2017

# A New Computational Framework for Efficient Parallelization and Optimization of Large Scale Graph Matching

Sahar Marefat Navaz
*Louisiana State University and Agricultural and Mechanical College*

# A NEW COMPUTATIONAL FRAMEWORK FOR EFFICIENT PARALLELIZATION AND OPTIMIZATION OF LARGE SCALE GRAPH MATCHING

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The School of Electrical Engineering and Computer Science

by
Sahar Marefat Navaz
M.Sc., Newcastle University, 2008
BEng., Northumbria University, 2006
May 2017

Dedicated to my parents.

# ACKNOWLEDGMENT

I would like to thank the following people whose continued support and guidance through all these years inspired me to complete my PhD program.

I would like to gratefully thank my adviser Prof. Ram for his support, patience, and most importantly giving me the opportunity and freedom to pursue my interests and helping me to obtain new perspectives to new challenges. His positive attitude mitigated many difficulties I faced during my time at LSU. Also, I would like to thank my co-advisor Dr. Xin Li for all his guidance and support in my research. Our weekly discussions have shaped the work presented in this dissertation. Additionally, I would like to thank Dr. Geralds and Dr. Lopoto for serving on my graduate committee.

A special thanks must go to my best friend Rod Tohid without whom this work would not be possible. I would never be able to thank him enough for his unending support, advice, and encouragement throughout this program. He is the only one who knows how hard these years have been to me and tried his best to comfort me, make me happy and gave me the strength to continue my study. I am grateful for all the time and energy he has given up to support me. Many thanks have to be given to my best friend Hanny Seo for her constant encouragement, support, understanding, patience, and continued effort to keep me happy throughout this trying time. For her heartfelt cares and concerns. And also, thanks to Henry and Ray for all their encouragement and support.With a special mention to James Breedlove for his tremendous efforts and helps. Also Paul for all the lovely lunches and talks we all had.I am appreciative of my friends at school and CCT through all these years mainly Ali and Vahid. Also, my colleagues Sameer, Ye and Zahra.Also, I wish to express my appreciation to all ECE staff particularly Beth Cochran for her constant support.

Finally, I should thank my parents, Shokoufeh and Hadi, for their endless support, unconditional love, patience, and sacrifices. Without their encouragements, I would have never

given myself the chance to continue my study. I appreciate the opportunity they have given me to pursue my education even though having their daughter studying abroad meant constant worry and distress to them. Also, I thank my brothers, Arash and Ashkan, whom I love dearly, for their continuous love, support, and concern. To my sister-in-law, Emily for her cheering and having faith in me, and to my precious niece and nephews. Thanks also to my relatives and close friends who have supported me along the way.

Thanks for all your encouragement!

# Contents

# List of Figures

# ABSTRACT

There are so many applications in data fusion, comparison, and recognition that require a robust and efficient algorithm to match features of multiple images. To improve accuracy and get a more stable result is important to take into consideration both local appearance and the pairwise relationship of features. Graphs are a powerful and flexible data structure, allowing for the description of complex relationships between data elements, whose nodes correspond to salient features and edges correspond to relational aspects between features. Therefore, the problem of graph matching is to find a mapping between the two sets of nodes that preserves the relationships between them as much as possible. This graph-matching problem is mathematically formulated as an IQP problem which solving it is NP-hard, and obtaining exact Optima only plausible for very small data. Therefore, handling large-scale scientific visual data is quite limited, necessitating both efficient serial algorithms, as well as scalable parallel formulations.

In this thesis, we first focused on exploring techniques to reduce the computation cost as well as memory usage of Pairwise graph matching by adopting a heuristic pruning strategy together with a redundancy pattern suppression scheme. We also modified the structure of the affinity matrix for minimizing memory requirement and parallelizing our algorithm by employing CPU's and GPU's accelerated libraries. Any pair of features with similar distance from first image results in same sub-matrices, therefore instead of constructing the whole affinity matrix, we only built the sub-blocked affinity for those distinct feature distances. By employing this scheme not only saved large memory and reduced computation time tremendously but also, the matrix-vector multiplication of gradient computation performed in parallel, where each block-vector calculation computed independently without synchronization. The accelerated libraries such as MKL, cuSparse, cuBlas and thrust applied to solving the GM problem, following the scheme of the spectral matching al-

gorithm. We also extended our work for Multi-graph imaging, since many tasks require finding correspondences across multiple images. Also, considering more graph improves the matching accuracy. Most algorithms obtain approximate solutions for solving the GM NP-hard problem, result in a weak optimal solution. Therefore, we proposed a new solver, which iteratively modified the affinity matrix and binarized the solution by optimizing the original problem with its integer constraints.

# Chapter 1
# INTRODUCTION

A great number of interesting and important applications in computer vision coming from real world relies on algorithms which require finding consistent correspondences between sets of features efficiently. Such as 2D and 3D registration, object/scene matching or recognition. Object category recognition is a challenging problem and defined as locating and recognizing objects of interest in a scene image taken in the real world using models of the known object. The two categories of recognition are the specific case and the generic category case which are shown in Figure 1.1. Being able to identify instances of a particular object, place, or person can be classified as the specific case. Recognizing to which class, different instances of a generic category belongs, for example, buildings, coffee mugs, or cars take place at the category level. In computer vision, the specific object recognition relies on a matching and geometric verification paradigm Logothetis and Sheinberg (1996). However, for generic object categorization, it often includes a statistical model of appearance or shape learned from examples. Categorizing an object requires gathering training images of the given category, and then extracting or learning a model that can make new predictions for object presence or localization in different images. There are many challenges for matching visual objects: variables such as illumination conditions, object pose, camera viewpoint, partial occlusions, and irrelevant background clutter can generate different images of same object which can be seen in Figure 1.2. In this thesis, we focus more on the specific case.

Affine invariant feature detectors have shown to be very useful in object recognition and categorization. These detection algorithms extract visual features from images that are

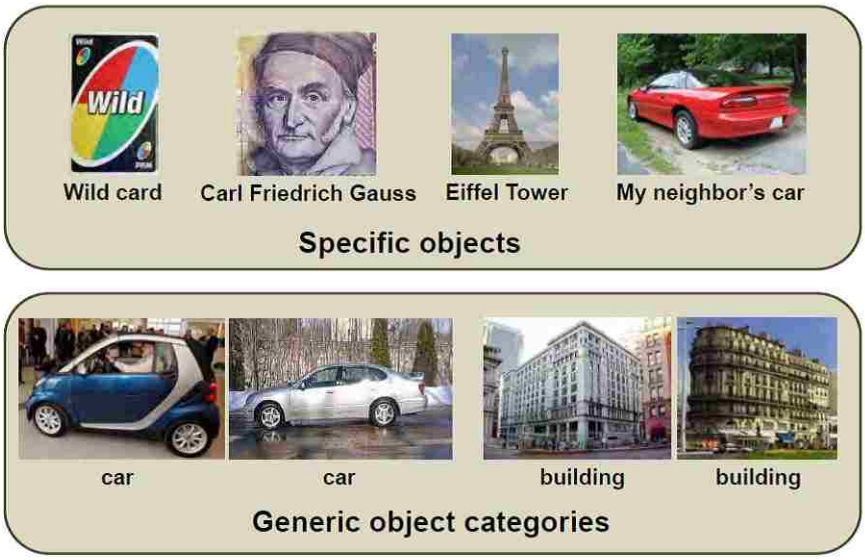



Figure 1.1: The the types of recognition.

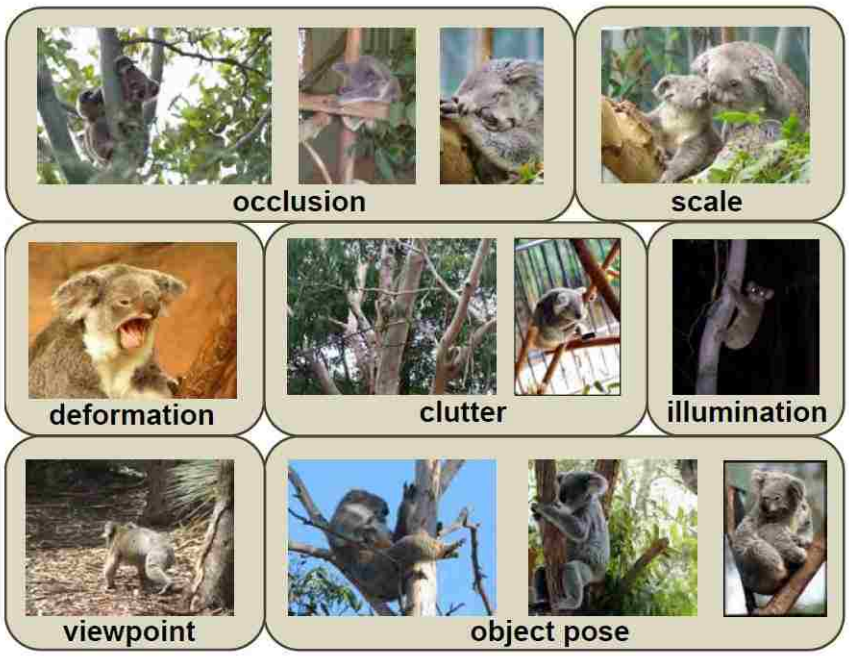



Figure 1.2: Different images of same object.

stable to illumination change, rotation, scale and slight viewpoint change. Local image features can include points, line segments, or curve segments. There are many feature detector methods exist to find a set of distinctive keypoints that are reliably localized in the presence of noise and under varying imaging conditions, viewpoint changes, translation, and rotation. Such as Harris detector, Hessian detector, MSER, SIFT, SURF.

## 1.1 Fundamental Concepts behind Feature Detection and Description

In image processing and computer vision, an image represented by features extracted from it. Although human eye can obtain all information from a raw image effortlessly and instantaneously, is not the same with computer algorithms. Images can be represented in two ways, namely, global features and local features. In the global representation, the image expressed by one multidimensional feature vector, which describes various characters of the image such as texture, color, or shape in the whole picture. On the other hand, local feature representation distinctively describes the image based on some salient regions while remaining invariant to illumination changes and viewpoint. Tuytelaars et al. (2008) define a local feature as "it is an image pattern which differs from its immediate neighborhood". Therefore, local invariant features are used to match local structures between images efficiently. The interest regions or keypoints of the image are a set of local feature descriptors extracted to present the local structures of the image. as illustrated in Figure 1.3.

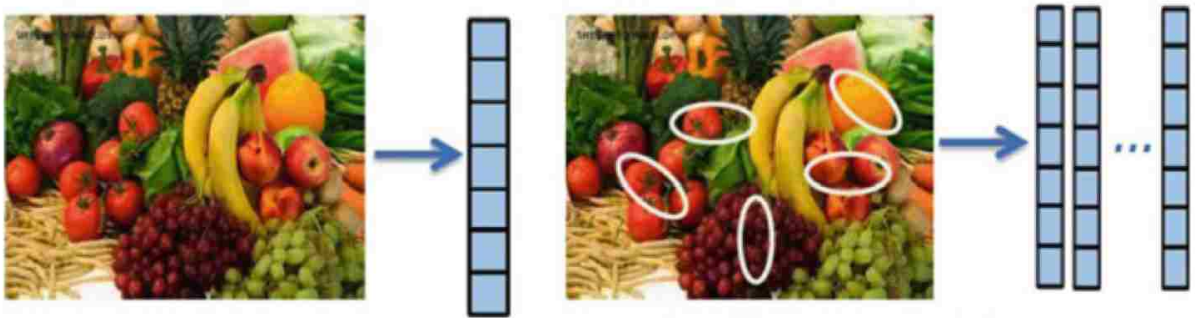


Figure 1.3: Global and local image features representation

Deciding on what type image to use significantly depend on the applications. For instance, in an application where a rough segmentation of the object of interest is available or in the web-scale image indexing application for handling quite big datasets, global features would be more suitable. However, for large-scale image search utilizing local features have much higher performance than global features provide Jegou et al. (2012). Therefore, it is important to employ an appropriate feature detector and extractor with certain properties that are suitable for the particular application. In computer vision, the following properties are necessary for a feature detector to haveTuytelaars et al. (2008).

### 1.1.1 Feature Detector Properties

- *Robustness:* same feature locations independent of scaling, rotation, shifting, photometric deformations, compression artifacts, and noise must be detected.
- *Repeatability:* same features of the same scene or object must repeatedly be recognized under varieties of viewing conditions.
- *Accuracy:* accurately localize the image features (same pixel locations), especially for image matching tasks, where precise correspondences are needed to estimate the epipolar geometry.
- *Generality:* detect features that can be used in different applications.
- *Efficiency:* detect features in new images quickly to support real-time applications.
- *Quantity:* detect all or most of the features in the picture. Where, the number of identified features should indicate the information content of the image for providing a compact image representation.

The feature detectors may classify into three categories, single-scale detectors, multi-scale detectors, and affine invariant detectors. Single scale means that there is only one representation of the features using detector's parameters. These detectors are stable to image transformations such as translation, changes in illuminations, rotation, and an addition of noise. However, they are incapable of dealing with the scaling problem. Therefore it is vital for having multi-scale detectors capable of extracting distinctive features under scale changes. So, same interest points from two images of the same scene that are related to a scale change can be detected. Therefore, for discriminative matching and also to be

insensitive to local image deformations, a set of interest points needs to be detected from a picture at a location $p(x, y)$, scale $s$, and orientation $\theta$. And, their content or image structure in a neighborhood of $p$ needs to encoded in a suitable descriptor. The descriptor should be aligned with $\theta$ and proportional to the scale $s$.

### 1.1.2 Feature Detectors Fundamental Concepts

Some of the general notion behind feature detection methods, which mentioned bellow, are from linear algebra's domain, where the local region of pixels used as a matrix Krig (2014), such as:

*Gradient Magnitude*: It's first derivative of the pixels in the local interest region:
$(\partial f(x, y)/\partial x)^2 + (\partial f(x, y)/\partial y)^2$.

*Gradient Direction*: This is the angle or direction of the largest gradient angle from pixels in the local region in the range$-\pi$ to $\pi$.
$\tan^{-1}(\partial f(x, y)/\partial x)/(\partial f(x, y)/\partial y)$.

*Laplacian*: This is the second derivative ,which can be computed using any of three terms:
$\partial^2 f(x, y)/\partial x^2$, $\partial^2 f(x, y)/\partial y^2$,$\partial^2 f(x, y)/\partial x \partial y$.
However, the Laplacian operator ignores the third term and computes a signed value of average orientation. $(\partial f(x, y)/\partial x)^2 + (\partial f(x, y)/\partial y)^2$.

*Hessian Matrix or Hessian*: A square matrix containing second-order partial derivatives describing surface curvature. The Hessian has several interesting properties useful for interest point detection methods This is based on the second derivative, as is the Laplacian, but the Hessian uses all three terms of the second derivative to compute the direction along which the second derivative is maximum as a signed value.

*Hessian Orientation*: This is the orientation of the largest second derivative in the range $-\pi$ to $\pi$ which is a signed value, and it corresponds to an orientation without direction. The smallest orientation can be computed by adding or subtracting $\pi/2$ from the largest value.

*Determinant of Hessian, Trace of Hessian, Laplacian of Gaussian*: All three names are used to describe the trace characteristic of a matrix, which can reveal geometric scale information by the absolute value, and orientation by the sign of the value. The eigenvalues of a matrix can be found using determinants.

*Eigenvalues, Eigenvectors, Eigenspaces*: Eigen properties are important to understanding vector direction in local pixel region matrices. When a matrix acts on a vector, and the vector orientation is preserved, and when the sign or direction is simply reversed, the vector is considered to be an eigenvector, and the matrix factor is considered to be the eigenvalue. An Eigenspace is therefore all eigenvectors within the space with the same eigenvalue. Eigen properties are valuable for interest point detection, orientation, and feature detection.

### 1.2 Feature Extraction Methods

As it mentioned before there are many feature detection and descriptors, some of the most common detectors, and descriptors are:

*The Harris or Harris-Stephens corner detector* Harris and Stephens (1988): The goal of the Harris method is to find the direction of fastest and lowest change for feature orientation, using a covariance matrix of local directional derivatives. The directional derivative values are compared with a scoring factor to identify corners, edges, and which are likely noise. Many Harris family algorithms can be implemented in a compute-efficient manner. Note that corners have an ill-defined gradient, since two edges converge at the corner, but near the corner the gradient can be detected with two different values with respect to x and y—this is a basic idea behind the Harris corner detector.

*The Hessian (Hessian-Affine) corner detector* Beaudet (1978) is designed to be affine invariant, and it uses the basic Harris corner detection method but combines interest points from several scales in a pyramid, with some iterative selection criteria and a Hessian matrix.

*The Scale Invariant Feature Transform (SIFT)* Lowe (2004) is the most well-known method for finding interest points and feature descriptors, providing invariance to scale, rotation,

illumination, affine distortion, perspective and similarity transforms, and noise. SIFT is a complete algorithm and processing pipeline, including both an interest point and a feature descriptor method. Overall compute complexity for SIFT is high.

*The Speeded-up Robust Features Method (SURF)* Bay et al. (2008) operates in a scale space and uses a fast Hessian detector based on the determinant maxima points of the Hessian matrix. To find feature orientation, a set of HAAR-like feature responses are computed in the local region surrounding each interest point within a circular radius, computed at the matching pyramid scale for the interest point. The SURF and SIFT pipeline methods are generally comparable in implementation steps and final accuracy, but SURF is one order of magnitude faster to compute than SIFT.

*The Maximally Stable Extremal Regions (MSER)* method Extremal et al. (2002) is a connected component of an appropriately thresholded image. The word 'extremal' refers to the property that all pixels inside the MSER have either higher (bright extremal regions) or lower (dark extremal regions) intensity than all the pixels on its outer boundary. The 'maximally stable' in MSER describes the property optimized in the threshold selection process. To compute a MSER, pixels are sorted in a binary intensity thresholding loop, which sweeps the intensity value from min to max. First, the binary threshold is set to a low value such as zero on a single image channel— luminance, for example. Pixels $<$ the threshold value are black, pixels $>=$ the threshold value are white. At each threshold level, a list of connected components or pixels is kept. The intensity threshold value is incremented from 0 to the max pixel value. Regions that do not grow or shrink or change as the intensity varies are considered maximally stable, and the MSER descriptor records the position of the maximal regions and the corresponding thresholds.

## 1.3 Feature based Matching

Typically the feature-based image matching process consist of three basic steps as it can be seen in Figure 1.4: feature detection, feature description, and feature matching. The purpose of using feature extractors is to select features that allow matching local structures between images efficiently. Therefore, it is vital for the extraction process to select distinctive features, and be precise and repeatable, as explained before. After a set of interest points extracted from an image, the next step is to encode in a descriptor that is suitable for discriminative matching. A vector with information to characterize the local visual appearance of that feature is then created. And these descriptors of a given images compared with each other to find the correct correspondences of the pictures.

Good structure of database and efficient searching algorithm is essential for finding the right matching candidates. One of a straightforward and fast method that is applied for matching is called the nearest neighbor. It basically involves computing the distance between all possible pairs of detected features and selecting those points whose distances are closer than some threshold. Although it is very simple to apply, it is not quite accurate, since using only a single threshold value of the Euclidean distance for determining how a feature matches against all the remaining points is not suitable.

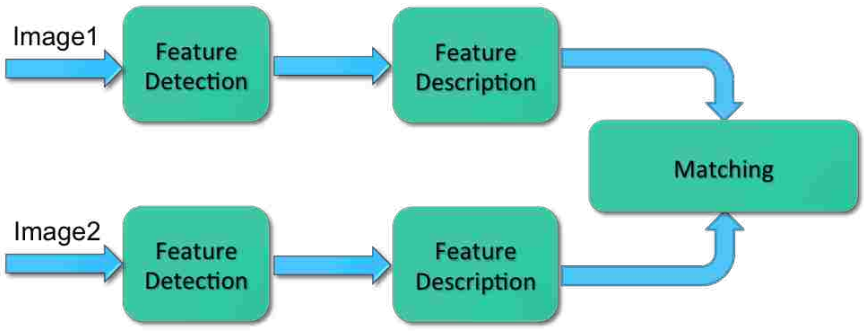



Figure 1.4: Three basic steps of image matching

## 1.4 Graph based Matching

Using only local appearance of each feature will not be descriptive enough for reliable matching between two consecutive frames. Therefore, the higher-order geometry and appearance of groups of features also needed to be considered. Graphs are used as a structure for matching since allows checking not only a single invariant point consistency but also a set of points that somehow have a relationship with each other. Any complex scenes in visual data can be represented as a graph in terms of relations between vertices. Different objects/regions are formulated as attributed graphs whose nodes correspond to salient features and edges correspond to relational aspects between features. In particular, graph matching used in many tasks of computer vision, diverse as object tracking Xiao et al. (2010), object recognition Duchenne et al. (2011b), shape matching, and image labeling among others, which require finding visual correspondences. Which is different from registration methods such as RANSAC Fischler and Bolles (1981), the point based matching Iterative Closet Point (ICP) Zhang (1994), as GM incorporates both the unary node-to-node and the second-order edge-to-edge structural similarity Yan et al. (2015).

