Title
Part I - Constrained Shortest-Path For Manifold Learning And Multiple Manifold Clustering
Part II - Community Detection In Large Graphs; Analysis, Design And Implementation

Permalink
https://escholarship.org/uc/item/4w64w79x

Author
Babaeian, Amir

Publication Date
2017

Peer reviewed|Thesis/dissertation
UNIVERSITY OF CALIFORNIA, SAN DIEGO

Part I - Constrained Shortest-Path For Manifold Learning And Multiple Manifold Clustering
Part II - Community Detection In Large Graphs; Analysis, Design And Implementation

A dissertation submitted in partial satisfaction of the requirements for the degree
Doctor of Philosophy

in

Mathematics with a specialization in Statistics

by

Amir Babaeian

Committee in charge:

Professor Ery Arias-Castro, Chair
Professor Li-Tien Cheng
Professor Michael J. Holst
Professor Dimitris Politis
Professor Lawrence K. Saul

2017
The dissertation of Amir Babaeian is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2017
DEDICATION

To my parents, my sister and my uncle for their love, endless support and encouragement.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature Page</td>
<td>iii</td>
</tr>
<tr>
<td>Dedication</td>
<td>iv</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>x</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>xi</td>
</tr>
<tr>
<td>Vita and Publications</td>
<td>xii</td>
</tr>
<tr>
<td>Abstract of the Dissertation</td>
<td>xiii</td>
</tr>
</tbody>
</table>

## I Constrained Shortest-Path For Manifold Learning And Multiple Manifold Clustering

| Chapter 1 Introduction                      | 2    |
| 1.1 Motivation                              | 2    |
| 1.2 Problem statement                       | 3    |
| 1.3 Contribution                            | 3    |
| 1.4 Structure of part I                     | 4    |

| Chapter 2 Manifold learning                | 5    |
| 2.1 Manifold learning problem              | 5    |
| 2.1.1 Isomap                                | 7    |
| 2.2 Constrained shortest-path distance     | 10   |
| 2.3 Approximate constrained shortest-path algorithm | 15   |
| 2.4 Exact constrained shortest-path algorithm | 17   |
| 2.4.1 Complexity of the exact constrained shortest-path algorithm | 19   |
| 2.5 Constrained shortest-path algorithm     | 20   |
| for manifold learning                      |      |
| 2.6 Theoretical support                     | 20   |
| 2.7 Implementation and numerical experiments| 28   |

<p>| Chapter 3 Manifold clustering              | 31   |
| 3.1 Manifold clustering problem            | 31   |
| 3.2 Algorithms for multi-manifold clustering | 32   |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Manifold Learning</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>MDS on matrix of distances</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>The geodesic distances on the graph (in red) approximate the Euclidean distance (in blue) Silva and Tenenbaum [2002]</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>Consistency of the discrete curvature</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>$x$ and $z$ lie in annulus neighborhood of point $y$.</td>
<td>16</td>
</tr>
<tr>
<td>2.6</td>
<td>(a) original graph. (b) edge graph</td>
<td>19</td>
</tr>
<tr>
<td>2.7</td>
<td>Simulated data illustrating the problem of manifold learning. Left: input data; Right: output from our method.</td>
<td>21</td>
</tr>
<tr>
<td>2.8</td>
<td>The proof sketch that shortest-path on the graph approximates the intrinsic metric.</td>
<td>24</td>
</tr>
<tr>
<td>2.9</td>
<td>Results of manifold learning algorithms on a self-intersect manifold. (a) self-intersect manifold; (b) our method; (c) Isomap; (d) LLE; (e) HLLE; (f) LTSA.</td>
<td>29</td>
</tr>
<tr>
<td>2.10</td>
<td>Results of manifold learning algorithms on a non-compact manifold with a hole. (a) non-compact manifold; (b) our method; (c) LLE; (d) HLLE; (e) LTSA; (f) MVU. Our algorithm preserves the similarity of manifold.</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Simulated data illustrating the problem of multi-manifold clustering. Left: 3D data. Right: output from our method.</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>A set of sample points in $R^3$ drawn from a union of three subspaces.</td>
<td>36</td>
</tr>
<tr>
<td>3.3</td>
<td>Result of our method on 8 synthetic datasets.</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>Examples where the other methods fail. (a) SCC, (b) SMMC, (c, d) K-Manifolds.</td>
<td>41</td>
</tr>
<tr>
<td>3.5</td>
<td>The 20 objects from the COIL-20 database.</td>
<td>42</td>
</tr>
<tr>
<td>3.6</td>
<td>Output of three different methods on the first 5 objects of COIL-20. Points are represented by their 3 top principal components.</td>
<td>43</td>
</tr>
<tr>
<td>3.7</td>
<td>Some sample images of Caltech dataset.</td>
<td>43</td>
</tr>
<tr>
<td>3.8</td>
<td>Result of human activity segmentation using Path-Based Clustering. There are 4 activities: walking (1), looking (2), sitting (3) and standing (4). Top: a sample of the sequence. Middle: ground truth. Bottom: output of our algorithm.</td>
<td>45</td>
</tr>
<tr>
<td>3.9</td>
<td>The first, 56th and last frame(135th) of Fox news video sequence.</td>
<td>46</td>
</tr>
<tr>
<td>3.10</td>
<td>Example of noisy data.</td>
<td>46</td>
</tr>
<tr>
<td>3.11</td>
<td>Left: Two intersecting curve with added uniform noise. Right: The performance of our algorithm and SMMC as a function of noise level.</td>
<td>47</td>
</tr>
</tbody>
</table>
Figure 3.12: Robustness of proposed method to outliers. Left: A curve (100 points) is embedded in a crowd of outlier points (1600 points). Right: The proposed method could automatically recover the curve from outliers.

