view for graph matching, in which they incorporated one-to-one mapping constraints by a reweighting jump scheme. Zass and Shashua [31] proposed the hyper-graph matching algorithm, in which they marginalize the affinity tensor into a one-dimensional vector and refines the vector by projecting it onto the space of soft assignment vectors. Duchenne et al. [10] extended the spectral matching method [19] to higher-order

relationships by using a multi-dimensional power method for tensors, in which the cost function is formulated by a tensor encoding the similarity between feature tuples. Lee et al. [18] generalized their second-order method [7] using the random walk approach to higher-order graph. These matching methods solve the IQP problem with an affinity matrix [3, 7, 8, 12, 19] or an affinity tensor [10, 18, 31] as an input, but the time complexity of matching algorithms and the storage size of each affinity depend on the number of feature points. Therefore, it could happen that a complete computation of the affinity is impracticable or unwarrantable when the matching methods find correspondences between large feature sets. In this paper, we solve this problem with the use of bases and index matrices as well as the bases power method.

6. Conclusions

In this paper, we propose FASM (FAst Spectral Matching), an approximate spectral matching algorithm for correspondence problems with large feature sets whose affinity matrix contains billions of nonzero elements. The main contributions are the followings.

1. **Approximation.** Exploiting the high redundancy in the affinity matrix used for the spectral matching, we propose an accurate approximation of the matrix by the linear combination of Kronecker products between 'bases' and index matrices which require much smaller memory than in previous works.
2. **Fast Algorithm.** Using the bases and index matrices, we develop an efficient algorithm to compute the principal eigenvector of the approximated matrix.
3. **Experiments.** Extensive experiments show that our FASM is up to $33\times$ faster, requires up to $645\times$ smaller memory than the exact algorithm, with little or no loss of accuracy.

Future research directions include extending our method to higher-order graph matching which utilizes tensors for encoding the affinity between feature tuples.

Acknowledgement

This paper was performed for the Intelligent Robotics Development Program, one of the 21st Century Frontier R&D Programs funded by the Ministry of Knowledge Economy of Korea.

References

[1] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, 2002.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] A. C. Berg, T. L. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondences. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 26–33, 2005.

[4] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[5] M. Brown and D. G. Lowe. Recognising panoramas. In *Internat. Conf. on Computer Vision (ICCV)*, pages 1218–1227, 2003.

[6] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[7] M. Cho, J. Lee, and K. M. Lee. Reweighted random walks for graph matching. In *European Conference on Computer Vision (ECCV)*, pages 492–505, 2010.

[8] T. Cour, P. Srinivasan, and J. Shi. Balanced graph matching. In *Conf. Neural Information Processing Systems (NIPS)*, pages 313–320, 2006.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Symposium on Operating Systems Design and Implementation (OSDI)*, pages 137–150, 2004.

[10] O. Duchenne, F. R. Bach, I.-S. Kweon, and J. Ponce. A tensor-based algorithm for high-order graph matching. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 1980–1987, 2009.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[12] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

[13] G. H. Golub and C. F. V. Loan. Matrix computations. *Johns Hopkins University Press*, 1996.

[14] R. A. Horn and C. R. Johnson. Matrix analysis. *Cambridge University Press*, 1985.

[15] G. Kim, M. Hebert, and S.-K. Park. Preliminary development of a line feature-based object recognition system for textureless indoor objects. In S. Lee, I. Suh, and M. Kim, editors, *Recent Progress in Robotics: Viable Robotic Service to Human*, volume 370 of *Lecture Notes in Control and Information Sciences*, pages 255–268. Springer Berlin / Heidelberg, 2008.

[16] H. W. Kuhn. The hungarian method for the assignment problem. In *Naval Research Logistics Quarterly*, volume 2, pages 83–97, 1955.

[17] K. Lange. Numerical analysis for statisticians. *Springer*, 1999.

[18] J. Lee, M. Cho, and K. M. Lee. Hyper-graph matching via reweighted random walks. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 1633–1640, 2011.

[19] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *Internat. Conf. on Computer Vision (ICCV)*, pages 1482–1489, 2005.

[20] M. Leordeanu, M. Hebert, and R. Sukthankar. Beyond local appearance: Category recognition from pairwise interactions of simple features. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2007.

[21] J. Maciel and J. Costeira. A global solution to sparse correspondence problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(2):187–199, 2003.

[22] A. Y. Ng, A. X. Zheng, and M. I. Jordan. Link analysis, eigenvectors and stability. In *Internat. Joint Conf. Artificial Intelligence (IJCAI)*, pages 903–910, 2001.

[23] S. Park, S. Kim, M. Park, and S.-K. Park. Vision-based global localization for mobile robots with hybrid maps of objects and spatial layouts. *Information Sciences*, 179:4174–4198, 2009.

[24] S. Park and S.-K. Park. Spectral scan matching and its application to global localization for mobile robots. In *IEEE Conf. Robotics and Automation (ICRA)*, pages 1361–1366, 2010.

[25] P. Pritchett and A. Zisserman. Wide baseline stereo matching. In *Internat. Conf. on Computer Vision (ICCV)*, pages 754–760, 1998.

[26] S. Se, D. G. Lowe, and J. J. Little. Vision-based global localization and mapping for mobile robots. *IEEE Transactions on Robotics*, 21(3):364–375, 2005.

[27] J. Shi and C. Tomasi. Good features to track. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

[28] K. K. T. K. X. Shyjan Mahamud, Lance R. Williams. Segmentation of multiple salient closed contours from real images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(4):433–444, 2003.

[29] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

[30] G. Stewart and J.-G. Sun. Matrix perturbation theory. *Academic Press*, 1990.

[31] R. Zass and A. Shashua. Probabilistic graph and hypergraph matching. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2008.

[32] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994.