A powerful way of describing links between features are pairwise relations, the similarity of the features or their spatial proximity. This naturally leads to the representation of the image as an attributed graph Zager and Verghese (2008). Then comparing features and relations obtained from two images can be reduced to comparing two corresponding graphs which is known as inexact graph matching, i.e., finding the correspondence between the nodes of the graphs that maximizes both nodes similarity as well as edge consistency. Image matching then transforms to finding points among attributed graphs, which optimally assigns nodes to their corresponding ones on the other images, by maximizing both nodes similarity (local appearance of features) and edge consistency (higher-order geometry and appearance of groups of features, such as distances or angles). This graph matching problem is mathematically formulated as an integer quadratic assignment problem that incorporates both the node-to-node (or unary, first-order) as well as the edge-to-edge (or

pairwise, second- order) agreements of two graphs. The are various way of graph matching which can be classified toConte et al. (2004):

1. *Exact graph matching*: requires every node from one graph to be matched to one node from the other (bijective mapping), such that the edge structure is preserved exactly. This is also called *graph isomorphism* and the graphs matched are called *isomorphic*.
2. *Inexact graph matching*: inexact matching consists of finding the mapping that best preserves the edge structure.
3. *Sub-graph matching*: if the two graphs have different number of nodes, than the matching is between a sub-graph of one graph to the other (or a sub-graph of the other).
4. *Weighted graph matching*: if each edge from each graph has some associated weight, and the matching consists of finding the mapping that preserves the edge weights as well as possible, in terms of some distance function.

### 1.4.1 Creating Graphs from Images

In order to create effective image graphs, these techniques need to clearly define what graph nodes and edges represent. This choice affects not only the reliability of the graph topology, but also the ability to create unary node features and binary edge features to describe the image content with.

- Points: may be composed of various types of corner, edge, and maxima shapes. A point needs to be chosen in a way that can be located quickly and computed ideally fast. Therefore, the point is the qualifier or keypoint around which a feature may be described. Given the diverse computer vision literature regarding keypoints Bay et al. (2008); Alahi et al. (2012); Bunke and Bühler (1993) and point set matching Chui and Rangarajan (2003); Caetano et al. (2004); Caetano and Caelli (2006), points are one of the more common choices for graph nodes Krig (2014). Points have an added advantage in that there are many existing keypoint descriptors, which can be used as powerful unary features, depending on the type of images used.
- Line segments: contain additional unary information, in the form of a length and angle. As with points, there are methods which use image data to determine edges and those which use Delaunay triangulation. Unlike points, each pair of line segments contains rotation, translation, and scale (RTS) invariant binary features. Thus, some

works have simply created a fully connected graph of line segmentsLu et al. (2004); van Wyk and van Wyk (2003). The cost here, of course, is that the graph topology cannot be utilized, either as part of the description or simply to speed up the matching process.

- Regions: Regions are another common choice for graph nodes, as they can be easily computed using a variety of computer vision algorithms. Myers et al. Myers et al. (2000) created graph edges using the Delaunay triangulation of the region centroids. However more commonly, the parts are connected by a region adjacency graph. A region adjacency graph connects each region to every other region for which it has adjacent pixels.

- Polylines, Polygons, and Other Shapes: Few works explore the possibility of using polylines, polygons, or other shapes, as graph nodes. The reason behind that is due to the lack of tried and tested edge generation methods for these types of shape. Additionally, the reduction of these forms to a single point, as would be required for Delaunay triangulation, is unlikely to give good results.

### 1.4.2 Graph Matching related work in Computer Vision

Graph matching methods are in general more robust for solving the correspondence problems since the geometrical information encoded in the representation and matching process. Researchers have been significantly working on graph matching in computer vision since the early 70's. Even though its numerous effectiveness has been demonstrated in many computer vision and pattern recognition tasks, solving it is NP-hard, and obtaining exact optima is not possible for moderate to large graphs. This limits its applicability in handling large-scale scientific visual data. Therefore, existing GM methods involve either finding approximate solutions or finding the global optima in polynomial time for a few types of graphs, including the planar graph Luks (1982)and tree structure Aho et al. (1989). Consequently, the main body of research in QAP has focused on producing more accurate and faster algorithms to solve it approximately Zhou and De la Torre (2012). Although extensive research has been done for decades, graph matching is still a challenging problem mainly due to two reasons: 1. In general, the objective function is nonconvex and prone to local minima; 2. The constraints that the solution has to satisfy are combinatorial. In computer vision, GM has diversified and grown along three aspects Leordeanu (2010).

1. defining different objective functions for matching; 2. developing efficient optimization algorithms for obtaining approximate solutions to the matching problem; 3. enhancing the object model represented by the graph, by producing different unary and higher order relationships between nodes and edges/hyper-edges.

#### 1.4.2.1 Different Optimization Algorithms for Graph Matching

Many approximation solutions have been proposed for solving the NP-hard IQP problem.Gold and Rangarajan (1996) presented one of the first of second-order matching methods resolve the IQP problem by the spectral relaxation with power iteration, in which Sinkhorn bistochastic normalization Sinkhorn (1964) is applied to satisfy two-way assignment constraints for a one-to-one mapping. Maciel and Costeira (2003) 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 created, and the discrete domain is relaxed into its convex hull. Leordeanu and Hebert (2005) approximated the IQP problem and solved it based on the assumption that correct corresponding feature pairs are mutually coherent while incorrect pairs are not. Cour et al. (2007) presented similar spectral matching algorithm to Leordeanu and Hebert (2005), and imposed affine constraints on the relaxed solution. Torresani et al. (2008) add a spatial coherence term, supporting spatial aggregation of matched features, which reduces the number of faulty correspondences. Cho et al. (2010) proposed a random walk view for graph matching, in which they incorporated one-to-one mapping constraints by a reweighting jump scheme to overcome local minima in graph matching better. Zass and Shashua (2008) studied the hyper-graph matching algorithm, in which they form the affinity tensor to a one-dimensional vector and refines the vector by projecting it onto the space of soft assignment vectors. Duchenne et al. (2011a) extended the spectral matching method Leordeanu and Hebert (2005) to higher-order connections by using a multi-dimensional power method for tensors, in which the cost function formulated by a tensor encoding the similarity between feature tuples. Lee et al. (2011) generalized their second-order method Cho et al. (2010) using the ran-

dom walk approach to higher-order graph. Leng et al. (2015) addeda perturbation to the eigenvector of the Laplacian matrix and utilize the characteristic that small eigenvalues are sensitive to disturbances, while large eigenvalues are relatively stable, also producing more reliable global solutions to graph matching for noisy scenes and structural corruption. All of these papers solve the IQP problem with an affinity matrix or an affinity tensor as an input, and the time complexity of matching algorithms and the storage size of each similarity matrix depend on the number of feature points. Therefore, a complete computation of the affinity matrix is impracticable when the feature sets are large. Kang et al. (2013) proposed a technique to solve this problem by use of bases and index matrices as well as the bases power method.

### 1.4.3 Outline and Contributions

We briefly enumerate here the main contributions of this thesis:

1. In Chapter 2 theory of feature matching and graph matching are discussed in more details. The spectral algorithms are introduced. The constructing the affinity (weighting) matrix using different methods, and discrete approximation methods are explained.

2. In Chapter 3 using techniques to reduce the computation complexity, as well as memory consumption is proposed. The reason behind benefit of using heuristic pruning strategy together with a redundancy pattern suppression scheme in increasing the accuracy is also explained. Scheme that used to modify the algorithm in order benefit from parallelization using optimized CPU's and GPU's library is suggested. Our proposed technique made solving the graph matching for big datas.

3. In chapter 4 multi image matching is explained, and our methods is extended for solving the multi graph matching. The use of multi imaging is essential in many applications, and being able to solve it is not practical in term of complexity, specially

for big datas. Therefore, employing our approaches and parallelization reduced the computation time significantly, thus, made it possible to find the correct correspondences between three large images.

4. In Chapter 5 more efficient and effective numerical solver is proposed, and the reason behind it is discussed. Potential future works are addressed. Finally conclusion and summary of the research work presented in this thesis discussed.

# Chapter 2
# BACKGROUND

There are numerous tasks in computer vision which demand efficient algorithms for obtaining consistent correspondences between two sets of features, such as shape matching, object recognition, 2-D and 3-D image registration, and wide baseline stereo vision. The problem of correspondence refers to the finding of a mapping between one set of data and another set of data. When the internal structure of these datasets takes into account, they are often considered not simply as datasets, but as two separate graphs. Thus, the correspondence problem generally perceive as graph matching. In this setting, graph vertices represent features extracted from each instance (e.g. a scene image and a model image). Image matching then transforms to finding correspondences among attributed graphs, which optimally assigns nodes to their corresponding ones on the other picture, maximizing both nodes similarity (local appearance of features) and edge consistency (higher-order geometry and appearance of groups of features).

## 2.1 Notations and graph construction

Having two sets of features $P$ and $Q$ extracted from two images each having $n_P$ and $n_Q$ features respectively, the objective is finding a correct correspondence feature in the second image for features in image one. In graph matching algorithm an undirected spatial graph can be presented as two attributed graphs $G^P = (V^P, E^P, A^P)$ and $G^Q = (V^Q, E^Q, A^Q)$ with vertex set $V$ and edge set $E$. For each node $i \in V^P$ and $i' \in V^Q$ there is an associated feature vector $A_i^P \in A^P$ and $A_{i'}^Q \in A^Q$ respectively. These features commonly describes the local appearance at node $i$, and $i'$. Moreover, for each edge $(i, j) \in E^P$ and

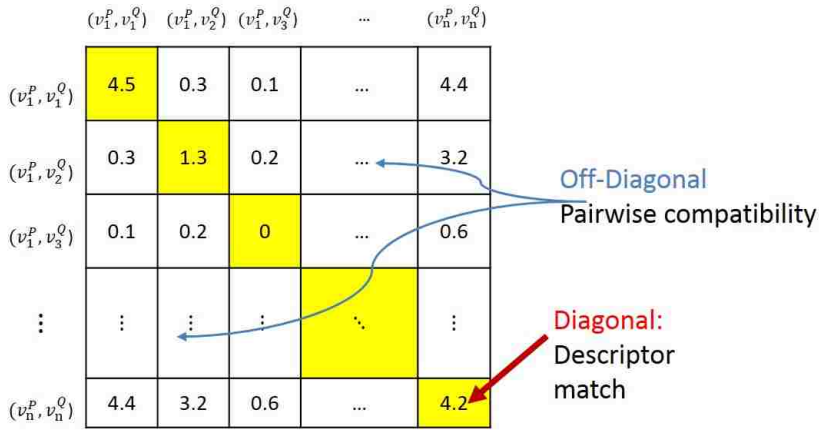



Figure 2.1: Weighted matrix M.

$(i', j') \in E^Q$ there is an associate vector $A_{ij}^P \in A^P$ and $A_{i'j'}^Q \in A^Q$ which describes the pairwise geometric relationship between nodes $i$ and $j$, and nodes $i'$ and $j'$ respectively. Instead of using a binary value, 1 or 0 indicating whether or not there is a connecting edge between a pair of nodes, an affinity(weighted) matrix can be built, where the entries of the matrix are weights that reflect the strength of a pairwise adjacency relationship. To match these two graphs, one need to find a mapping between $V^P$ and $V^Q$ that best conserves the attributes between edges $A_{ij} \in E^P$ and $A_{i'j'} \in E^Q$. There are various ways to construct the weighted matrix $M$, which is presented in the following section. Figure 2.1, shows the affinity matrix that is created using the candidate assignments of the graph. For each node $A_i^P$ and $A_{i'}^Q$ the unary score function can be defined as $M_{ii',ii'} = S(A_i^P, A_{i'}^Q)$, representing how well the local appearance between the candidate matches agrees, which are stored on the diagonal of matrix M. For each pair of edges $(i, j) \in E^P$ and $(i', j') \in E^Q$ there is a score function $M_{ii',jj'} = S(A_i^P, A_{ij}^P, A_{i'}^Q, A_{i'j'}^Q)$, stored in the off-diagonal of matrix $M$, measures the second-order relationships (defined by $A_{ij}^P$ and $A_{i'j'}^Q$) between the pair of candidate correspondences $(i, i')$ and $(j, j')$, contain information regarding how well the pair-wise geometric information between candidate assignments is preserved.

Furthermore, for finding the optimal solution to feature matching problem, most of the methods use the graph matching formulation based on integer quadratic problem (IQP).

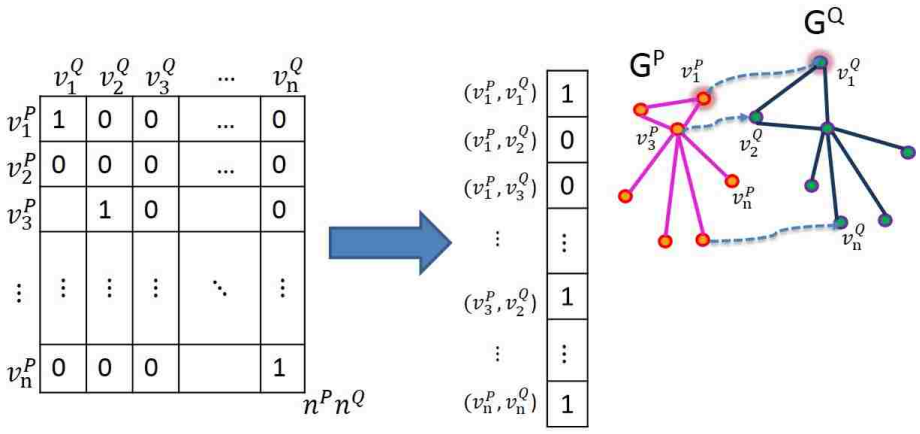



Figure 2.2: Indicator vector.

Since, it incorporates both the node-to-node (unary, first-order) as well as the edge-to-edge (pairwise, second-order) agreements, for finding the indicator vector $x^*$ that respects certain mapping constraints (such as one-to-one or many-to-one) and maximizes the quadratic score function:

$$x^* = argmax(x^T M x) \quad \text{s.t. } Ax = 1, x \in \{0,1\}^{n^P n^Q}, \tag{2.1}$$

$x$ is an indicator vector, Figure 2.2, such that $x_{ii'} = 1$ if feature $i$ from one node image is matched to feature i' from the other image and zero otherwise, given the constraints $Ax = 1$, and $x \in \{0,1\}^{n^P n^Q}$. The one-to-one constraints imposed on x such that one feature from one image matched to only one feature from the other image. Graph matching problem formulated as a constrained quadratic assignment problem that focused mainly on finding the mapping between the nodes of two graphs such that the pair-wise similarity preserved as well as possible.

## 2.2 Spectral correspondence by pairwise feature geometry

The geometric relationship between feature points has been enforced into correspondence problemDuchenne et al. (2011a); Cour et al. (2007), where feature geometric relationship

formulated as a spatial graph. To characterize the global structural properties of graphs using the eigenvalues and eigenvectors of either the adjacency matrix or the closely related Laplacian matrix spectral graph theory employed as an approximate solution for graph matching Chung (1997). The main purpose of spectral graph theory is to utilize the distribution of eigenvalues to provide a complete knowledge of graph structure Carcassoni and Hancock (2003).

### 2.2.1 Different weighting functions

The purpose of constructing the wighting matrix is to calculate the probability of adjacency relations between vertices in the graph. Many ways have been suggested in the literature to construct the affinity matrix, which can be classified according to a broad-based taxonomy from their derivative Carcassoni and Hancock (2003). If the derivative is monotonically increasing, then the weighting function is increasing. If the derivative is asymptotically constant, then the weighting function is sigmoidal. Finally, if the derivative asymptotically approaches zero, then the weighting function is re-descending. Given two points $i$ and $j$ with position $y_i$ and $y_j$, the corresponding element in the affinity matrix for these two vertices in a graph representation is given by different weighting functions as follows Figure 2.3. $\sigma$ controls the width of weighting kernel.

- Gaussian Wighting:

$$m_{ij} = \exp\left[-\frac{1}{2\sigma^2}\left\|y_i - y_j\right\|\right] \tag{2.2}$$

This weighting function is re-descending.

- Sigmoidal weighting:

$$m_{ij} = \frac{2}{\pi\left\|y_i - y_j\right\|}\log\cosh\left[\frac{\pi}{\sigma}\left\|y_i - y_j\right\|\right] \tag{2.3}$$

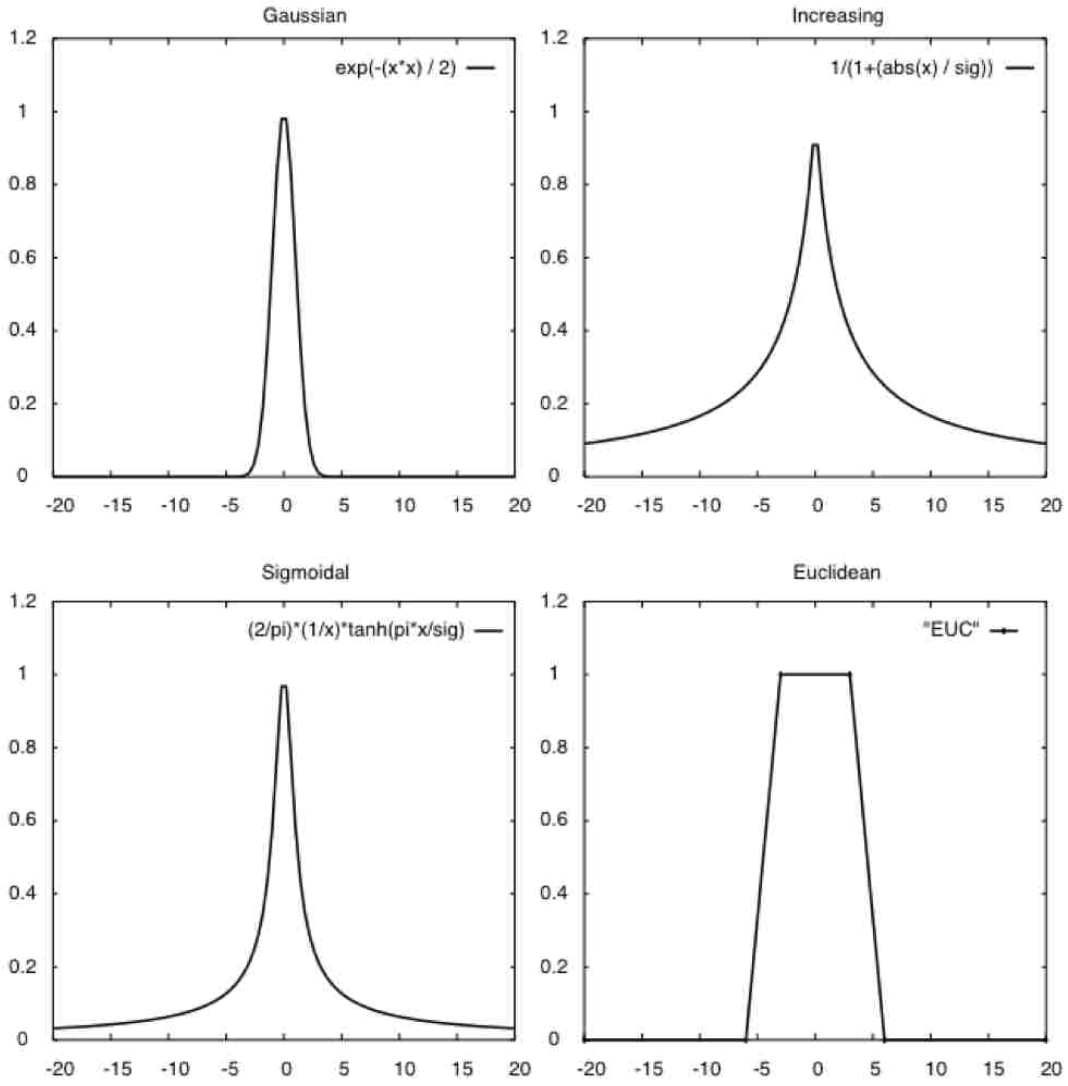



Figure 2.3: Weighting functions

- Increasing weighting:

$$m_{ij} = \left[1 + \frac{1}{\sigma} \|y_i - y_j\|\right]^{-1} \tag{2.4}$$

- Euclidean weighting function:

$$m_{ij} = K(\|y_i - y_j\|), \text{ where } K(\eta) = \begin{cases} 1 & if\ k \prec \sigma_1 \\ 1 - \frac{1}{\sigma_1 - \sigma_2}[\eta - \sigma_1] & if\ \sigma_1 \prec k \prec \sigma_2 \\ 0 & otherwise \end{cases} \tag{2.5}$$

Here $\sigma_1$ is the half-width of the ceiling of the function and $\sigma_2$ is the half-width of the base.

### 2.2.2 Eigen Decomposition for Finding Correspondences

The correspondences between the two data sets of the affinity matrix can be found by solving the eigenvalue and computing the associated eigenvectors. The eigenvectors are ordered according to the values of their associated eigenvalues and are used to construct a pattern matrix. The eigenvectors of the affinity matrices can be observed as the base vectors of an orthogonal transformation on the original data identities. The matrix structure for a data set is attained by solving the eigenvalue equation

$$|M - \lambda I| = 0 \tag{2.6}$$

where $\lambda_i,\ i = 1, \ldots, n$ is the $i^{th}$ eigenvalue of matrix $M$, and its associated eigenvector $v_i$ is computed using the equation

$$Mv_i = \lambda_i v_i \tag{2.7}$$

eigenvectors are arranged according to the values of their associated eigenvalues.