Figure 3.13: Left: Two intersecting Spheres. Right: The distribution of maximum angle along an unconstrained shortest-path between all the points and a given landmark.

Figure 3.14: The sensitivity of six synthetic datasets to the angle constraint.

Figure 5.1: The Zachary Karate Club network, an example of a sparse network.

Figure 7.1: Left: Change of labels in one iteration of Algorithm 10. Right: Change of labels in one iteration of Algorithm 12 with $\alpha = 1$.

Figure 8.1: Simple stochastic block model with parameters $n = 100, K = 3, p_1 = .09, p_0 = .005$.

Figure 9.1: MapReduce steps.

Figure 9.2: Top: MapReduce for iterative algorithm. Bottom: Spark for iterative algorithm.

Figure 9.3: Send, Reduce and Update steps on a simple graph. In this example, Reduce function computes the messages sent to the node with $ID = 2$.

Figure 10.1: Separability measure on the Amazon, DBLP, Youtube, Live Journal and Friendster datasets.

Figure 10.2: Density measure on the Amazon, DBLP, Youtube, Live Journal and Friendster datasets.
### LIST OF TABLES

| Table 3.1: Clustering accuracy on synthetic data. | 41 |
| Table 3.2: Clustering accuracy of multiple subjects for the Caltech dataset | 44 |
| Table 10.1: Number of times out of 50 experiments the algorithm returns the correct community structure | 77 |
| Table 10.2: The number of iterations message passing Algorithms 12 and 13 take to detect perfect communities. The number of nodes in each community is $n$. | 79 |
| Table 10.3: The number of iterations Algorithms 12 and 13 take to detect perfect communities. The number of communities is $K$. | 80 |
| Table 10.4: Experimental set up and result in Spark for the real datasets. | 81 |
| Table 10.5: Average F1 score on detected communities by applying Algorithm 12 | 82 |
I would like to thank UCSD and all the friends I found here in mathematics department. I would like to thank Prof. Ery Arias-Castro for his insightful recommendations and advice regarding my thesis, particularly the material for Part I and stochastic block model for Part II. I would like to thank the rest of my thesis committee: Prof. Cheng, Prof. Holst, Prof. Politis and Prof. Saul, for their constructive comments and encouragement, as well as other faculty members in the mathematics department whom I had a chance to learn from. Thanks to co-authors of my papers, Alireza Bayestehtashk, Mojtaba Bandarabadi and Mohammadreza babacee.