### 2.2.3 Spectral Approximation for Integer Quadratic Programming Problem

Spectral matching made it quite successful for solving computer vision tasks by influencing a few improvements over previous methods. It has broad applicability because it does not impose any constraints on the unary and pairwise scores. The overall requirements are for

the scores to be non-negative and increase with the quality of the agreement, but that can be easily provided by all graph matching applications. It reduces the complexity of the problem from NP-hard to a low-order polynomial complexity by dropping the integer constraints on the solution and solve the problem by computing the leading eigenvector of the affinity matrix. It is quite intuitive, easy to understand and implement. Its solution is based on an intuitive insight into the structure of the similarity matrix, based on the expected geometric alignments between correct assignments against the accidental associations of incorrect ones. This idea inspires the appropriate design of meaningful first and second-order scores. These properties contribute to the popularity of spectral matching, as the preferred choice for many computer vision applications Leordeanu (2010).

Having the affinity matrix $M$, the correspondence problem summarize to finding the best cluster $C = \{i, i'\}$ of assignments, so as to maximize the graph matching score defined as:

$$Max \sum_{ii' \in C,\, jj' \in C} S\left(A^P_{ij}, A^Q_{i'j'}\right) \tag{2.8}$$

The function $S(.,.)$ measures the similarity between edge attributes $S(A^P_{ii}, A^Q_{i'i'})$ is simply the score associated with the match $i, i'$.

The affinity matrix $M$ is a $\left(n^P \mathrm{x} n^Q\right) \mathrm{x} \left(n^P \mathrm{x} n^Q\right)$ matrix that each entry indicates relationships between two pairs of the candidate correspondences, i.e. $m_{ij} = S(A^P_{ij}, A^Q_{i'j'})$. Therefore, pairwise geometric relationships are combined to form the affinity matrix. To utilize pairwise information equation 2.8 can be reformulated as an integer quadratic programming equation 2.1. The quadratic programming, takes into account both unary and second-order terms. This reflects the similarities in local appearance (candidate correspondences), as well as in the pairwise geometric relationships (pairwise agreements) between the matched features. The difficulty of quadratic programming problem depends on the structure of the matrix $M$, but in most cases it is NP-hard, and there is no efficient algorithm that can guarantee optimality bounds. Therefore, so much effort has been made in finding good

approximate solutions by relaxing the integer one-to-one constraints to obtain an optimal solutions to the new problem efficiently. Spectral approximation of this problem, referred to as spectral matching, takes advantage of the particular properties of geometric matching. It is expected that the optimum solution of the relaxed problem to be close to that of the original problem with integer constraints.

Spectral Matching (SM) in Leordeanu and Hebert (2005) drops the constraints entirely and only incorporates it during the discretization step. They also fixed the norm of $x$ to $1$, since the relative values between the elements of $x$ matter. The resulting objective function is:

$$x^* = argmax(x^T M x) s.t. \left\| x \right\| = 1 \tag{2.9}$$

In this method the only constraint is that the unary and pair-wise scores should be non-negative and they should decrease with the deformation errors between the candidate correspondences (increase with the quality of the match). Based on the assumption that correct corresponding feature pairs are mutually coherent and form a strongly connected cluster, while incorrect pairs are not, a graph matching algorithm extracts the correct assignments of the weighted affinity matrix, by computing the leading eigenvector of matrix M. And, the elements of the eigenvector can be interpreted as confidences that each candidate assignment is correct. From the Rayleigh's ratio theorem, $x$ that will maximize the dot-product $x^T Mx$ is the principal eigenvector of $M$. Since $M$ has non-negative elements, by Perron-Frobenius theorem, the elements of $x$ will be in the interval $[0, 1]$. Then the continuous solution of $x$ is binarized by maximizing the dot-product with the leading eigenvector of $M$. The assumption is that the affinity matrix is a slightly perturbed version of an ideal matrix, with the rank =1, for which maximizing this dot product will give the global optimum of 2.9. Ideal matrix with a rank = 1 is shown in Figure 2.4, where correct correspondences are strongly connected to form the main cluster in M.

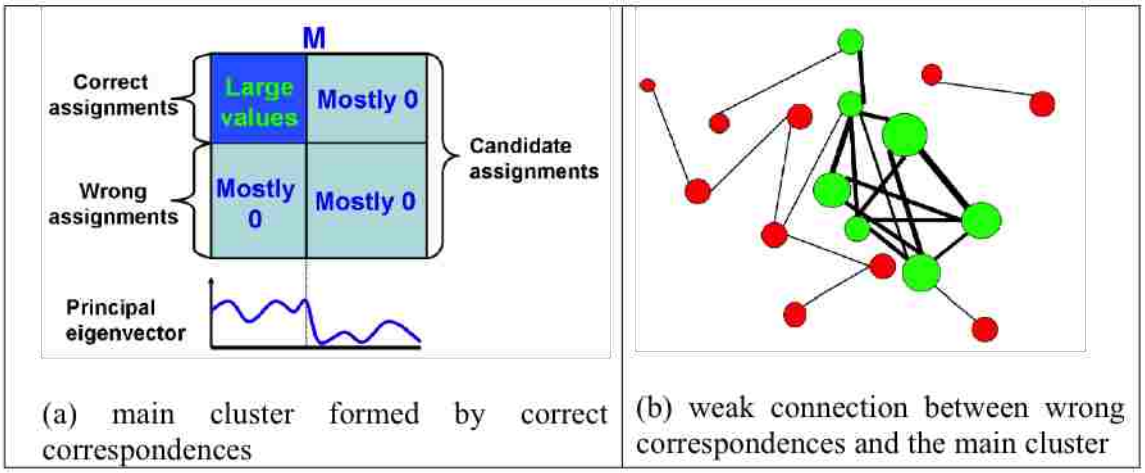



Figure 2.4: Ideal matrix with rank = 1

Spectral Graph Matching with Affine Constraints (SMAC) was developed later in Cour et al. (2007), which determines the optimal solution of a modified score function, with a tighter relaxation that imposes the affine constraints $Ax \in b$ (general form of mapping constraint) during optimization and dropping the binary constraints on $x$.

$$x^* = argmax \frac{x^T M x}{x^T x} s.t. Ax = b \tag{2.10}$$

Their proposed solution is given by the leading eigenpair of $P_A M P_A x = \lambda x$, where $x$ is scaled so that $Ax = b$; $P_A = I_n - A_{eq}^T \left(A_{eq} A_{eq}^T\right)^{-1} A_{eq}$, $A_{eq} = [I_k, 0]\,(A - (1/b_k)\, bA_k)$; $A_k$ and $b_k$ denotes the last row of $A$, $b$, respectively; and k is the number of constraints. An important aspect is that $P_A$ is in general, a full matrix even when $M$ is sparse. The discrete solution is obtained to binarize the continuous $x$.

### 2.2.4 Spectral Graph Matching

Having two candidate assignments $a = (v_i, v_{i'})$ and $b = (v_i, v_{i'})$, for simplicity $a = (i, i')$ and $b = (j, j')$, the pairwise score of $M(a, b)$ can be calculated by:

$$M(a,b) = \begin{cases} 4.5 - \frac{(d_{i,j} - d_{i',j'})^2}{2\sigma_d^2}, & \text{if } \left|d_{i,j} - d_{i',j'}\right| \prec 3\sigma_d; \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

where $d_{i,j}$ and $d_{i',j'}$ are the distances between the points $i$ and $j$, and between their candidate matches $i'$ and $j'$ respectively. $sigma_d$ is a parameter to control the sensitivity of the matching: larger $sigma_d$ allows more deformations of the data, and more pairwise relationships to have positive scores in the matrix $M$. As it can be seen in Figure 3.5 $M$ is a symmetric matrix, and all of its elements are positive placed in the range of 0 to 4.5. Intuitively, the value of $M(a, b)$ is big if the two pairs $a = (i, i')$ and $b = (j, j')$ are spatial consistent, meaning the value of $|v_i - v_j|$ and$|v_{i'} - v_{j'}|$be close as much as possible.

Given the matrix $M$, the correspondence problem reduces to finding the best cluster $C$ of assignments $i, i'$ that maximize the matching score $S$, such that the mapping constraint are met.

$$S = \sum\nolimits_{a,b \in C} M(a, b) = x^T M x, \tag{2.12}$$

where $x$ is an indicator vector with an binary value for each assignment, such that $x(a) = 1$, $a = (i, i')$, if feature $i$ of image 1 feature set $(P)$ is matched with feature $i'$ of image 2 feature set $(Q)$, and 0 otherwise.

Given the mapping constraints, the optimal solution is the binary vector $x^*$ that maximizes $S$.

$$x^* = argmax(x^T M x) \tag{2.13}$$

The matching score $S$ is mainly depends on number of links adjacent to each correspondence pair, and the weights on those links. In order to solve the Equation 2.13, the Integer Quadratic Problem, Leordeanu and Hebert (2005), dropped both the mapping and integral constraints on $x$, such that the value of its elements be real and in $[0, 1]$. Then the value of $x^*(a)$ interpreted as the association of $a$ with the best cluster, requiring the norm of $x$ to be 1. Then, by the Raleigh's quotient theorem Lange (2010), the solution of Equation 2.13 is given by the largest eigenvalue of matrix $M$. And, by Perron-Frobenius theorem Horn

**Algorithm 1** Spectral Matching Algorithm

1. Build the symmetric non-negative matrix $\mathbf{M}$ using 2.11 formulation.
2. Initialize the solution vector $x$ with zeros. Find the the principal eigenvector of $\mathbf{M}$, $x^*$.
3. Find the maximum value of $x^*$, and set that assignment in $x$ to 1.

   $a^* = argmax_{a \in C}(x^*(a))$, where $C$ is the set of all the candidate assignments.
4. If $x^*(a^*) = 0$, stop the binarization. Otherwise set $x^*(a^*) = 1$ and remove that assignment from $C$.
5. Remove from $C$ all the potential assignments which conflict with $a^* = (i, i')$, that have the form of $(i, *)$ and $(*, i')$.
6. If $C$ is empty return the solution $x$, otherwise return to step 3.

and Johnson (2012) the elements of $x^*$ will be in the interval $[0, 1]$, since values of elements $M$ are positive. Since, the mapping constraint is dropped at the optimization step, the binarization of $x^*$ is needed to obtain a robust solution.

The overall steps of spectral technique including the binarization are as follows:

The spectral matching algorithm has been used successfully for small data. However, there is a scalability issue in computing the eigenvector of $\mathbf{M}$. The size of $\mathbf{M}$ is $n^P n^Q$x $n^P n^Q$, where $\mathbf{P}$ and $\mathbf{Q}$ are the nodes of graph1 and graph2 respectively, and the number of nonzero elements in the matrix $\mathbf{M}$ can grow in proportion to number of nodes. Therefor constructing $\mathbf{M}$ explicitly in memory is prohibitive. Kang et al. Kang et al. (2013), proposed a solution to this problem by constructing an approximated affinity matrix $\hat{\mathbf{M}}$ which is explained in the next chapter.

### 2.2.5 Eigenvector Computation

For computing the principle eigenvector of a matrix a very simple algorithm called the power method can be used. This algorithm converges geometrically to the largest eigenvalue of the input matrix Golub and Van Loan (2012). In this algorithm the matrix $\mathbf{M}$ multiplied

**Algorithm 2** Power Method

**Input:** *matrix* **M**
**Output:** *Principle eigenvector* $v$ **of M**

1 : **initialize** $v \leftarrow$ *random vector with* $||v|| = 1$;

2 : **repeat**

3 : $v \leftarrow$ **Mv;**

4 : $v \leftarrow v/||v||$;

5 : **until convergence**

by a randomly initialized vector until the normalize vector converges, which is the principle eigenvector. The following algorithm describes this computation.

### 2.3 Accelerated libraries

Libraries such as MKL (Intel Math Kernel Library) and GPU accelerated libraries provide highly optimized algorithms and functions that increase application performance and reduce execution time tremendously. Using libraries enables GPU acceleration without in-depth knowledge of GPU programming. Many GPU-accelerated libraries follow standard APIs, thus enabling acceleration with minimal code changes. Libraries offer high-quality implementations of functions encountered in a broad range of applications.

The reason for employing these accelerated libraries are for optimized matrix vectors multiplication and also sparse matrix vector operation. Since most of our sub-matrices are sparse using GPU library cuSparse, and CPU MKL sparse library are quite suitable for improving the performance.

### 2.3.1 MKL

Intel Math Kernel Library (MKL) accelerates math processing and neural network routines that increase application performance and reduce development time. MKL includes highly vectorized and threaded Linear Algebra, Fast Fourier Transforms (FFT), Neural Network, Vector Math and Statistics functions. The easiest way to take advantage of all of that processing power is to use a carefully optimized math library. Even the best compiler can't compete with the level of performance possible from a hand-optimized library. If your application already relies on the BLAS or LAPACK functionality, simply re-link with Intel MKL to get better performance on Intel and compatible architectures.

Using Intel MKL can save development, debug and maintenance time in the long run because today's code will run optimally on future generations of Intel processors with minimal effort. Intel has engineered this ready-to-use, royalty-free library, to allow you to focus on and deliver features your customers have requested.

### 2.3.2 cuBLAS

cuBLAS, or the CUDA Basic Linear Algebra Subroutines (cuBLAS) library, is a GPU-optimized set of Linear Algebra functions. Just as Thrust is a CUDA analogue to the C++ STL, cuBLAS is based on Intel's MKL BLAS. cuBLAS boasts support for all standard BLAS routines, as well as tremendous speedup. cuBLAS was used in the tests to perform matrix multiplication, specifically using the cublas Sgemm() function. This is useful because this function performs the matrix multiplications and allows us to transpose the matrices all in one function call. 5.1.1.2

### 2.3.3 cuSparse

The NVIDIA CUDA Sparse Matrix library (cuSPARSE) provides a collection of basic linear algebra subroutines used for sparse matrices that delivers up to 8x faster performance than the latest MKL. The cuSPARSE library is designed to be called from C or C++, and the latest release includes a sparse triangular solver.

### 2.3.4 Thrust

Templated Parallel Algorithms & Data Structures. One of the most prevalent API sets available is Thrust. Thrust is a library that closely emulates the C++ Standard Template Library. One useful element of Thrust is the provision of data structures, such as vectors, that can be created in CPU or GPU-space. The Thrust library was specifically used in the GPU integer SAXPY algorithm in order to create and populate vectors on the CPU and GPU, and performing vector transformations.

In the next chapter the impact of local and global relationship of features for finding the correct correspondences between two images are discussed. The approximated affinity construction is applied to our work, and modification to the framework of spectral matching algorithm for reducing the overall computation, and parallelization are all explained with our results in more details in the next chapter.

# Chapter 3
# PAIRWISE MATCHING OPTIMIZATION AND PARALLELIZATION USING ACCELERATED LIBRARIES FOR DIFFERENT ALGORITHMS

## 3.1 Pairwise Matching

The overall graph matching pipeline of our algorithm for two images using spectral matching is shown in Figure 3.1.

Given a pair of images, we first extracted invariant features $n^P, n^Q$ from them respectively using one of the detection method; in this thesis, we used SIFT and MSER. Then graph $G^P = (V^P, E^P)$, and $G^Q = (V^Q, E^Q)$ constructed, where, vertices represent the extracted features, and edges between them represent the euclidean distance between those features. Next graph constructed, where, vertices represent the extracted features, and edges between them interpret the relationship between those features. The objective is finding a

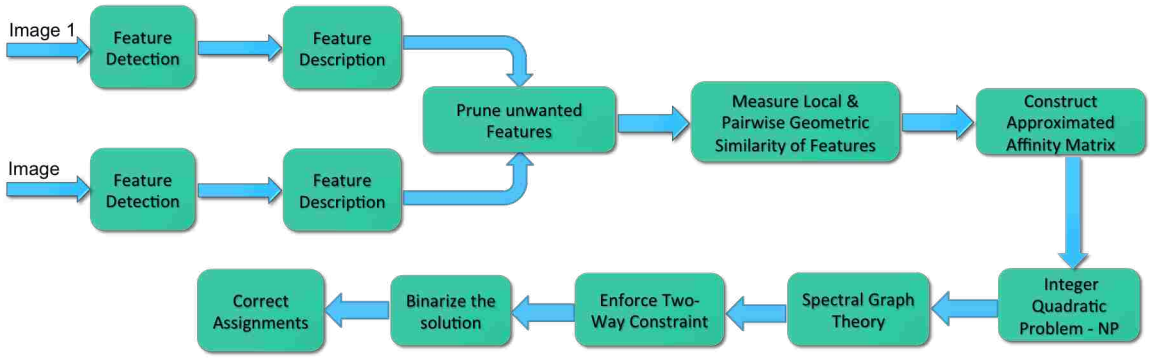



Figure 3.1: Overall pipeline of our algorithm using spectral technique

correspondence mapping to optimize a given similarity-based cost function.

After the graph constructed for the two given images, the affinity matrix created for the list of candidate assignments **L**. Where the affinities for every assignment $a \in L$ and every pair of assignments $(a, b) \in L$ stored. For each candidate assignments $a = (i, i')$ the associated score or affinity $M(a, a)$ that measures how well feature $i \in P$ matches $i' \in Q$ can be computed using their descriptors, as equation 3.1. For each pair of assignments $(a, b)$, $a = (i, i')$ and $b = (j, j')$, the score of $M(a, b)$ that indicate how compatible the data features $(i, j)$ are with the model features $(i', j')$ can be estimated using equation 2.11. The unary and the pairwise scores of assignments stored on the diagonal and off-diagonal of the affinity matrix$M$ respectively.

$$d(H_i, H_{i'}) = 1 - \|H_i - H_{i'}\| = 1 - \frac{\sqrt{\sum_{s=1}^{128} (H_i^s - H_{i'}^s)^2}}{128} \tag{3.1}$$

### 3.1.1 Approximated Affinity Matrix Construction

The affinity matrix constructed by equation 2.11 is composed of several sub-matrices with size of $n^Q$ by $n^Q$ as shown below, where the subscripts of $M$ indicating to which pair of nodes from graph1 it corresponds to. Such as, the sub-matrix $M_{12}$ constructed using nodes *1* and *2* of graph1 and all the graph2 vertices. This can be illustrated more easily on Figure 3.6. As it can be seen the red bounding rectangles sub-matrix constructed using the vertices *1* and *2* of graph1 and all the vertices of graph2 in Figure 3.5, which contribute to high redundancy in the affinity matrix.

$$M = \begin{bmatrix} 0 & M_{12} & M_{13} & \dots & M_{1n_P} \\ M_{21} & 0 & M_{23} & \dots & M_{2n_P} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ M_{n_P1} & M_{n_P2} & M_{n_P3} & \dots & 0 \end{bmatrix}$$

Also, it can be seen from Figure 3.5, that the distance between the points 3 and 4 is almost the same as the points 5 and 6 which resulted in the sub-matrices with the red bounding rectangles. Also. the the points 3 and 5 is almost the same as the points 4 and 6 values are almost the same therefore the sub-matrices with the green bounding rectangles contain similar entries. Therefore, we can say that if the value of distances between pairs of point in a first graph is the same, then their corresponding sub-matrices values would be same.

In order to have the same sub-matrices, Kang et al. (2013) proposed the idea to approximate the pairwise distances by using 3.2.

$$\hat{d_{ij}} = w(\lfloor \frac{d_{ij}}{w} \rfloor + \frac{1}{2}), \tag{3.2}$$

where, $w$ is the width of a bin, it is used to approximated the distances placed in the same bin be approximated to the center distance of that bin.
Furthermore, for two pints $i, j$ of $P$ the $i', j'$element of sub-matrix $\hat{M}_{ij}(i', j')$ can be determined by the following equation,

$$\hat{M}_{ij}(i', j') = \begin{cases} 4.5 - \frac{(\hat{d_{ij}} - d_{i',j'})^2}{2\sigma_d^2}, & \text{if } \left|\hat{d_{ij}} - d_{i',j'}\right| \prec 3\sigma_d; \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Having $z$ distinct approximated pairwise distances $\hat{d_{ij}}$, $z = \frac{d_{max}}{w}$, which $d_{max}$ is the maximum distance of the points in $P$, Equation 3.3 creates $z$ distinct $n^Q$ x $n^Q$ sub-matrices of $\hat{\boldsymbol{M}}$. Thus, the amount of memory required for storing the values of affinity matrix is reduced form $n^P$ x $n^P$ by $n^Q$ x $n^Q$ to $z$ by $n^Q$ x $n^Q$ . By exploiting this technique the computation time dropped since requiring much less memory can benefit from the cache functionality.

### 3.1.2 Candidate Assignments of Different Approaches

The size of the affinity matrix depends on the number of candidate assignments **L**, a larger the number of candidates list, the bigger would be the size of the similarity matrix $M$. In this thesis the candidate assignments are selected in four different approaches which are as follows and illustrated in more details later in this section.

1. *Completed graph matching* (CGM): all the extracted features from the two given images considered to construct the graphs. Therefore, $n^P \mathrm{x} n^Q$ would be the number of candidates**L**, and consequently $n^P \mathrm{x} n^Q \mathrm{x} n^P \mathrm{x} n^Q$ would be the size of affinity matrix.
2. *Approximated graph matching* (AGM): in this method similar to previous method graphs constructed using all the extracted features. Moreover, the number of candidates **L** stay same as the previous method, $n^P \mathrm{x} n^Q$, but the size of the affinity matrix shrinks to $z$ number of distinct $n^Q$ by $n^Q$ sub-matrices. Where z is the unique value of the approximated distances $d_{i,j}$.
3. *Preprocessed filtered out graph matching* (PFGM): in this technique KNN method used to find the locally matched features. Resulting in $kn^P$ number of candidates **L**. Where $k$ is the number of the closest neighboring corresponding features for each point. Consequently, $kn^P x\, kn^P$ for the size of affinity matrix.
4. *Pruned blocked graph matching* (PBGM): this is our proposed method for selecting the assignment candidates. By pruning the less probable correct assignments the affinity constructed with the size of $n^{P'} \mathrm{x} n^{Q'} \mathrm{x} n^{P'} \mathrm{x} n^{Q'}$, and the number of candidate assignment would be $z'$ by $n^{P"} \mathrm{x} n^{Q"}$. Where $n^{P"} \mathrm{x} n^{Q"}$ represent a number of featured selected after our pruning system, and $z'$ represent the number of distinguished blocked sub-affinity matrices.

We compared the result of these four approaches in term of accuracy and computation time, and it proved that our method, the PBGM, outperforms the other methods and subsequently required much smaller memory.

## 3.2 Graph Matching using Different Algorithm

As described before, explicitly constructing the weighted matrix **M** and iteratively computing the first eigenvector of the matrix demands a huge storage, and it is computationally expensive. The majority of graph matching techniques such as Cho et al. (2010); Duchenne

et al. (2011b); Lee et al. (2011); Aho et al. (1989) applied algorithms like K-Nearest Neighbor (KNN) method for finding the matches between sets of feature points as the candidate assignments. As a result, the number of candidate assignments reduces from $n^P x n^Q$ to $k n^P$ resulting in smaller affinity matrix and reducing the dimension of the problem search space. The local appearance provides an efficient filter for selecting promising matching. Therefore, using methods such as KNN for finding the initial matches might be beneficial for finding the global optima. On the other hand, there have been many researchers that applied all the feature points as their assignment candidates to improve the accuracy score. Using more candidate assignment reduce the ambiguity for finding the correct correspondences. Since, correct assignments establish agreement links between them, by using more candidates, more link to the actual correspondences occurs, which, increases the association score. However, that increased the dimension of the problem space.

To get the benefit from the approximated technique, and also use the preprocessed initial matches we decided to combine these two for making our candidate assignments. So, in our algorithm we first matched the feature points locally, to pruned the assignments with lower probability of being correct, then added additional candidates in order to accommodate for constructing the approximated affinity method. So, instead of using all of the feature points as our assignment candidates, we constructed new candidate assignments, which we call it *Pruned Blocked Graph Matching(PBGM).* Our candidate assignment decreased from $n^P \mathrm{x} n^Q$ to $n^{P"} \mathrm{x} n^{Q"}$, and also instead of constructing the whole affinity matrix we only construct the $z'$ blocks. Therefore, the affinity matrix size shrink to $z'$ blocks of$n^{Q"} \mathrm{x} n^{Q"}$. The benefits of constructing these blocks is that they can constructed separately and their eigenvector can computed individually, thus they can constructed and computed in parallel.

**Algorithm 3** PBGM

1. Detect and extract feature points from the given two images, $P$, and $Q$ .
2. Find locally matched features, using nearest neighbor method $P'$, and $Q$' .
3. Pruned unwanted features that are not included in the list from step2 $P$", and $Q$" .
4. Calculate the approximate distances between pair of all the features from feature set $P$", $\tilde{d_{ij}}$.
5. For each distinct $\tilde{d_{ij}}$ locate $\forall i, j$ which corresponds to that same value.
6. $\forall i, j$ find their corresponding matching points in the initial matching list, and combine those points and add them to candidate assignment list, in order to have a same block for each distinct $\tilde{d_{ij}}$.
7. For each distinct $\tilde{d_{ij}}$, build the sub-matrix blocked, $B' = \left\{B'_0, B'_1, \ldots, B'_{z'}\right\}$.
8. For each block $B'$ save the indices that corresponds to the distinct $\tilde{d_{ij}}$.
9. Calculate the eigenvector by multiplying each sub-matrix blocked affinity $\mathbf{B'}$ with $v(y-1) * n^{Q"} + 1 \; : \; y * n^{Q"}$ and add the result to $v(x-1) * n^{Q"} + 1 \; : \; x * n^{Q"}$. Where $x$, and y are the nodes indices belonging to the sub-blocked affinity.

The overall pipeline of our work is in Algorithm 3. To demonstrate the overall steps of our work more clearly two images of different mug being used as in 3.2. In the first step the salient features detected and extracted using SIFT method. From image1 6 points are extracted {1, 2, 3, 4, 5, 6}, and from image2, 5 points {A, B, C, D, E}. Then the nearest neighbor method applied to match those feature points locally, the initial matching points are shown in the right side of the Figure 3.3. As it can be seen Figure 3.3 the matching points are {1A, 1D, 3C, 4A, 4D, 5E, 6E}. After the features are extracted and matched locally the next step is constructing the graphs of the images 3.4 and finding their pairwise distances 3.5.

We can see that the distances between nodes $v_3$ , $v_4$ and nodes $v_5$ , $v_6$ are quite similar $d_{34} \simeq d_{56}$, the same thing applies between vertices $v_3$ , $v_5$ and nodes $v_4$ , $v_6$ , $d_{35} \simeq d_{46}$,


Figure 3.2: Extracted features of the mugs using SIFT algorithm.

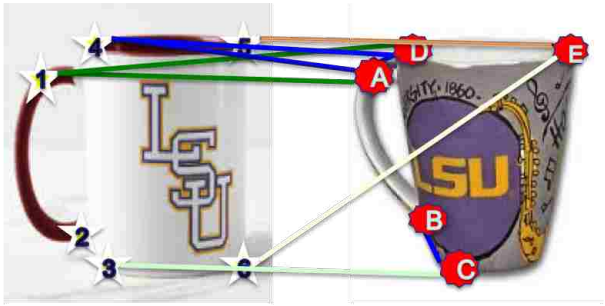



Figure 3.3: Locally matched features, initial matches, using KNN method.

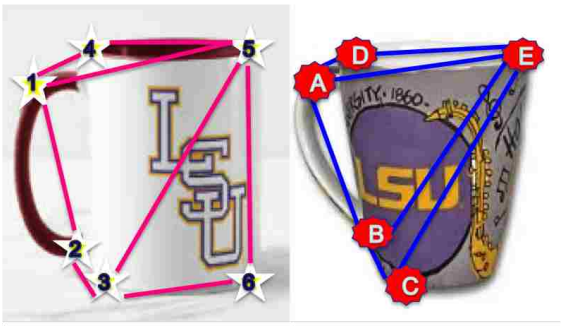



Figure 3.4: Graph representation using all the extracted features.

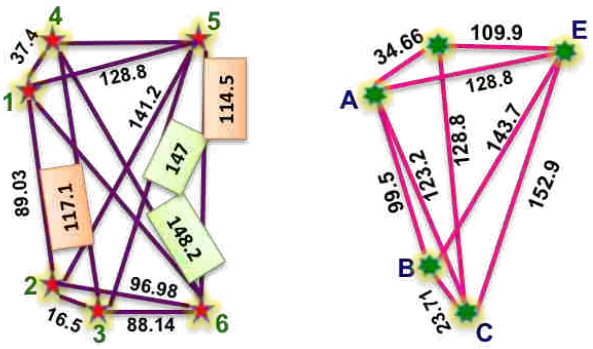
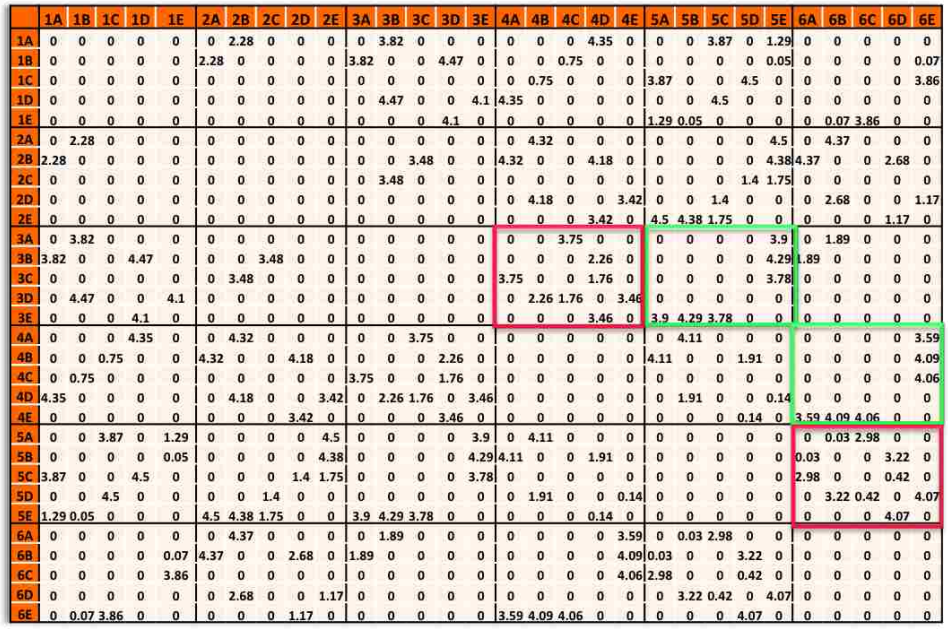



Figure 3.5: Graph construction of the two mugs images.

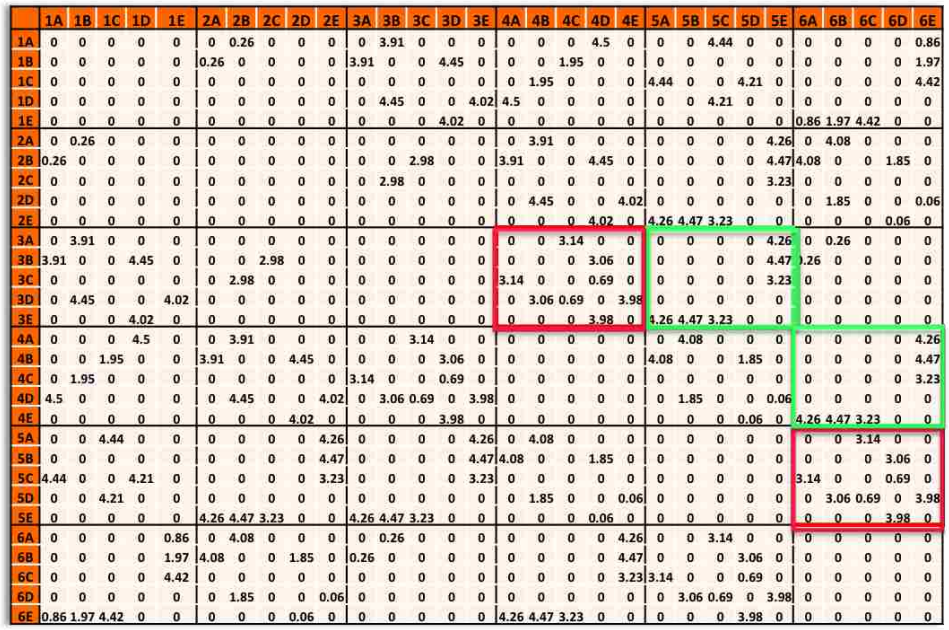
Constructing the affinity matrix using CGM method as it shown in Figure 3.6, we can see that the sub-matrices $\mathbf{M_{34}}$ and $\mathbf{M_{56}}$ values and also the sub-matrices $\mathbf{M_{35}}$ and $\mathbf{M_{46}}$ entries are quite similar. The CGM affinity matrix constructed using equation 2.11, resulted in total size of 900, using 6 features from image1 and 5 features from image2, 6x5x6x5. To construct the affinity matrix for AGM method the Equation 3.3 is employed, the pairwise distances of image1 first approximated using equation 3.2 by assigning the value of $w = 10$, and $\sigma = 5$ in this example. Since the distances of graph1 approximated, causing the the sub-matrices $\mathbf{M_{34}} = \mathbf{M_{56}}$and the sub-matrices $\mathbf{M_{35}} = \mathbf{M_{46}}$, therefore only one of the sub-matrices need to be constructed and stored. As it shown in Figure 3.7 using the approximated distances, the values of sub-matrices in red boarders are same, and also the values of the sub-matrices with green boarders. Therefore the total size of the AGM affinity would be 13 * 5 * 5 = 325, where 13 is the number of unique sub-matrices, and 5 number of nodes in graph2.

Figure 3.8 display the affinity build by PFGM approach using only the initial matches points, resulting in matrix size of 7 * 7 = 49, where 7 is number of initial matches. In our method, PBGM we first pruned the candidates by keeping those that are in initial matching list, therefore, our candidate assignments are {1, 3, 4, 5, 6} from graph1, then their approximated distances calculated and the sub-blocked matrices constructed for the



| | 1A | 1B | 1C | 1D | 1E | 2A | 2B | 2C | 2D | 2E | 3A | 3B | 3C | 3D | 3E | 4A | 4B | 4C | 4D | 4E | 5A | 5B | 5C | 5D | 5E | 6A | 6B | 6C | 6D | 6E |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1A | 0 | 0 | 0 | 0 | 0 | 0 | 2.28 | 0 | 0 | 0 | 0 | 3.82 | 0 | 0 | 0 | 0 | 0 | 0 | 4.35 | 0 | 0 | 0 | 3.87 | 0 | 1.29 | 0 | 0 | 0 | 0 | 0 |
| 1B | 0 | 0 | 0 | 0 | 0 | 2.28 | 0 | 0 | 0 | 0 | 3.82 | 0 | 0 | 4.47 | 0 | 0 | 0 | 0.75 | 0 | 0 | 0 | 0 | 0 | 0 | 0.05 | 0 | 0 | 0 | 0 | 0.07 |
| 1C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.75 | 0 | 0 | 0 | 3.87 | 0 | 0 | 4.5 | 0 | 0 | 0 | 0 | 0 | 3.86 |
| 1D | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.47 | 0 | 0 | 4.1 | 4.35 | 0 | 0 | 0 | 0 | 0 | 0 | 4.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.1 | 0 | 0 | 0 | 0 | 0 | 0 | 1.29 | 0.05 | 0 | 0 | 0 | 0 | 0.07 | 3.86 | 0 | 0 |
| 2A | 0 | 2.28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.5 | 0 | 4.37 | 0 | 0 | 0 |
| 2B | 2.28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.48 | 0 | 0 | 4.32 | 0 | 0 | 4.18 | 0 | 0 | 0 | 0 | 0 | 4.38 | 4.37 | 0 | 0 | 2.68 | 0 |
| 2C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.4 | 1.75 | 0 | 0 | 0 | 0 | 0 |
| 2D | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.18 | 0 | 0 | 3.42 | 0 | 0 | 1.4 | 0 | 0 | 0 | 2.68 | 0 | 0 | 1.17 |
| 2E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.42 | 0 | 4.5 | 4.38 | 1.75 | 0 | 0 | 0 | 0 | 0 | 1.17 | 0 |
| 3A | 0 | 3.82 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.75 | 0 | 0 | 0 | 0 | 0 | 0 | 3.9 | 0 | 1.89 | 0 | 0 | 0 |
| 3B | 3.82 | 0 | 0 | 4.47 | 0 | 0 | 0 | 3.48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2.26 | 0 | 0 | 0 | 0 | 0 | 4.29 | 1.89 | 0 | 0 | 0 | 0 |
| 3C | 0 | 0 | 0 | 0 | 0 | 0 | 3.48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.75 | 0 | 0 | 1.76 | 0 | 0 | 0 | 0 | 0 | 3.78 | 0 | 0 | 0 | 0 | 0 |
| 3D | 0 | 4.47 | 0 | 0 | 4.1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2.26 | 1.76 | 0 | 3.46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3E | 0 | 0 | 0 | 4.1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.46 | 0 | 3.9 | 4.29 | 3.78 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4A | 0 | 0 | 0 | 4.35 | 0 | 0 | 4.32 | 0 | 0 | 0 | 0 | 0 | 3.75 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.59 |
| 4B | 0 | 0 | 0.75 | 0 | 0 | 4.32 | 0 | 0 | 4.18 | 0 | 0 | 0 | 0 | 2.26 | 0 | 0 | 0 | 0 | 0 | 0 | 4.11 | 0 | 0 | 1.91 | 0 | 0 | 0 | 0 | 0 | 4.09 |
| 4C | 0 | 0.75 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.75 | 0 | 0 | 1.76 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.06 |
| 4D | 4.35 | 0 | 0 | 0 | 0 | 0 | 4.18 | 0 | 0 | 3.42 | 0 | 2.26 | 1.76 | 0 | 3.46 | 0 | 0 | 0 | 0 | 0 | 0 | 1.91 | 0 | 0 | 0.14 | 0 | 0 | 0 | 0 | 0 |
| 4E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.42 | 0 | 0 | 0 | 0 | 3.46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.14 | 0 | 3.59 | 4.09 | 4.06 | 0 | 0 |
| 5A | 0 | 0 | 3.87 | 0 | 1.29 | 0 | 0 | 0 | 0 | 4.5 | 0 | 0 | 0 | 0 | 3.9 | 0 | 4.11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.03 | 2.98 | 0 | 0 |
| 5B | 0 | 0 | 0 | 0 | 0.05 | 0 | 0 | 0 | 0 | 4.38 | 0 | 0 | 0 | 0 | 4.29 | 4.11 | 0 | 0 | 1.91 | 0 | 0 | 0 | 0 | 0 | 0 | 0.03 | 0 | 0 | 3.22 | 0 |
| 5C | 3.87 | 0 | 0 | 4.5 | 0 | 0 | 0 | 0 | 1.4 | 1.75 | 0 | 0 | 0 | 0 | 3.78 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2.98 | 0 | 0 | 0.42 | 0 |
| 5D | 0 | 0 | 4.5 | 0 | 0 | 0 | 0 | 1.4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.91 | 0 | 0 | 0.14 | 0 | 0 | 0 | 0 | 0 | 0 | 3.22 | 0.42 | 0 | 4.07 |
| 5E | 1.29 | 0.05 | 0 | 0 | 0 | 4.5 | 4.38 | 1.75 | 0 | 0 | 3.9 | 4.29 | 3.78 | 0 | 0 | 0 | 0 | 0 | 0.14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.07 | 0 |
| 6A | 0 | 0 | 0 | 0 | 0 | 0 | 4.37 | 0 | 0 | 0 | 0 | 1.89 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.59 | 0 | 0.03 | 2.98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6B | 0 | 0 | 0 | 0 | 0.07 | 4.37 | 0 | 0 | 2.68 | 0 | 1.89 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.09 | 0.03 | 0 | 0 | 3.22 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6C | 0 | 0 | 0 | 0 | 3.86 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.06 | 2.98 | 0 | 0 | 0.42 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6D | 0 | 0 | 0 | 0 | 0 | 0 | 2.68 | 0 | 0 | 1.17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.22 | 0.42 | 0 | 4.07 | 0 | 0 | 0 | 0 | 0 |
| 6E | 0 | 0.07 | 3.86 | 0 | 0 | 0 | 0 | 0 | 1.17 | 0 | 0 | 0 | 0 | 0 | 0 | 3.59 | 4.09 | 4.06 | 0 | 0 | 0 | 0 | 0 | 4.07 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 3.6: CGM Affinity matrix