Thanks to Amin Ghadesi for helping me regarding the graphs of my thesis. I also want to thank my friend Sinan Aksoy for his helpful discussions and advise regarding this manuscript. Last but not least, I want to thank my parents, my sister and my uncle whose support and encouragement has helped me through my life.
VITA

2006 BSc in Electrical Engineering, University of Mazandaran, Babolsar, Iran

2009 MSc in Electrical Engineering, Amirkabir University of Technology, Tehran, Iran

2013 MSc in Statistics, University of California, San Diego

2017 PhD in Mathematics with a specialization in Statistics, University of California, San Diego

PUBLICATIONS

“Nonlinear subspace clustering using curvature constrained distances”, Pattern Recognition Letter 2015, 68, 118-125

“Multiple manifold clustering using curvature constrained path”, PloS one Journal 2015, 10 (9), e0137986.
ABSTRACT OF THE DISSERTATION

Part I - Constrained Shortest-Path For Manifold Learning And
Multiple Manifold Clustering
Part II - Community Detection In Large Graphs; Analysis, Design
And Implementation

by

Amir Babaeian

Doctor of Philosophy in Mathematics with a specialization in Statistics

University of California San Diego, 2017

Professor Ery Arias-Castro, Chair

In Part I of this thesis, we address the problem of manifold learning and clustering by introducing a novel constrained shortest-path algorithm. In the case of self-intersection, existing methods such as Isomap fail to capture the true shape of the surface near the intersection. We tackle this problem by imposing a curvature constraint to the shortest-path algorithm used in Isomap. We demonstrate theoretically that under a regularity assumption on manifolds, the proposed algorithm is able to capture the underling parameter space of the manifolds with self-intersection. We apply our method to simulated and real datasets and show it performs comparably to current state-of-the-art methods such as K-manifold and spectral multi-manifold clustering clustering.
In Part II, we propose a new community detection algorithm for large networks. We introduce a new distributed implementation of message passing algorithm with a dynamic probability of update. We analyze our algorithm on a few random and real networks to provide insight about the choice of parameter. We discuss the implementation of the algorithm on the Spark platform. We also analyze the performance of the algorithm in terms of speed and accuracy using given ground truths and a few modularity measures like separability and density of detected communities. We run our algorithm on graphs with millions of nodes and billions of edges. Experiments show that the proposed algorithm is efficient and linearly adaptable to the scale of our data by adding extra machines as the size of our networks increase.
Part I

Constrained Shortest-Path For Manifold Learning And Multiple Manifold Clustering
Chapter 1

Introduction

Today, we live in a world with a lot of high-dimensional and complex data and with very exiting data analysis frameworks. Since last decade, the amount of collected data is increasing exponentially. For instance, the amount of uploaded video in youtube is 500 hours every minute. Evidently, we are dealing with very high-dimensional data in different areas of machine learning and pattern recognition. In most cases the, high-dimensional data lie close to a low-dimensional manifold embedded in a high-dimensional ambient space. Finding this low-dimensional data representation is a challenging problem which has attracted great attention from researchers.

1.1 Motivation

Manifold learning and subspace clustering has a lot of applications in data representation tasks such as video segmentation, web data analysis, bioinformatics and earth observation data processing. Current feature extraction techniques represent the content of each data sample with very high-dimensional and complex features that bring new challenges in computation and storage. However, not all feature values are informative. As it is shown in recent work, most of the data are drawn form very low dimensional manifolds that are embedded in the original ambient space. This motivates researches to develop new techniques to recover these manifolds which is a very challenging and attractive research area in com-
puter vision and machine learning. This has also motivated us to develop new techniques that could surpass the limitations of available methods.

1.2 Problem statement

In the first part of this thesis we exploit the problem of self intersected manifold learning and multiple manifold clustering. Most of the proposed methods for manifold learning assume that the data has no self intersection. However, in real applications, there are a lot of cases where the data is drawn from one or more self intersected manifolds. Therefore, these techniques fail when data has such a self intersection property. To resolve this problem we investigated this issue by proposing a new distance definition, the so-called constrained shortest-path distances.

1.3 Contribution

This part is mainly focused on manifold learning and subspace clustering. The goal is to provide a computational framework which is as automatic as possible while being accurate to extract a low-dimensional manifold. Specifically, we propose a novel manifold leaning algorithm and subspace clustering based on defining a new shortest-path between each pair of data points. More precisely, the contributions of thesis are comprised as:

- Providing a review of state-of-the-art in the area of dimensionality reduction and manifold learning and clustering
- Proposing a constrained shortest-path distance based on notions of curvature and angle
- Introducing novel manifold learning and clustering algorithms based on constrained shortest-path distance
- Evaluating proposed algorithms by applying them to both synthetic and real datasets and comparing the obtained results with the state-of-the-art
1.4 Structure of part I

The first part of thesis is divided into four main building blocks. Each of these parts contains thematically related sections and all parts build upon each other. The outline of this part reads as follows:

1. Introduction: We give general information about the motivation, problem statement, and contribution of this thesis.

2. Manifold Learning: We introduce background concepts including previous work in dimensionality reduction and manifold learning. We also discuss the Isomap algorithm. A novel constrained shortest-path distances is introduced followed by a novel manifold learning algorithm. All the required supporting parts, as well as experiments and results, are presented that illustrate the efficiency of the proposed algorithm.