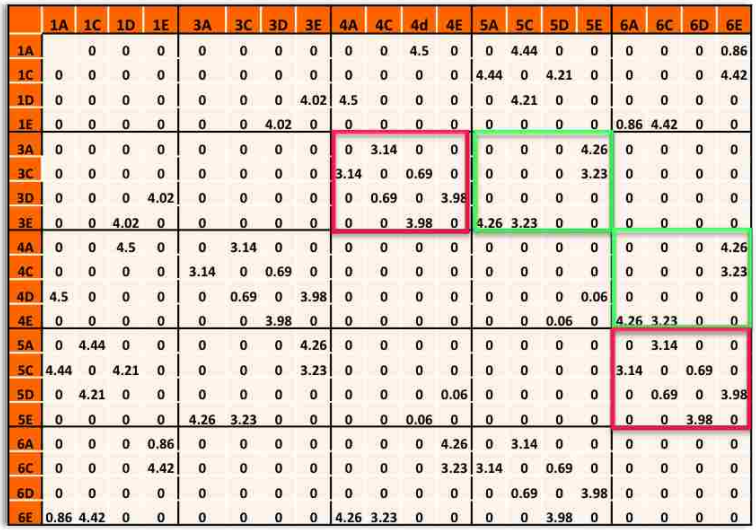
| | 1A | 1B | 1C | 1D | 1E | 2A | 2B | 2C | 2D | 2E | 3A | 3B | 3C | 3D | 3E | 4A | 4B | 4C | 4D | 4E | 5A | 5B | 5C | 5D | 5E | 6A | 6B | 6C | 6D | 6E |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1A | 0 | 0 | 0 | 0 | 0 | 0 | 0.26 | 0 | 0 | 0 | 0 | 3.91 | 0 | 0 | 0 | 0 | 0 | 0 | 4.5 | 0 | 0 | 0 | 4.44 | 0 | 0 | 0 | 0 | 0 | 0 | 0.86 |
| 1B | 0 | 0 | 0 | 0 | 0 | 0.26 | 0 | 0 | 0 | 0 | 3.91 | 0 | 0 | 4.45 | 0 | 0 | 0 | 1.95 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.97 |
| 1C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.95 | 0 | 0 | 0 | 4.44 | 0 | 0 | 4.21 | 0 | 0 | 0 | 0 | 0 | 4.42 |
| 1D | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.45 | 0 | 0 | 4.02 | 4.5 | 0 | 0 | 0 | 0 | 0 | 0 | 4.21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.02 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.86 | 1.97 | 4.42 | 0 | 0 |
| 2A | 0 | 0.26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.91 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.26 | 0 | 4.08 | 0 | 0 | 0 |
| 2B | 0.26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2.98 | 0 | 0 | 3.91 | 0 | 0 | 4.45 | 0 | 0 | 0 | 0 | 0 | 4.47 | 4.08 | 0 | 0 | 1.85 | 0 |
| 2C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2.98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.23 | 0 | 0 | 0 | 0 | 0 |
| 2D | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.45 | 0 | 0 | 4.02 | 0 | 0 | 0 | 0 | 0 | 0 | 1.85 | 0 | 0 | 0.06 |
| 2E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.02 | 0 | 4.26 | 4.47 | 3.23 | 0 | 0 | 0 | 0 | 0 | 0.06 | 0 |
| 3A | 0 | 3.91 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0 | 0 | 0 | 0 | 0 | 4.26 | 0 | 0.26 | 0 | 0 | 0 |
| 3B | 3.91 | 0 | 0 | 4.45 | 0 | 0 | 0 | 2.98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.06 | 0 | 0 | 0 | 0 | 0 | 4.47 | 0.26 | 0 | 0 | 0 | 0 |
| 3C | 0 | 0 | 0 | 0 | 0 | 0 | 2.98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0 | 0.69 | 0 | 0 | 0 | 0 | 0 | 3.23 | 0 | 0 | 0 | 0 | 0 |
| 3D | 0 | 4.45 | 0 | 0 | 4.02 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.06 | 0.69 | 0 | 3.98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3E | 0 | 0 | 0 | 4.02 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.98 | 0 | 4.26 | 4.47 | 3.23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4A | 0 | 0 | 0 | 4.5 | 0 | 0 | 3.91 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.08 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.26 |
| 4B | 0 | 0 | 1.95 | 0 | 0 | 3.91 | 0 | 0 | 4.45 | 0 | 0 | 0 | 0 | 3.06 | 0 | 0 | 0 | 0 | 0 | 0 | 4.08 | 0 | 0 | 1.85 | 0 | 0 | 0 | 0 | 0 | 4.47 |
| 4C | 0 | 1.95 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0 | 0.69 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.23 |
| 4D | 4.5 | 0 | 0 | 0 | 0 | 0 | 4.45 | 0 | 0 | 4.02 | 0 | 3.06 | 0.69 | 0 | 3.98 | 0 | 0 | 0 | 0 | 0 | 0 | 1.85 | 0 | 0 | 0.06 | 0 | 0 | 0 | 0 | 0 |
| 4E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.02 | 0 | 0 | 0 | 0 | 3.98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.06 | 0 | 4.26 | 4.47 | 3.23 | 0 | 0 |
| 5A | 0 | 0 | 4.44 | 0 | 0 | 0 | 0 | 0 | 0 | 4.26 | 0 | 0 | 0 | 0 | 4.26 | 0 | 4.08 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0 |
| 5B | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.47 | 0 | 0 | 0 | 0 | 4.47 | 4.08 | 0 | 0 | 1.85 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.06 | 0 |
| 5C | 4.44 | 0 | 0 | 4.21 | 0 | 0 | 0 | 0 | 0 | 3.23 | 0 | 0 | 0 | 0 | 3.23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0 | 0.69 | 0 |
| 5D | 0 | 0 | 4.21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.85 | 0 | 0 | 0.06 | 0 | 0 | 0 | 0 | 0 | 0 | 3.06 | 0.69 | 0 | 3.98 |
| 5E | 0 | 0 | 0 | 0 | 0 | 4.26 | 4.47 | 3.23 | 0 | 0 | 4.26 | 4.47 | 3.23 | 0 | 0 | 0 | 0 | 0 | 0.06 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.98 | 0 |
| 6A | 0 | 0 | 0 | 0 | 0.86 | 0 | 4.08 | 0 | 0 | 0 | 0 | 0.26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.26 | 0 | 0 | 3.14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6B | 0 | 0 | 0 | 0 | 1.97 | 4.08 | 0 | 0 | 1.85 | 0 | 0.26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.47 | 0 | 0 | 0 | 3.06 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6C | 0 | 0 | 0 | 0 | 4.42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.23 | 3.14 | 0 | 0 | 0.69 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6D | 0 | 0 | 0 | 0 | 0 | 0 | 1.85 | 0 | 0 | 0.06 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.06 | 0.69 | 0 | 3.98 | 0 | 0 | 0 | 0 | 0 |
| 6E | 0.86 | 1.97 | 4.42 | 0 | 0 | 0 | 0 | 0 | 0.06 | 0 | 0 | 0 | 0 | 0 | 0 | 4.26 | 4.47 | 3.23 | 0 | 0 | 0 | 0 | 0 | 3.98 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 3.7: Affinity matrix using the approximate pairwise distances.

| | 1A | 1D | 3C | 4A | 4D | 5E | 6E |
|---|---|---|---|---|---|---|---|
| 1A | 0 | 0 | 0 | 0 | 4.35 | 1.29 | 0 |
| 1D | 0 | 0 | 0 | 4.35 | 0 | 0 | 0 |
| 3C | 0 | 0 | 0 | 3.75 | 1.76 | 3.78 | 0 |
| 4A | 0 | 4.35 | 3.75 | 0 | 0 | 0 | 3.59 |
| 4D | 4.35 | 0 | 1.76 | 0 | 0 | 0.14 | 0 |
| 5E | 1.29 | 0 | 3.78 | 0 | 0.14 | 0 | 0 |
| 6E | 0 | 0 | 0 | 3.59 | 0 | 0 | 0 |

Figure 3.8: PFGM Affinity matrix

unique distances. The whole affinity matrix using the pruned assignments is shown in Figure 3.9, and sub-blocked matrices with same values are displayed with red and green boarders. Since, we only want to construct and store one representation of the similar sub-matrices, their values need to be same. Therefore, for building sum-matrices using nodes 3 and 4 not only their matching correspondences in the initial list needed to be found, but also nodes 5 and 6 matching assignments needed to be found in order to make the same sub-matrices. Thus, the matched points of nodes 3, 4, 5, 6 in the initial matching list selected which are {C, A, D, E, E} and their sub-blocked matrices constructed using those points. In our experiments we noticed that since their are many nodes with same pairwise distances the corresponding correspondences that are selected from initial matches would be same. Resulting in employing all the nodes that are presented in the initial matches. Therefore, all the blocks have a same size and their affinities needed to be calculated in order to have similar sub-blocked matrices. In this example using the PBGM method the total affinities would be 6 * 4 * 4 = 96, where 6 is the non zero unique valued sub-blocked, and 4 is the number of the pruned nodes of graph2. We can see in this example that the size of memory to construct the affinities reduced from 900 using the first method to 96 in our method. Our experiments also presented that the accuracy we got from our method compared to PFGM increased.



| | 1A | 1C | 1D | 1E | 3A | 3C | 3D | 3E | 4A | 4C | 4d | 4E | 5A | 5C | 5D | 5E | 6A | 6C | 6D | 6E |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1A | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.5 | 0 | 0 | 4.44 | 0 | 0 | 0 | 0 | 0 | 0.86 |
| 1C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.44 | 0 | 4.21 | 0 | 0 | 0 | 0 | 4.42 |
| 1D | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.02 | 4.5 | 0 | 0 | 0 | 0 | 4.21 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1E | 0 | 0 | 0 | 0 | 0 | 0 | 4.02 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.86 | 4.42 | 0 | 0 |
| 3A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0 | 0 | 0 | 0 | 4.26 | 0 | 0 | 0 | 0 |
| 3C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0.69 | 0 | 0 | 0 | 0 | 3.23 | 0 | 0 | 0 | 0 |
| 3D | 0 | 0 | 0 | 4.02 | 0 | 0 | 0 | 0 | 0 | 0.69 | 0 | 3.98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3E | 0 | 0 | 4.02 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.98 | 0 | 4.26 | 3.23 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4A | 0 | 0 | 4.5 | 0 | 0 | 3.14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.26 |
| 4C | 0 | 0 | 0 | 0 | 3.14 | 0 | 0.69 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.23 |
| 4D | 4.5 | 0 | 0 | 0 | 0 | 0.69 | 0 | 3.98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.06 | 0 | 0 | 0 | 0 |
| 4E | 0 | 0 | 0 | 0 | 0 | 0 | 3.98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.06 | 0 | 4.26 | 3.23 | 0 | 0 |
| 5A | 0 | 4.44 | 0 | 0 | 0 | 0 | 0 | 4.26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0 |
| 5C | 4.44 | 0 | 4.21 | 0 | 0 | 0 | 0 | 3.23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.14 | 0 | 0.69 | 0 |
| 5D | 0 | 4.21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.06 | 0 | 0 | 0 | 0 | 0 | 0.69 | 0 | 3.98 |
| 5E | 0 | 0 | 0 | 0 | 4.26 | 3.23 | 0 | 0 | 0 | 0 | 0.06 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.98 | 0 |
| 6A | 0 | 0 | 0 | 0.86 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.26 | 0 | 3.14 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6C | 0 | 0 | 0 | 4.42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.23 | 3.14 | 0 | 0.69 | 0 | 0 | 0 | 0 | 0 |
| 6D | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.69 | 0 | 3.98 | 0 | 0 | 0 | 0 |
| 6E | 0.86 | 4.42 | 0 | 0 | 0 | 0 | 0 | 0 | 4.26 | 3.23 | 0 | 0 | 0 | 0 | 3.98 | 0 | 0 | 0 | 0 | 0 |

Figure 3.9: Affinity matrix constructed using pruned assignments.

### 3.2.1 Bijectivity Constraint

The one to one bijectivity constraint need to be enforced to the solution for getting the correct correspondences. This two-way constraint is a sparse matrix consist of 0s and 1s. It can be formulated as a linear constraint $Ax \leq b$, where $A$ is the constraint matrix. On each row of $A$, the non-zero elements give the indices of the assignments that are associated with a same keypoint in one frame. Therefore, this matrix indicate the assignments that have mutual conflicts. No more than one among them should be selected in the final solution, as indicated by the vector $b = (1, 1, ..., 1)^T$. For instance, the assignments$(i, j)$, $(i', j)$, and $(i, j')$ are in conflict with each other therefore only one of them can be selected in the final solution. In our work the constraint matrix applied to the affinity matrix before the eigenvector computation to remove the assignments that are in conflict in order to improve the final solution. And, the greedy algorithm is used at the binarization step to enforce the bijectivity constraint as it explained in Algorithm 1.

### 3.2.2 Computing Eigenvector

After the affinity matrices constructed using different approaches, and the conflicted nodes been removed, the spectral matching algorithm is applied as explained in the previous chapter. The $x_d^*$ solved by multiplying the matrix with a randomly initialized vector repeatedly

until the normalized vector converged. The principal eigenvalue and its corresponding eigenvector first computed, denoted as $x^*$, then elements in $x^*$ sorted, the greatest element, $g$, and set $x_d^*(g) = 1$. Then, iteratively, following the descending order, all elements in $x^*$ that do not have conflict with existing marked elements selected, and assigned to *1* , while marking the indicator of each conflict element to *0*. The converged vector is the principal eigenvector of the matrix. In our method since each affinity can be constructed separately they can also multiplied by the eigenvector separately, which are the great candidate for parallelization. Thrust is used to construct each of the blocks in parallel. And, the CPU's and GPU's accelerated libraries MKL, cuBlas, cuSpars for matrix vector multiplication.

### 3.3 Result - Implementation

The implementation of our work is as follows: we first extracted feature points from images using MSER, we also used SIFT in some of our example to get more discriminative points. For find the locally matching points KNN used to find k nearest neighbor for each feature in the first image. If the distances between the feature points and the neighbor points bigger than some threshold then those features would not be included in our candidate list. K determined the maximum number of matches each feature can have. The value of K set to 3 for PBGM method and 10 for the other algorithms, CGM, AGM, and PFGM. Using the distance of 128-dim SIFT descriptor, all the possible candidate matches selected if the feature pair has a closer distance in SIFT feature space than a threshold of $\delta = 0.8$, allowing multiple correspondences for each feature. The Maximum number of initial matches set to 3000.After the initial matches points were determined, distances between every points for each graph calculated. The approximated distances calculated in our example using 10 some example 5 for $w$ depends on the range of the distances, if the difference of the maximum and minimum distances values were larger then we set $w$ to 10, otherwise used 5. We implemented the matrix and vector class and the CSR and COO sparse version of

the matrix also constructed. Thrust library enabled us to provide a unified interface for both CPU and GPU. Depends on type of images and the size of them different approaches can be selected to get the best performance. For noisy type of images the PBGM works better, and for images that features description is good PFGM can be used. As it displayed in result the CGM algorithm could not be used in most of the examples, because of the amount of memory needed to construct the full affinity matrix. But, if the size of images be very small and have a good description, but the ambiguity be high then user can be benefited by using CGM method. Therefore, by having a unified interface best methods can be selected depending on the input images. The following result compared the four approaches in term of accuracy, memory and speed time. Result also compared to the original approach without using any optimization.

The ground truths (GT) manually specified for all candidate correspondences of each image pair, and the accuracy using both Recall and Precision Equation 3.4 computed for all the different methods and compared with each other. In the images where the homography transformation were available the accuracy also estimated using reprojection error. Having set of points $P$, the points$p_i$projected on the second image using it's homography. Then the euclidean distance between the reprojected points $p_i^{'}$and the second image points $q_j$calculated, if the distance of those two points are smaller than some threshold then those correspondence are considered as correct $\left\|p_i^{'} - q_j\right\| \prec \epsilon$.

$$
\begin{aligned}
&Precision = correct\ one\,(T_P)/all\,the\,GT \\
&Recall = correct\ one\,(T_P)/all\,the\,chosen\,one\,(T_P + F_P)
\end{aligned}
\tag{3.4}
$$

Thrust library used in the overall implementation of all the four methods, CGM, AGM, PFGM, and PBGM to provide a unified interface for both CPU and GPU. Each blocked affinity sub-matrices constructed in parallel using Thrsut library. The MKL, cuBlas and

cuSparse library applied for the matrix-vector multiplication for computing the principal eigenvector. As shown in the examples, the experiment and the dataset are designed for producing the challenging feature matching problems where unary local features are very ambiguous. Our PBGM clearly outperforms other methods both in accuracy and execution time.

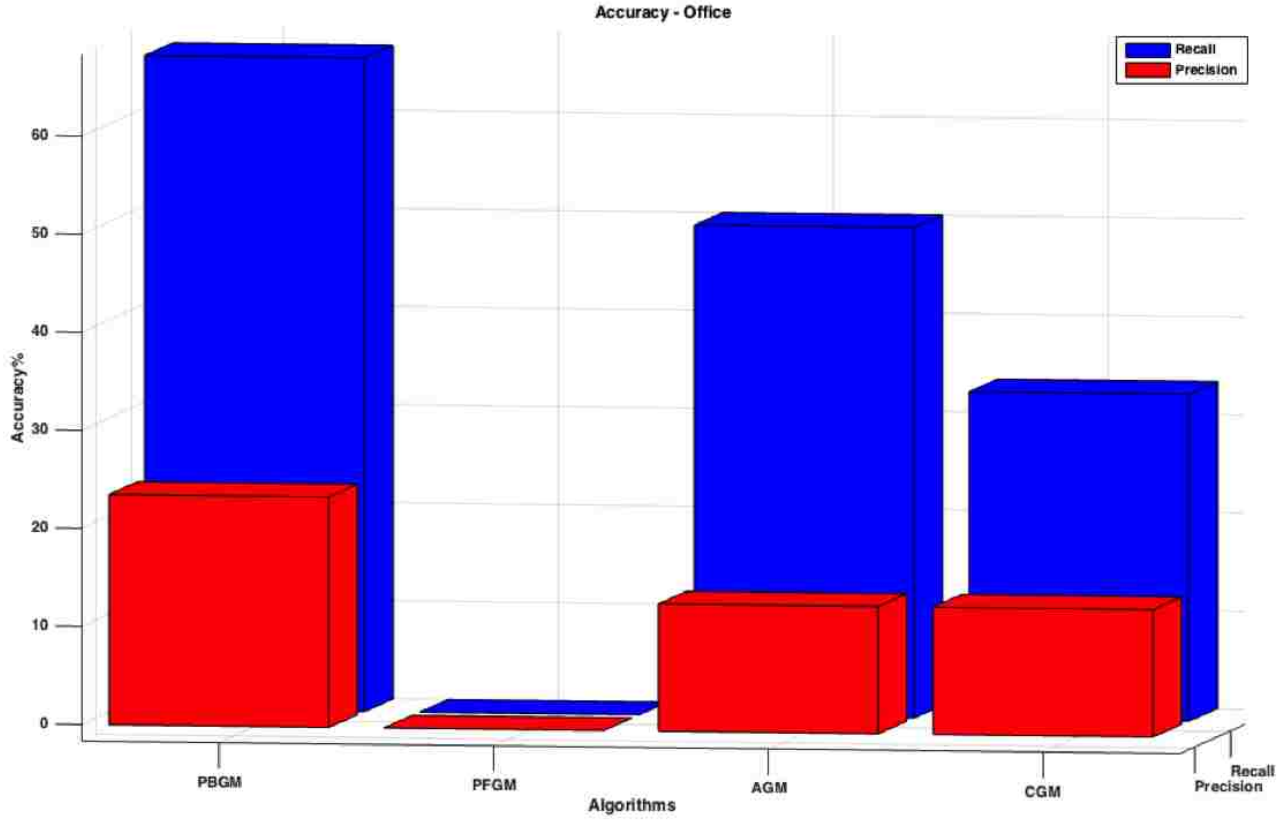
### 3.3.1 Accuracy

The accuracy of each algorithm estimated using recall and precision methods. As it can seen from the result our methods works better when there are ambiguity and they are noisy. The final correspondences also displayed using the PBGM method. In each example the bar plot shows the accuracy of different algorithms, and real images display the final correspondences. True matches are represented by pink lines, and false matches by black lines. The accuracy result of the different method for each pair of images using Recall and Percentage, also summarized in the table of Figure 3.10.

Figure 3.10 shows that using PBGM increase the accuracy by 66.66% compare to PFGM, and 50% and 26.99% compare to CGM and AGM respectively. The images that used in this example are depth data taken with Kinect v2. Since, the images are not discriminative our method PBGM resulted in a better performance than the other methods. We got 66.66% accuracy using recall and 23.5% using precision.

It can be seen from Figure 3.12 that using PBGM increase the accuracy value by 29.97% compare to PFGM, and 12.27%compare to CGM and AGM. The images that used in this example have homography transformation between them. Input data contain repeated pattern that cause the PFGM score to be lower than the other algorithm. As, it is shown in the table 3.12 the accuracy obtained from using our method is 95.74%.

Figure 3.14 shows that using PBGM increase the accuracy by 50.99% compare to PFGM,




| Accuracy \Methods | Recall | Precision |
|---|---|---|
| PBGM | 66.66% | 23.5% |
| PFGM | 0% | 0% |
| AGM | 50% | 7% |
| CGM | 33.33% | 7% |

Figure 3.10: Recall and Precision accuracy estimation

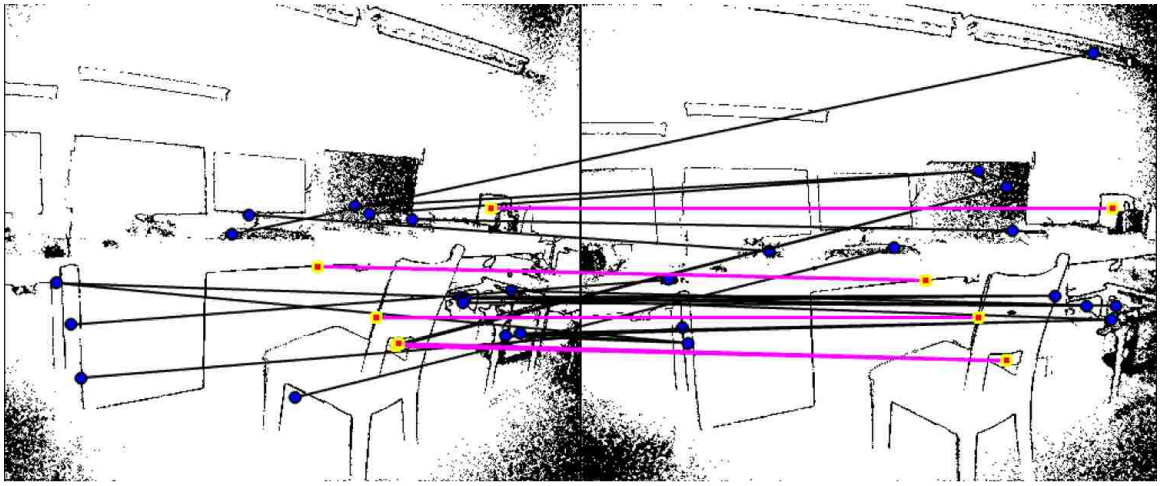


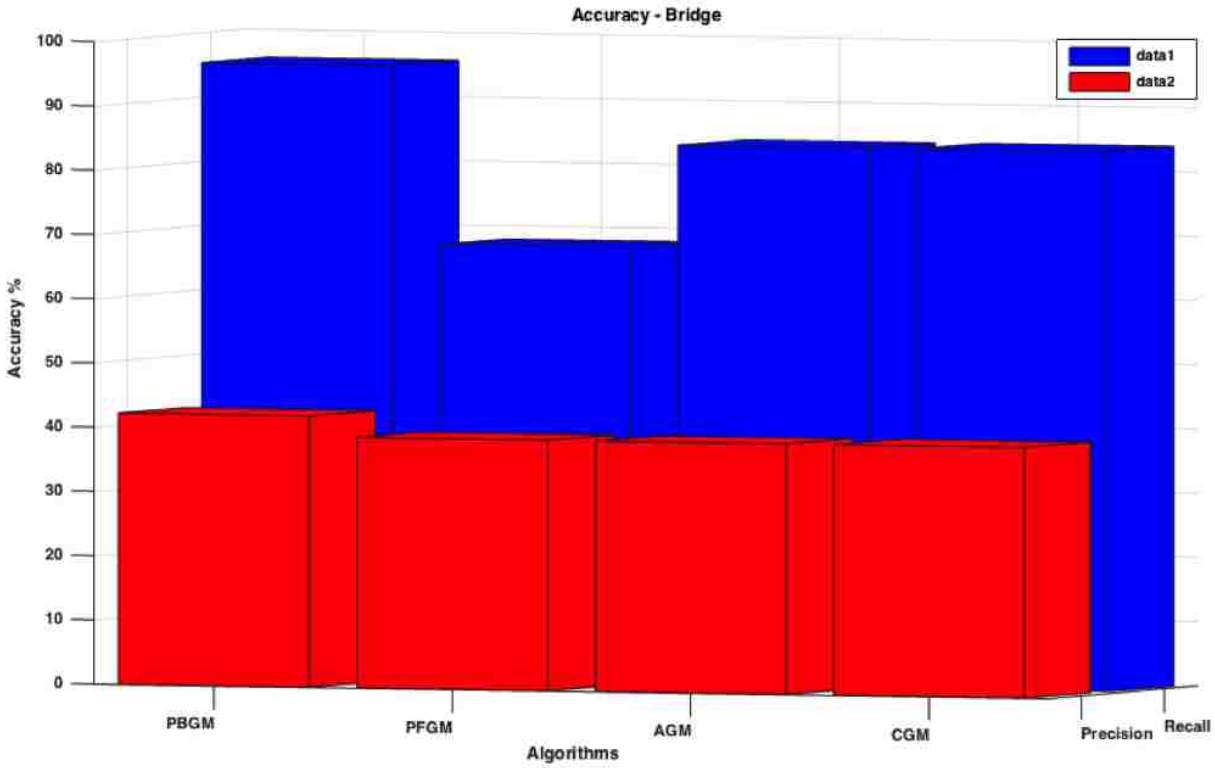
Figure 3.11: Final correspondences

and 24.25% and 27.75% compare to CGM and AGM respectively. Gaussian noise added to input images in this example. Since, the images are noisy the features extracted are not discriminative, therefore, our method PBGM resulted in a better performance than the PFGM. The CGM, and AGM algorithms score is higher because more candidates used and their geometric relationship help to increase the accuracy. The table shows the accuracy score obtained from all the methods using Recall and Precision.

Figure 3.17 display images taken with affine transformation between them. Result from Figure 3.16 show the accuracy estimated for all the different algorithms. Since, input images contain no noise and ambiguity the result obtained from PFGM is higher than other methods, but not much from our method PBGM. Our result is quite comparable to PFGM, using Recall, Precision and Reprojection error.

### 3.3.2 Execution Time

Performances of the four algorithms (CGM, AGM, PFGM, PBGM) using CPU and GPU-accelerated libraries (MKL, cuBlas, cuSparse) displayed in the following figures. Three different examples are used to visualize the execution time of all the methods, start from




| Accuracy \Methods | Recall | Precision | Homography |
|---|---|---|---|
| PBGM | 95.74% | 42.23% | 78.23% |
| PFGM | 68% | 39% | 75.51% |
| AGM | 84% | 39% | 67.86% |
| CGM | 84% | 39% | 51.09% |

Figure 3.12: Recall and Precision accuracy estimation

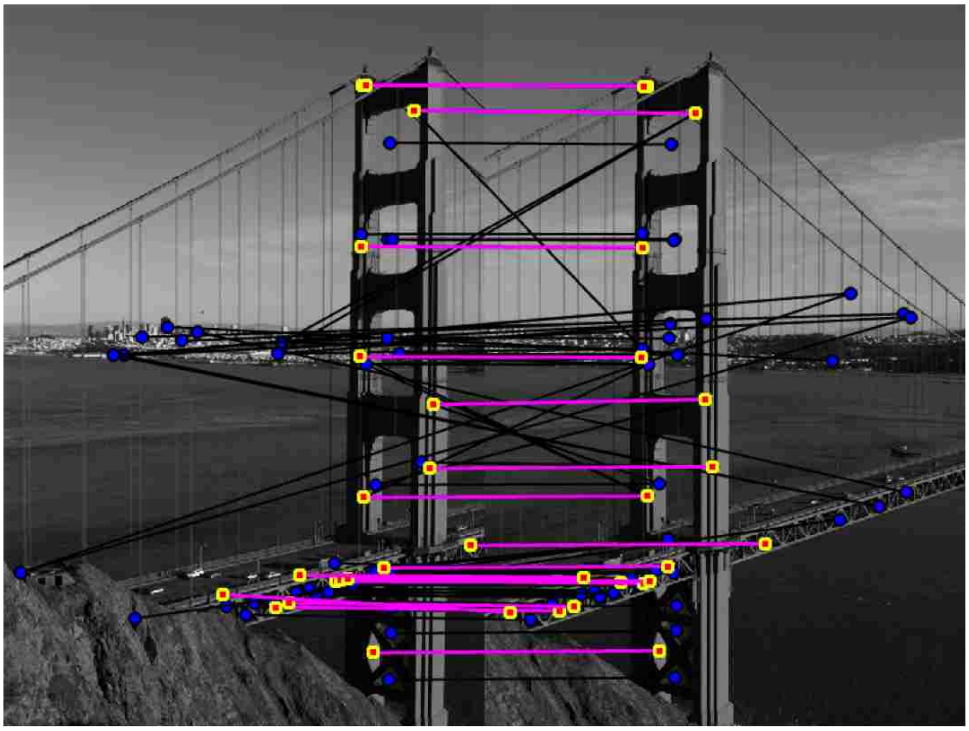
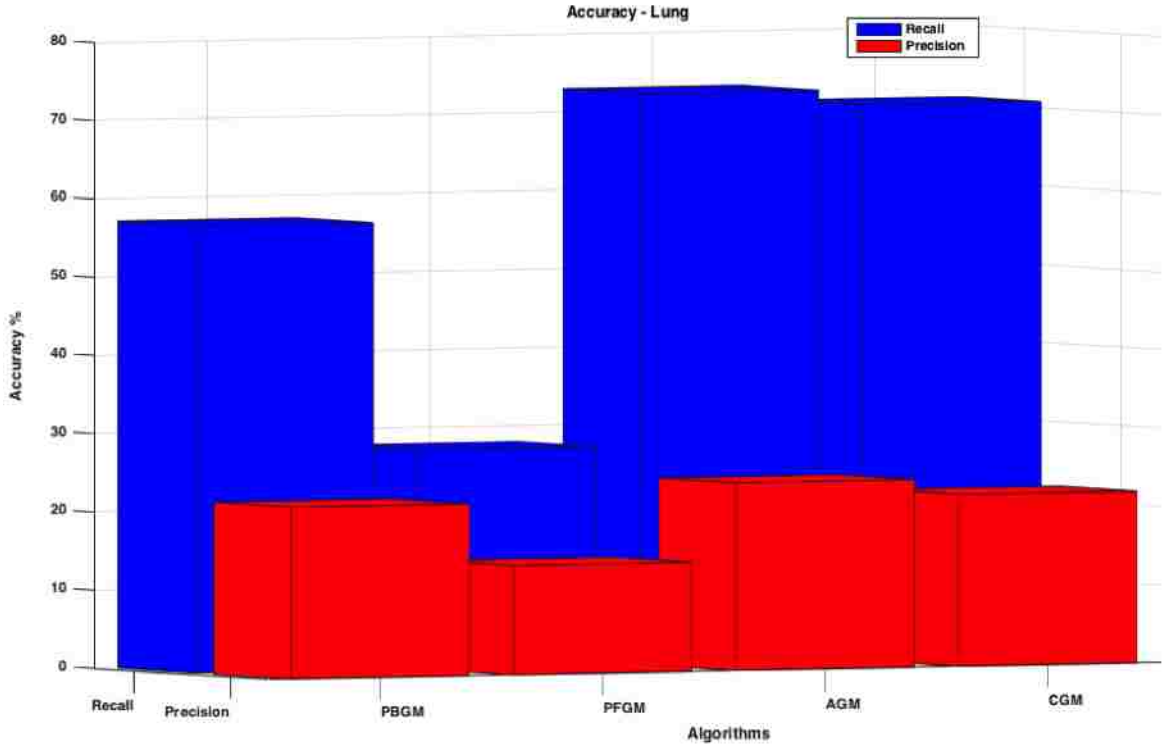


Figure 3.13: Final correspondences

smallest data size. As it can be seen in Figure 3.19and Figure **??**, the execution time are not reported using the CGM method as the affinity matrix did not fit in the memory for the larger input data.

In the first example the total size for constructing the whole affinity matrix is $3.93 * 10^8$. Using our method PBGM by feature pruning and redundancy pattern suppression technique the memory requirement significantly reduced by 99.58. Comparing to the original method without any optimization our method showed the massive 93.43% speed up. Using GPU libray, cuBlas the speed up we got compared to MKL is only 20.45% since transferring data to GPU added overhead, and therefore increased the execution time. Also, sparsifying the matrix, using Compressed Sparse Row (CSR) caused a massive slowdown.

In this example Figure 3.19 the total size for constructing the whole affinity matrix is $86.5 * 10^9$. As it mentioned earlier since the data got bigger constructing the whole affinity matrix using CGM was not possible because of huge memory footprint of constructing the affinity . Therefore only the result of other method presented here. Using PBGM




| Accuracy \Methods | Recall | Precision |
|---|---|---|
| PBGM | 57.14% | 22% |
| PFGM | 28% | 14% |
| AGM | 73% | 24% |
| CGM | 71% | 22% |

Figure 3.14: Recall and Precision accuracy estimation

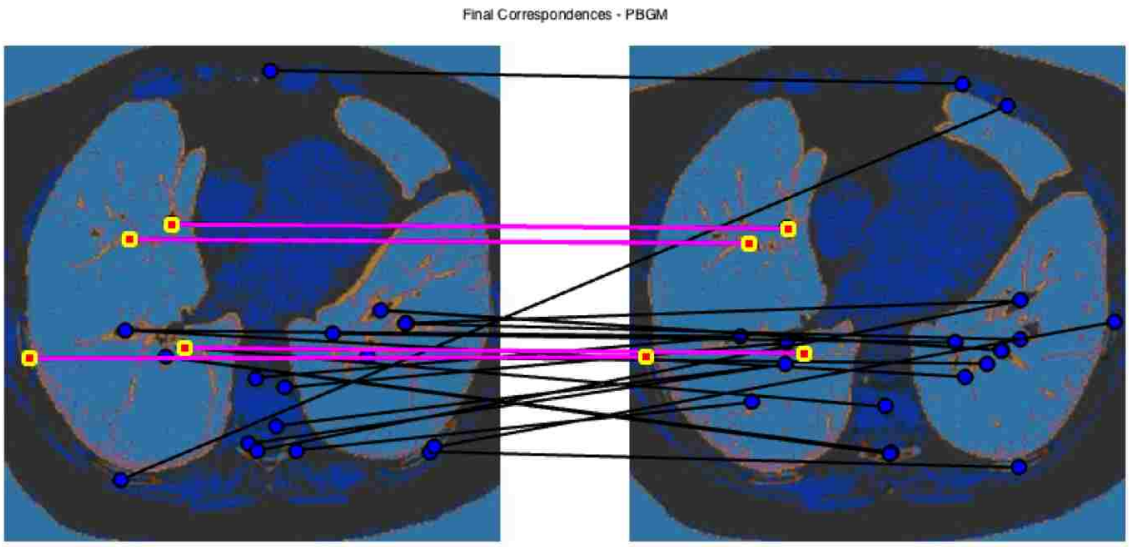
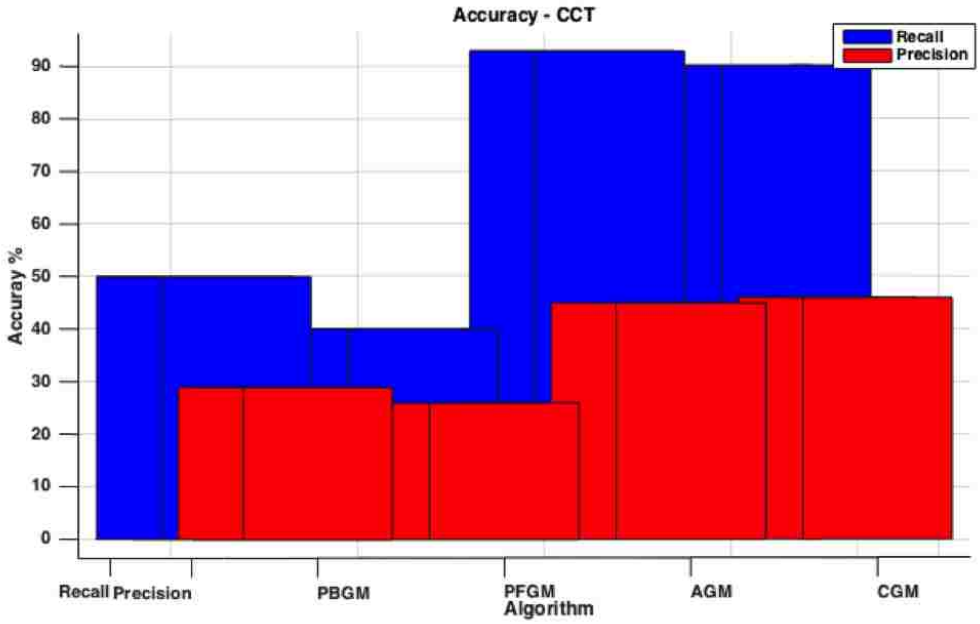



Figure 3.15: Final correspondences




| Accuracy \Methods | Recall | Precision | Homography |
|---|---|---|---|
| PBGM | 90.06% | 46% | 96.66% |
| PFGM | 92.85% | 45% | 98.45% |
| AGM | 40% | 26% | 50% |
| CGM | 50% | 29% | 58.09% |

Figure 3.16: Recall and Precision accuracy estimation

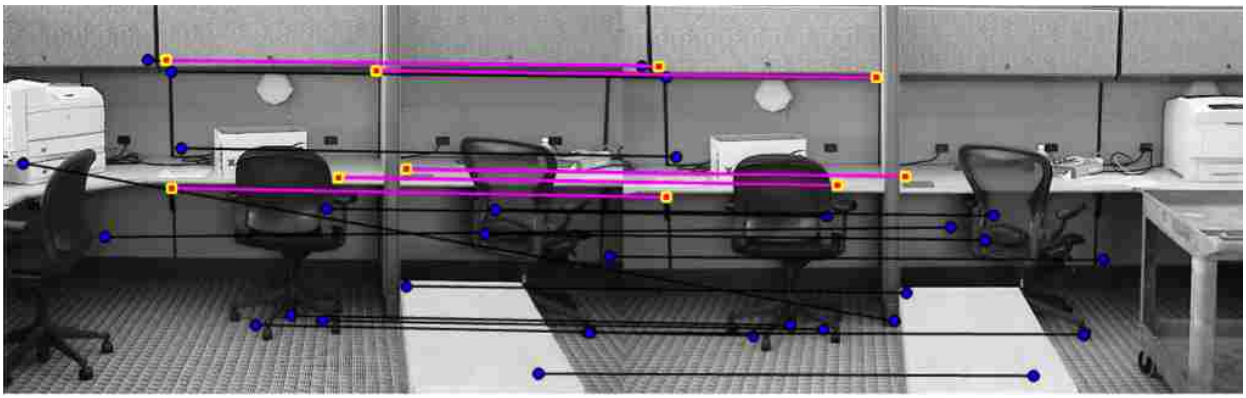


Figure 3.17: Final correspondences

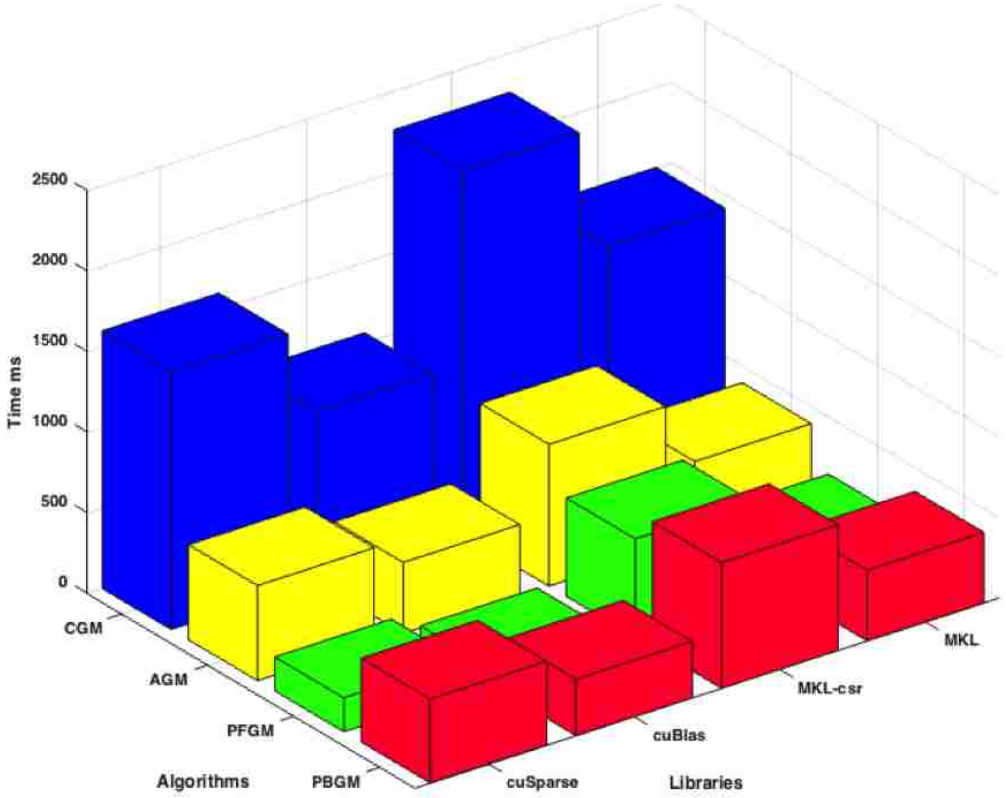



Figure 3.18: Execution time using different algorithms and libraries with size: $3.93 * 10^8$.

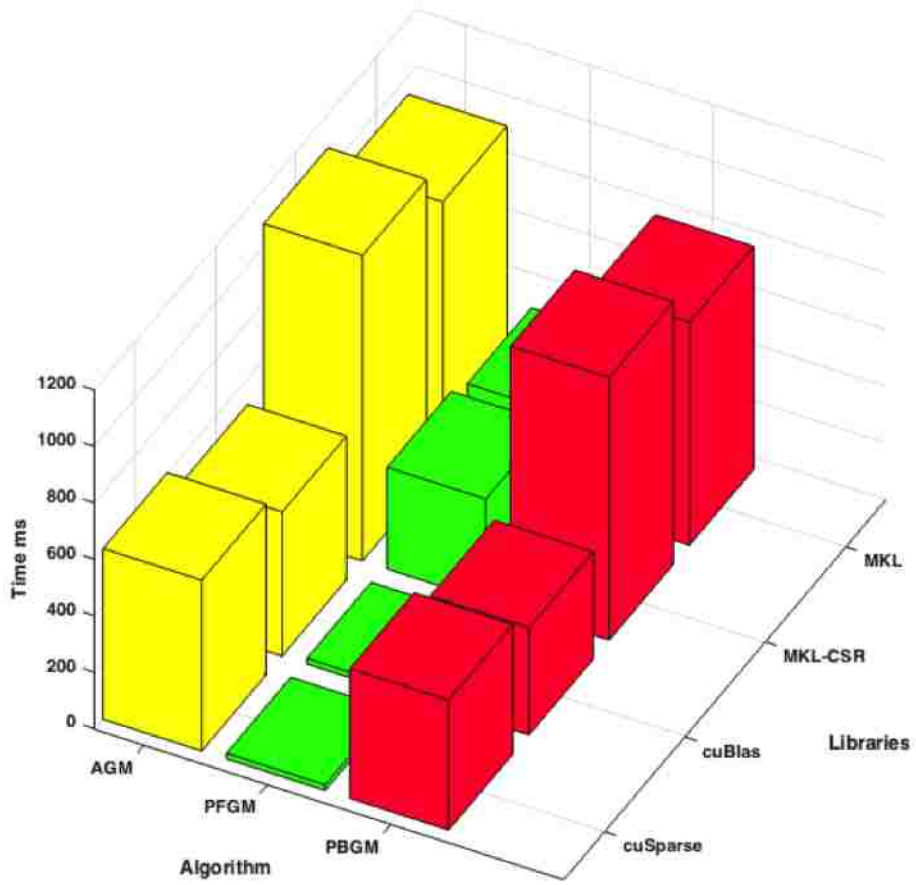



Figure 3.19: Execution time using different algorithms and libraries with size: $86.5 * 10^9$

method the amount of memory required to store the blocked affinity sub_matrices reduced by 99.98. Also, as the data size grew and the amount of computation increased, we got 51.89% of speed up using cuBlas comparing to MKL.

In the third example Figure **??** the total size for constructing the whole affinity matrix is $3.98 * 10^{12}$. As it mentioned earlier since the data got bigger constructing the whole affinity matrix using CGM was not possible because of huge memory footprint of constructing the affinity . Therefore only the result of other method presented here. The memory space needed to store the blocked affinity sub_matrices reduced by 99.99. As the input size gets larger the sparsed version, using cuSparse, the computation run faster with speed up of $51.59\%$ comparing to MKL, and the result from cuBlas showed the speed up of 39.89%.

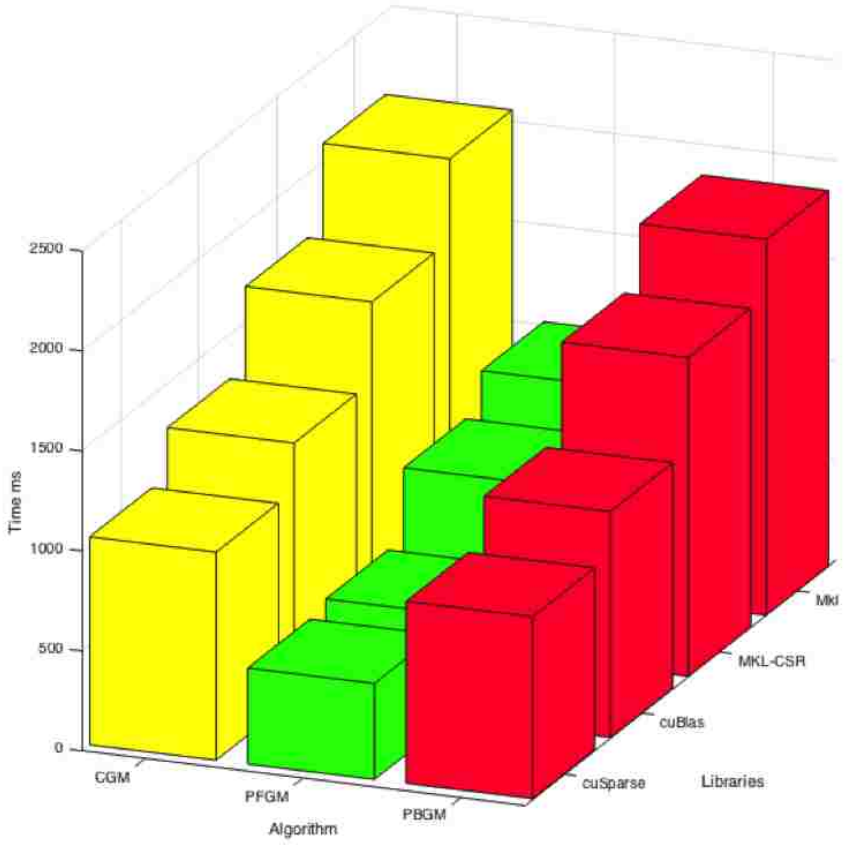



Figure 3.20: Execution time using different algorithms and libraries with size: $3.98 * 10^{12}$

# Chapter 4
# MULTI IMAGE MATCHING OPTIMIZATION AND PARALLELIZATION

## 4.1 Introduction

Finding feature correspondences between two images is a fundamental problem in computer vision with various applications such as structure image registration, shape analysis, motion. While previous efforts focused mostly on matching a pair of pictures, many tasks require finding correspondences across multiple images. Such as matching and stitching partially overlapped point clouds sequentially scanned from a 3D scene, and reconstruct the entire scene which is a fundamental problem in computer graphics, and robotics. Simultaneous Localization And Mapping (SLAM) is one the application where a robot equipped with a range scanning sensor can navigate around an unknown environment to reconstruct the surrounding and locate its own position. Due to the advance of modern imaging and scanning technologies, multi-graph matching is applied to infusing multi-source sensor data, multi-view assembly, and multi-source topic alignment. It is also related to graph clustering, classification, and indexing. In many applications, visual objects do not appear in isolation or in pair, but more frequently in collections or families. Also, considering more graph instead of two improves the matching accuracy, since it is required to find global consistent correspondence by using the information of pairwise affinity of all graph. Such an improvement is accomplished through avoiding trapping to local optima, due to the large deformation, appearing in the pairwise case since only affinity between two local graphs is explored Yan et al. (2013). Therefore, it is appealing to design effective and efficient multi-graph matching algorithms beyond conventional pairwise matching solvers

Yan et al. (2014). One way of matching the multi-graph could be solving it in a sequential manner Pevzner (1992), where each step executes a pairwise matching of two sequential graphs. And, it could be designed by different orders to cover all graphs in a path, e.g., $G1 \rightarrow G2 \rightarrow \cdots \rightarrow GN$. However, no matter in what order the graph placed, a single error in the corresponding sequence would could create a large number of wrong pairwise matches. Having graphs $G_1, G_3, G_3$ another strategy could be using pair matching solver to find the mapping between $G_1$ $G_2$, $G_2$ $G_3$, and $G_3$ $G_1$, however using this method it would be likely to get inconsistent or redundant mapping compared with the mapping induced by $G_1$ $G_2$, $G_2$ $G_3$, especially given large number of nodes or significant corruption. And, not considering the mapping between $G_3$ $G_1$, might result in incorrect mapping since the mapping between those two graphs might be more accurate if $G_2$ be heavily corrupted.

### 4.2 Multi-graph Matching

Having three images, $I_1$, $I_2$ and $I_3$ with three extracted feature sets $F_1$, $F_2$, $F_3$a graph $G = (V, E)$ can be constructed, where each node $a = (i, i', i'') \in V$ is an three correspondence tuple, such that $i \in F_1$, $i' \in F_2$, $i'' \in F_3$. An edge $e_{ij} = (v_i, v_j) \in E$ is constructed if the correspondence tuples $a = (i, i', i'')$ and $b = (j, j', j^{"})$ are spatially consistent to each other. And a weight function $M(e_{ij})$ on edge $e_{ij}$ measures the spatial consistency between vertices $v_i$ and $v_j$. If there is a rigid transformations between images, then the spatial consistency means the distance between corresponding points should be preserved $|v_i - v_j| = |v_{i'} - v_{j'}| = |v_{i''} - v_{j''}|$. We used the same technique as for pair-wise graph matching to select the assignment candidates. We first used the KNN method to find the initial matches between every two consecutive images, and filtered out any match that was not consistent on a three image circle. Given three images, first find the initial matches between $I_1$ and $I_2$, $I_2$ and $I_3$, $I_3$ and $I_1$ using nearest neighbor matching and kept the ones those matches that be consistent in all of them, such as $(f_{1,m}, f_{2,n}, f_{3,k})$, where $(f_{1,m}, f_{2,n}) \in L_{1,2}$, $(f_{2,n}, f_{3,k}) \in L_{2,3}$ and $(f_{3,k}, f_{1,m}) \in L_{3,1}$. $L$ is the candidate list between

every two images, and $m, n$ and $k$ are the feature elements. Figure 4.1demonstrates more clearly how the locally matched candidates can be selected using a three-way matching loop. As it can be seen from Figure 4.1 points 1 matches point a, and points a matches point A, also since feature A and feature 1 are matched then candidate 1aA can be chosen as an initial matching candidate. The rest of the candidates are selected using the same technique.

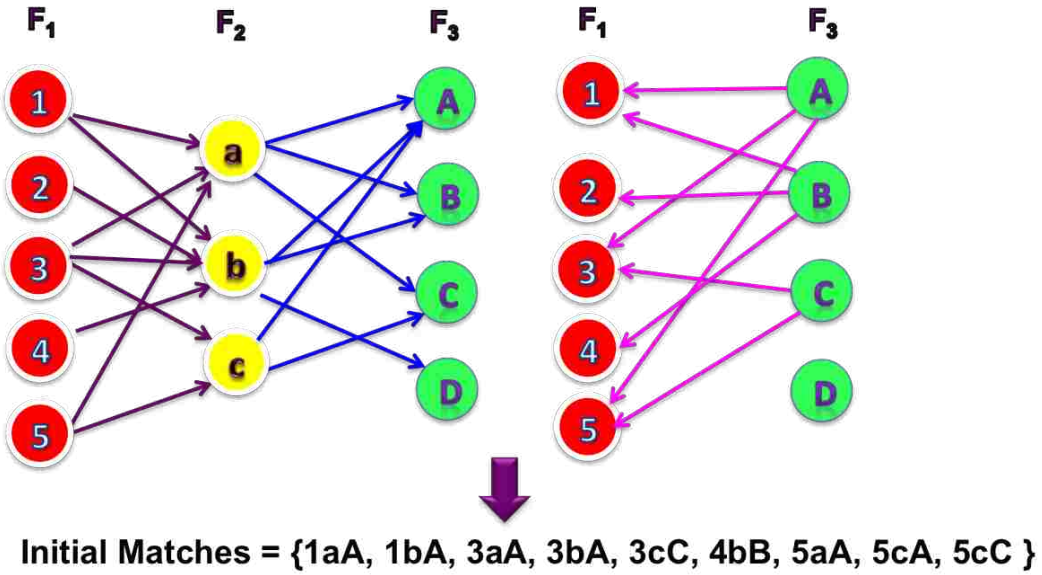



Figure 4.1: Initial correspondences between three images using KNN method.

As it shown in the previous chapter in most of our experiments the CGM algorithm could not be run because of it's large memory footprint, thus in the multi-imaging as the number of images increases, and the number of features grows, using the CGM algorithm would not be plausible. Therefore, our method PBGM would be the best option since the memory size needed is much less, and it can constructed and computed in parallel. The *Pruned Blocked Graph Matching (PBGM)* technique constructed as explained in previous chapter. First using the initial matches candidates unwanted features get pruned. Then the approximate distances between candidates of image1 computed, and for each distinct $\tilde{d_{ij}}$, the sub-matrix blocked affinity constructed. In our multi-image matching problem to find the spatial consistency between two correspond tuples $v_i$ and $v_j$ the Zheng et al. (2016) measurement

is employed. Since, we are interested in images with rigid transformation, Scale Jacobian Metric technique is used to find the three distances. And their spacial consistency would be maximum if $\|f_i f_j\| = \|f_i f_j\| = \|f_i f_j\|$. let $(x_1, y_1) = (0, 0)$, $(x_2, y_2) = (d_{i'j'}, 0)$, and $(x_3, y_3) = \left(\frac{d_{ij}^2+d_{i'j'}^2-d_{i"j"}^2}{2d_{ij}}, \sqrt{d_{i"j"}^2 - \frac{d_{ij}^2+d_{ij}^2-d_{ij}^2}{4d_{ij}^2}}\right)$. Then, the function is defined as follows. If distance satisfy the triangle inequality $d_{ij}-d_{i'j'} \prec d_{i"j"} \prec d_{ij}+d_{i'j'}$ then:

$$S(a,b) = \frac{C * \sqrt{3 * (d_{ij}-d_{i'j'} + d_{i"j"})(d_{ij}+d_{i'j'} - d_{i"j"})(-d_{ij}+d_{i'j'} + d_{i"j"})(d_{ij}+d_{i'j'} + d_{i"j"})}}{d_{ij}^2 + d_{i'j'}^2 + d_{i"j"}^2} \tag{4.1}$$

The point to point similarities estimated using features descriptors, for SIFT descriptors with 128-dimensional vector calculated using equation 4.2.The same idea or technique can be extended to more images. After the affinity for each unique blocked sub-matrix constructed the values of the correspondence that are in conflict with each other set the zero as explained in the previous chapter. We set C value 4.5 as Leordeanu and Hebert (2005).

$$D(H_i, H_{i'}, H_{i"}) = \frac{1}{N}\left(d(H_i, H_{i'}) + d(H_{i'}, H_{i"}) + d(H_{i"}, H_i)\right) \tag{4.2}$$

### 4.2.1 Affinity Matrix using Different Approaches

The four different type of affinity constructed using the four different algorithm ,CGM, AGM, PFGM, PBGM, as discussed in Chapter 3. The node to node and pairwise affinities computed using Equation 4.1, and 4.2 respectively. Having three feature sets of $\mathbf{P}$,$\mathbf{Q}$,$\mathbf{R}$ with $n^P, n^Q, n^R$numbers, the CGM affinity matrix constructed using all the features with size of $n^P$x$n^Q$x$n^R$x$n^P$x$n^Q$x$n^R$. As it shown in the previous chapter in most of our examples the CGM method could not be used because of the memory, therefore in the multi-imaging as the number of images increases, and the number of features grows, using the CGM algorithm is not feasible. Therefore, our method PBGM would be the best option since the memory size needed is much less, and it can constructed and computed in parallel. The

affinity matrix using PFGM approach constructed by finding the locally matched features between every two consecutive frame using KNN method as explained earlier. Therefore, the size of the affinity matrix would be the common matches of the locally pair-wised matched images. Using the PBGM algorithm for each unique pairwise distances of image1 features, the blocked sub-matrices constructed using the pruned assignments containing image2 and image3 features. The candidates pruned as explained earlier by using the initial matching points. Therefore the memory space required for employing the PBGM method would be $z''$ by $n^{Q''}$x $n^{R''}$x $n^{Q''}$x $n^{R''}$, where $n^{Q''}$, and $n^{R''}$are the pruned points.

### 4.2.2 Bijectivity Constraint

The bijectivity constraints for multi-images would be same idea as the pairwise-matching. If node $(i, i', i'')$ selected together then it is prohibited to select nodes $(j, i', j'')$, $(j, j', i'')$, $(i, j', j'')$. Therefore, the bijectivity constraint can be formulated as a linear constraint $Ax \leq b$. $A$ is a sparse matrix consists of *0* and *1* elements. On each row of $A$, the non-zero elements give the indices of tuples that are associated with a same keypoint in one frame. Therefore, these tuples have mutual conflicts. No more than one among these tuples should be selected in the final solution, as indicated by the vector $b = (1, 1, ..., 1)^T$ .

### 4.2.3 Eigenvector Computation

The spectral matching algorithm as explained before used to solve the $x_d^*$. By first computing the principal eigenvalue and its corresponding eigenvector, denoted as $x^*$ , then sort elements in $x^*$ , and find the greatest element $g$ and set $x_d^*(g) = 1$. Then, iteratively, following the descending order, find all elements in $x^*$ that do not have conflict with existing marked elements and set those elements to *1*, while marking the indicator of each conflict element to *0*. In our method since each affinity constructed in parallel and multiplied by the eigenvector separately. Thrust is used to construct each of the blocks in parallel. And, the CPU's and GPU's accelerated libraries MKL, cuBlas, cuSpars for matrix vector multiplication. All the four methods constructed using aforementioned accelerated libraries to

find their eigenvalues.

### 4.3 Result

The result of our method PBGM is compared using Thrust and the optimized libraries in terms of execution speed. Result also compared to the original approach without using any optimization. By using cuBlas library the performance increases tremendously since it is done in parallel. By increasing number of features, there would be more computation therefore, the overhead for sending the data to GPU compensated.It can be seen from images since the affinity values are non zero values, using sparse representation reduce the memory space requirement, also reduce the computation time. But in a smaller inputs did not pay off for the smaller examples since manipulating sparse matrixes are computationally expensive compared to the original representation. This caused a slowdown Figure 4.3 and Figure 4.4. Nevertheless, for larger number of features, Figure 4.5, we clearly see that sparsifying the matrix clearly improves the run time.

In the first example the computation time using no optimization is displayed as it can be seen in Figure 4.2. The total size for constructing the whole affinity matrix is $6.89 * 10^7$. As the input data in this example is small the affinity matrix could be constructed. Comparing to the the whole computation without any acceleration the speed up we got using MKL, MKL-CSR, cuBlas and cuSparse is $80.24\%$, $48.05\%$, $90.88\%$, $86.42\%$ respectively. The amount of memory reduced by $99.89\%$.

Figure 4.3 display the same example as Figure 4.2, but without the no acceleration bar. Comparing the four different algorithm the experiment showed that the execution time dropped by $53.84\%$. using cuBlas compared to MKL. But since sparsifying the blocked sub-matrices slow down the performance we got only $31.25\%$ reduction compared to MKL.

In this example Figure 4.4 the total size for constructing the whole affinity matrix is $8.37 * 10^8$. Using PBGM method the amount of memory required to store the blocked affinity sub_matrices reduced by $99.93\%$. Also, as the data size grew and the amount

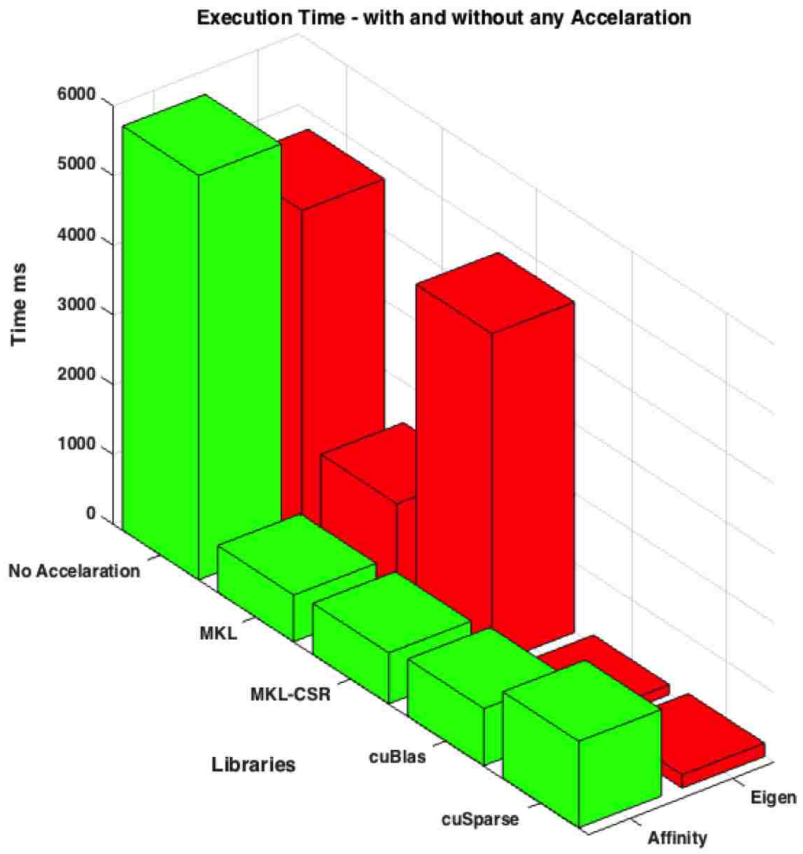



Figure 4.2: Execution time using different accelerated libraries. - Total size: $6.89 * 10^7$

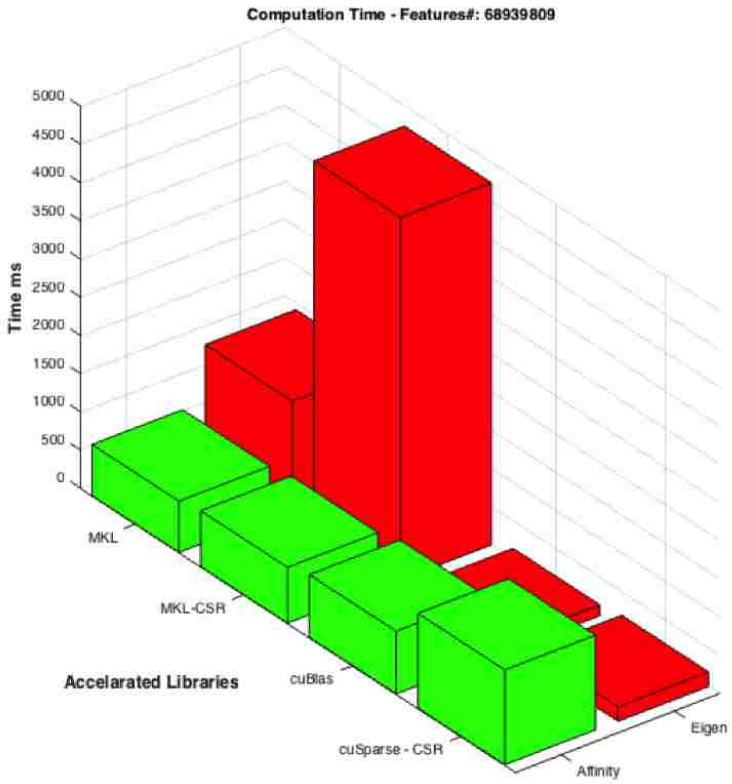



Figure 4.3: Execution time using different accelerated libraries. - Total size: $6.89 * 10^7$

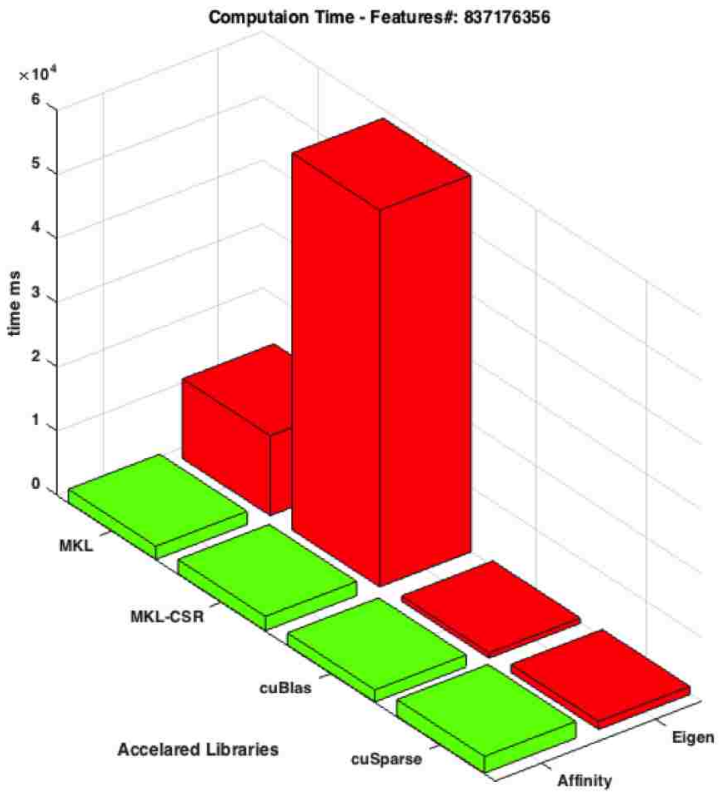



Figure 4.4: Execution time using different accelerated libraries. - Total size: $8.37 * 10^8$

of computation increased, we got 81.54% of speed up using cuBlas, and $73.51\%$ using cuSparse comparing to MKL.

Figure 4.5 is the result of a experiment with total size of $1.2 * 10^{12}$. The result showed that using our method PBGM required up to $99.99\%$ less memory than using CGM method. Also, as the input size got bigger the sparsed version, using cuSparse, the computation run faster with speed up of $95.35\%$ comparing to MKL, and $91.99\%$ using cuBlas.

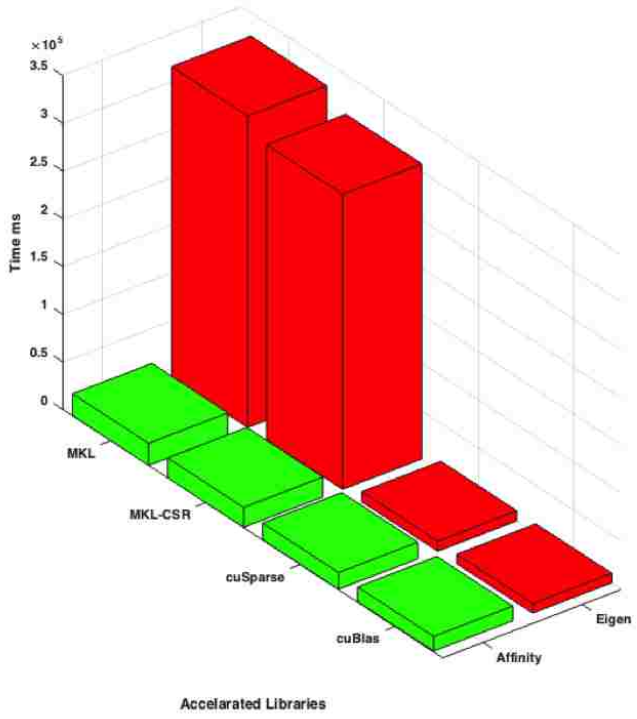



Figure 4.5: Execution time using different accelerated libraries. - Total size: $1.2 * 10^{12}$

# Chapter 5
# FUTURE WORK AND CONCLUSION

## 5.1 Conclusion

Matching features of two images or different parts of a single image is a well-known problem in the area of computer vision. Stabilizing consistent correspondences between two sets of features is a computationally expensive operation. For good matching performance, it is important to take into consideration not only the local appearance of features (a linear term) but also the higher-order geometry and appearance of groups of features (a quadratic term). In many computer vision problems, and complex scenes graphs utilized as an abstract representation. Where the vertices of graphs represent the local characteristics of features extracted from given images, the edges relate to relational aspects between those features. Given two images, each represented as an attributed graph the objective is finding the correct correspondences between those two graphs such that both terms, a linear and quadratic term, be preserved. Typically, the problem of graph matching mathematically formulated as a quadratic assignment problem. Solving this problem is NP-hard, and an exact optimal algorithm can only work for very small graphs. Therefore, extensive research has been done on developing more accurate and faster algorithms to solve the problem approximately.

Explicitly constructing the affinity matrix $\mathbf{M}$ using all the feature points of both images, and computing the first eigenvector of $\mathbf{M}$ are hopeless due to the heavy storage requirement of $\mathbf{M}$. Hence, many researchers employ using only the initial matching points to reduce the dimension of the problem search space, but it results to not accurate solution. Since, only using the locally matched points as the candidate assignments to find the correct correspondences on noisy images, not discriminative images, or images with lots of ambi-

guity would not give correct correspondences. Therefore, our algorithm, PBGM, would be the best option in this circumstances, as it not only reduce the memory requirement and overall calculation, computation but also delivers great performance on not discriminative, noisy images. We reduced the memory usage and the computation cost of Pairwise graph matching by adopting a heuristic pruning strategy together with a redundancy pattern suppression scheme. Most of the graph matching techniques are sequential since there is dependency involve for constructing and computing the affinity matrix. Moreover, all of the implementations are matrix-vector multiplication, which is a very well-known problem in computer science area, and numerous techniques have been integrated into compilers and libraries to speed up such computation. Blocking the affinity matrix provides an additional level of parallelism. Since each block can be constructed and used independently of other blocks, we have all the computations can be done totally in parallel without any synchronization (embarrassingly parallel). We parallelized our algorithm, PBGM and used CPU's GPU's accelerated libraries to reduce the execution time. We also improve the performance of the other three methods, CGM, AGM, and PFGM by employing MKL cuBlas, cuSparse. Using Thrust library has enabled us to provide a unified interface for both CPU and GPU, where allows us to heuristically chose the best combination of algorithm and architecture based on the input images. The experimental result proved that our proposed algorithm PBGM require up to 99.97% less memory than the original method CGM. And, 93.43% faster than the CGM without any accelerations. Also, comparing to PFGM, the accuracy improved about 60%. We showed that by using our method graph matching for big data was feasible.

Current graph matching methods mostly focused on two-graph matching directing on establishing one-to-one correspondences between a pair of feature points. However, in many applications, similar objects do not appear in a pair but more in collections. Also, considering more graph instead of two improves the accuracy, since it is required to find global consistency through all the given data. As it mentioned, solving GM is memory wise and

computationally expensive. Using our heuristic pruning strategy and redundancy pattern along with parallelizing the algorithm enabled us to address this problem in a more efficient way.

We also extended our work for Multi-graph imaging using our proposed PBGM method for reducing the memory footprint by 99.91% comparing to CGM method and increasing the speed up. We also modified the structure of the affinity matrix for minimizing memory requirement and parallelizing our algorithm by employing CPU's and GPU's accelerated libraries. The matrix-vector multiplication of gradient computation performed in parallel, where each block-vector calculation computed independently. The Scale Jacobian metric technique employed to measure the similarities between the candidates. And, Thrust library used to unified interface for both CPU and GPU. To improve the overall accuracy for all the four methods we proposed the new algorithm for a more robust solver, but more work needs to be done which is in our future work.

### 5.2 Future Work

Spectral matching algorithms are sensitive to contaminations which could create false edges in the graphical model, causing the solution deviate from the actual global optimum. That is why the performances of spectral matching algorithms decline dramatically with increasing noise. Therefore, a robust to noise and an efficient matching algorithm is of great interest to us. As it mentioned before none of the previous spectral methods such as Leordeanu and Hebert (2005), considered the original integer constraints during optimization. Based on the assumption that the continuous optimum is close to the discrete global optimum little computational time applied to binarize the solution. Integer Projected Fixed Point (IPFP) algorithm Leordeanu et al. (2009) presented in their paper that by considering the binarization step during the post-processing, can improve the result significantly. Based on these investigations, they used an iterative algorithm to improve any continuous or dis-

**Algorithm 4** IPFP

1. $x^* = x, \quad S^* = x^T M x, \quad k = 0, \quad x_i \geq 0$;
2. let $b_{k+1} = P_d(M\, x_k)$, where $P_d$ is the projection form continuous to discrete domain, and $b_{k+1}$ is the discrete vector.

   $C = x_k^T M(b_{k+1} - x_k), \qquad D = (b_{k+1} - x_k)^T M(b_{k+1} - x_k)$ ;
3. if $D \geq 0$, *set* $x_{k+1} = b_{k+1}$, else let $r = min\,\{-C/D, 1\}$
4. if $b_{k+1}^T M b_{k+1} \geq S^*$, then set $S^* = b_{k+1}^T M b_{k+1}$, and $x^* = b_{k+1}$
5. if $x_{k+1} = x_k$, stop and return the solution $x^*$
6. set $k = k + 1$ and go back to step2.

crete solution quickly. Each iteration consists of two stages. The first one optimizes in the discrete domain, a linear approximation of the quadratic function around the current solution. It gives a direction along which the second stage maximizes the original quadratic score in the continuous domain. The stage two can be viewed as a projection on the discrete domain and this algorithm is called Integer Projected Fixed Point (IPFP) algorithm Leordeanu et al. (2009). It aims to optimize the following continuous problem, in which the integer constraint from Equation 2.9 is removed:

$$x^* = argmax(x^T M x) s.t.\; Ax = 1,\; (x_i \in x) \succeq 0 \tag{5.1}$$

Algorithm 4 shows the steps of IPFP algorithm. In step 1, the quadratic score $x_K^T M x_k$ is first approximated by the first-order Taylor expansion around the current solution $x_k : x^T M x \approx x_K^T M x_k + 2 x_K^T M \left(x - x_k^T\right)$. In step 2, the two stages are introduced. Stage one: the continuous approximation is maximized within the discrete domain by the projection $\mathrm{P}_d$ under the one-to-one discrete constraints. Since all possible discrete solutions have the same norm, $\mathrm{P}_d$ boils down to finding the discrete vector $b_{k+1} = argmax\; b^T M x_k$ . Stage two: the same discrete $b_{k+1}$ maximizes the dot product in the continuous domain, $Ax = 1, x \succ 1$.

Step 3 and Step 4 are the updating rules. Finally, Step 5 is the termination criteria. IPFP algorithm is, in fact, a relaxation problem of the original one by removing the integer constraints. It is equivalent to the original problem if the proximity matrix M is convex. The algorithm is a sequence of linear assignment (or independent labeling) problems, in which the next solution is found based on the previous one. Step 3 ensures that the quadratic score increases with each iteration, and Step 4 guarantees that the the initial solution is not better than the returned binary solution. The role of $b_{k+1}$ is to provide a direction of largest possible increase in the first-order approximation, within both the continuous domain and the discrete domain simultaneously. The original quadratic score can be also further maximized in the continuous domain (as long as $b_{k+1} \neq x_k$ ). In their paper, they claimed the algorithm converges in about 5 to 10 steps, which makes it very efficient. Theoretical properties of IPFP have been analyzed. It is shown that IPFP has strong convergence and climbing guarantees, which is stated in Leordeanu et al. (2009) as follows:

1. The quadratic score $x_K^T M x_k$ increases at every step $k$ and the sequence of $x_k$ converges.

2. The algorithm converges to a maximum of the relaxed problem.

3. If $M$is positive semi-definite with positive elements, then the algorithm converges in a finite number of iterations to a discrete solution, which is a maximum of the relaxed problem.

4. if $M$ has non-negative elements and rank=1, then the algorithm will converge and return the global optimum of the original problem after the first iteration.

We proposed the new algorithm for the pairwise matching, Algorithm 5 as a more reliable and robust solver for finding the global optima. Our iterative algorithm works by modifying the affinity matrix and optimizing the original problem with its integer constraints. To binarize the solution during the optimization stage we applied the IPFP algorithm Leordeanu et al. (2009)as discussed earlier. We added the node to node affinity score using Hager et al. (2000) approach to increase the matching score of the correct correspondences. In the first

stage, we constructed the affinity matrix by finding the pairwise spatial score between every nodes of the two graphs. The unary score computed by selecting the maximum of the pairwise score between each of the features of the first image to all the second image features. Step 10 ensures that the quadratic score increases with each iteration, and Step 11 guarantees that the the initial solution is not better than the returned binary solution. The role of $b_{l+1}$ is to provide a direction of largest possible increase in the first-order approximation, within both the continuous domain and the discrete domain simultaneously. The original quadratic score can be further maximized in the continuous domain (as long as $b_{l+1} \neq x_l$ ).

**Algorithm 5** New solver

***Input:*** Two sets of features P and Q, each having |P| and |Q| features respectively.
***Output:*** **x** a binary integer variable indicating correct correspondences in the final solution.

1. repeat;
2. construct the affinity matrix **M**, using pairwise geometry similarity;
3. construct the diagonal matrix **D**,

   where$d(ii^{'},ii^{'}) = max\left\{m(ii^{'}, jj^{'}) : 1 \leq jj^{'} \leq n^P n^Q\right\}$;
4. initialize $\boldsymbol{x}$ uniformly or randomly with $||\mathbf{x}|| = 1$;
5. $x \leftarrow Mx + D - Dx$;
6. $x^* = x, \quad S^* = x^T Mx, \quad l = 0, \quad x_i \geq 0$;
7. let $b_{l+1} = P_d(M\, x_l)$, where $P_d$ is the projection form continuous to discrete domain, and $b_{l+1}$ is the discrete vector.

   $C = x_l^T M(b_{l+1} - x_l), \qquad B = (b_{l+1} - x_k)^T M(b_{l+1} - x_l)$ ;
8. if $B \geq 0, set\, x_{l+1} = b_{l+1}$, else let $r = min\left\{-C/B, 1\right\}$
9. if $b_{l+1}^T M b_{l+1} \geq S^*$, then set $S^* = b_{l+1}^T M b_{l+1}$, and $x^* = b_{l+1}$
10. if $x_{l+1} = x_l$, stop and return the solution. Else, set $l = l + 1$ and go back to step1.

Our hope is by using this algorithm and some further modification improve our solution. We plan to extend our new solver to use for multi imaging. Also, accelerate our problem furthermore by employing distributed system using MPI.

# Bibliography


Alfred V Aho, Mahadevan Ganapathi, and Steven WK Tjiang. Code generation using tree matching and dynamic programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(4):491–516, 1989.

Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 510–517. Ieee, 2012.

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

Paul R Beaudet. Rotationally invariant image operators. In *International Joint Conference on Pattern Recognition*, volume 579, page 583, 1978.

Horst Bunke and Urs Bühler. Applications of approximate string matching to 2d shape recognition. *Pattern recognition*, 26(12):1797–1812, 1993.

Tibério S Caetano and Terry Caelli. Approximating the problem, not the solution: An alternative view of point set matching. *Pattern recognition*, 39(4):552–561, 2006.

Tibério S Caetano, Terry Caelli, and Dante AC Barone. An optimal probabilistic graphical model for point set matching. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 162–170. Springer, 2004.

Marco Carcassoni and Edwin R Hancock. Spectral correspondence for point pattern matching. *Pattern Recognition*, 36(1):193–204, 2003.

Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *Computer Vision–ECCV 2010*, pages 492–505. Springer, 2010.

Haili Chui and Anand Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2):114–141, 2003.

Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004.

Timothee Cour, Praveen Srinivasan, and Jianbo Shi. Balanced graph matching. *Advances in Neural Information Processing Systems*, 19:313, 2007.

Olivier Duchenne, Francis Bach, In-So Kweon, and Jean Ponce. A tensor-based algorithm for high-order graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(12):2383–2395, 2011a.

Olivier Duchenne, Armand Joulin, and Jean Ponce. A graph-matching kernel for object categorization. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1792–1799. IEEE, 2011b.

Maximally Stable Extremal, J Matas, O Chum, M Urban, and T Pajdla. Robust wide baseline stereo from. In *In British Machine Vision Conference*. Citeseer, 2002.

Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

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

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

William W Hager, Soon Chul Park, and Timothy A Davis. Block exchange in graph partitioning. In *Approximation and Complexity in Numerical Optimization*, pages 298–307. Springer, 2000.

Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

Herve Jegou, Florent Perronnin, Matthijs Douze, Jorge Sánchez, Patrick Perez, and Cordelia Schmid. Aggregating local image descriptors into compact codes. *IEEE transactions on pattern analysis and machine intelligence*, 34(9):1704–1716, 2012.

U Kang, Martial Hebert, and Soonyong Park. Fast and scalable approximate spectral graph matching for correspondence problems. *Information Sciences*, 220:306–318, 2013.

Scott Krig. Interest point detector and feature descriptor survey. In *Computer Vision Metrics*, pages 217–282. Springer, 2014.

Kenneth Lange. *Numerical analysis for statisticians*. Springer Science & Business Media, 2010.

Jungmin Lee, Minsu Cho, and Kyoung Mu Lee. Hyper-graph matching via reweighted random walks. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1633–1640. IEEE, 2011.

Chengcai Leng, Wei Xu, Irene Cheng, and Anup Basu. Graph matching based on stochastic perturbation. *IEEE Transactions on Image Processing*, 24(12):4862–4875, 2015.

Marius Leordeanu and Martial Hebert. A spectral technique for correspondence problems using pairwise constraints. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1482–1489. IEEE, 2005.

Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and map inference. In *Advances in neural information processing systems*, pages 1114–1122, 2009.

Marius Dan Leordeanu. *Spectral graph matching, learning, and inference for computer vision*. Carnegie Mellon University, 2010.

Nikos K Logothetis and David L Sheinberg. Visual object recognition. *Annual review of neuroscience*, 19(1):577–621, 1996.

David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

Jianfeng Lu, Terry Caelli, and Jingyu Yang. A comparison of least squares and spectral methods for attributed graph matching. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 325–333. Springer, 2004.

Eugene M Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of computer and system sciences*, 25(1):42–65, 1982.

João Maciel and João P Costeira. A global solution to sparse correspondence problems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(2):187–199, 2003.

Richard Myers, RC Wison, and Edwin R Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, 2000.

Pavel A Pevzner. Multiple alignment, communication cost, and graph matching. *SIAM Journal on Applied Mathematics*, 52(6):1763–1779, 1992.

Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, pages 876–879, 1964.

Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *Computer Vision–ECCV 2008*, pages 596–609. Springer, 2008.

Tinne Tuytelaars, Krystian Mikolajczyk, et al. Local invariant feature detectors: a survey. *Foundations and trends® in computer graphics and vision*, 3(3):177–280, 2008.

Barend J van Wyk and Michaël A van Wyk. Kronecker product graph matching. *Pattern Recognition*, 36(9):2019–2030, 2003.

Jiangjian Xiao, Hui Cheng, Harpreet Sawhney, and Feng Han. Vehicle detection and tracking in wide field-of-view aerial video. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 679–684. IEEE, 2010.

Junchi Yan, Yu Tian, Hongyuan Zha, Xiaokang Yang, Ya Zhang, and Stephen M Chu. Joint optimization for consistent multiple graph matching. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1649–1656. IEEE, 2013.

Junchi Yan, Yin Li, Wei Liu, Hongyuan Zha, Xiaokang Yang, and Stephen Mingyu Chu. Graduated consistency-regularized optimization for multi-graph matching. In *Computer Vision–ECCV 2014*, pages 407–422. Springer, 2014.

Junchi Yan, Jun Wang, Hongyuan Zha, Xiaokang Yang, and Stephen Chu. Consistency-driven alternating optimization for multigraph matching: A unified approach. *IEEE Transactions on Image Processing*, 24(3):994–1009, 2015.

Laura A Zager and George C Verghese. Graph similarity scoring and matching. *Applied mathematics letters*, 21(1):86–94, 2008.

Ron Zass and Amnon Shashua. Probabilistic graph and hypergraph matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994.

Shuai Zheng, Jun Hong, Kang Zhang, Baotong Li, and Xin Li. A multi-frame graph matching algorithm for low-bandwidth rgb-d slam. *Computer-Aided Design*, 78:107–117, 2016.

Feng Zhou and Fernando De la Torre. Factorized graph matching. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 127–134. IEEE, 2012.

# VITA

Sahar Marefat Navaz was born in Tehran, Iran. She graduated from University of Northumbria, Newcastle, UK in 2006, with bachelor's degree in Electrical and Communication Engineering. In 2008, she received her master's degree in Communication and Signal Processing from University of Newcastle, Newcastle, UK. She is currently a PhD student at the school of Electrical Engineering and Computer Science at Louisiana State University, Baton Rouge.