3. Manifold Clustering: Having discussed the constrained shortest-path distances, this chapter addresses the problem of multiple manifold clustering using the proposed distances. We begin by providing a review of related work in this area and then present the algorithm. Finally, experiments and results obtained in different real and synthetic datasets are explained.

4. Summary and Conclusion: The most important parts presented in the thesis are summarized and closing remarks are given. Additionally, possible directions for future work on related topics are explored.
Chapter 2

Manifold learning

2.1 Manifold learning problem

Manifold learning is now an established line of research within machine learning and statistics. Paraphrasing Cayton (2005), the task of manifold learning can be defined as follows:

Problem 1 (Manifold embedding). Given a sample \( x_1, \ldots, x_n \in \mathbb{R}^D \) from a smooth \( d \)-dimensional submanifold \( \mathcal{M} \) admitting a single global isometric chart \( f: \mathcal{M} \to \mathbb{R}^d \), find \( u_1, \ldots, u_n \in \mathbb{R}^d \) such that \( u_i = f(x_i) \) (up to a rigid motion). Figure 2.1.

We note at once that the problem is ill-posed, since without further constraints, \( f \) can only be recovered up to a diffeomorphism of \( \mathbb{R}^d \). The classical method of Principal Components Analysis (PCA) is tailored to the special case where \( \mathcal{M} \) is an affine surface. Given a dataset, PCA uses an orthogonal transformation of data, using this transformation we project the data points on to the direction called first principal component. The projection of points on to this direction have maximum variance. More recent methods address the situation where \( \mathcal{M} \) is curved, like Self-Organizing Maps Kohonen (1982), Principal Curves/Surfaces Hastie and Stuetzle (1989), Isomap Tenenbaum et al. (2000), Schwartz et al. (1989), Local Linear Embedding (LLE) Roweis and Saul (2000), Laplacian Eigenmaps Belkin and Niyogi (2003), Manifold Charting Brand (2003), Diffusion Maps Coifman and Lafon (2006), Hessian Eigenmaps (HLLE) Donoho and Grimes (2003), Local Tan-
gnet Space Alignment (LTSA) [Zhang and Zha (2004)], Maximum Variance Unfolding (MVU) [Weinberger et al. (2004)], and many others see Van der Maaten et al. (2008); Cayton (2005); Perrault-Joncas and Meila (2012). Locally Linear Embedding (LLE) [Roweis and Saul (2000)] uses local symmetries of linear reconstructions to find non-linear structure in high dimensional data. We first build a neighborhood graph of all the points. Each point in lower dimensional space is estimated using its neighbors. Other variant of LLE algorithm are Hessian Locally-Linear Embedding (Hessian LLE) [Wang (2011)], Modified Locally-Linear Embedding (MLLE) [Zhang and Wang (2006)]. Laplacian Eigenmaps (LE) [Belkin and Niyogi (2001)] is another technique where uses spectral decomposition and preserve local property of points. Stochastic Neighbor Embedding (SNE) [Hinton and Roweis (2002)] treat distances between points as a probability where points close to each other have higher probability than far away points. Maximum Variance Unfolding (MVU) [Weinberger and Saul (2006)] uses semidefinite programming to reduce the dimensional of the data. This algorithm works based on the fact that in order to unfold the manifold we need to pull out the points as far away from each other. Some of these algorithms assume that $f$ above is an isometry, in which case $f$ may be recovered up to a rigid motion under some circumstances. For example, Isomap and MVU require that $f(\mathcal{M})$ is convex [Bernstein et al. (2000); Arias-Castro and Pelletier (2013)].

In this Thesis we address the more general situation where the surface $\mathcal{M}$ may self-intersect, a situation that may arise, for example, in speech recognition [Brand (1999)]. We propose a variant of landmark Isomap [Silva and Tenenbaum (2002)] for solving this problem, where the geodesic distances are estimated using constrained shortest-path distances in a neighborhood graph. The algorithm performs well on simulated examples and is shown to succeed at finding an embedding up to a rigid motion when $\mathcal{U}$ is convex and $\psi$ is a local isometry, generalizing what Isomap can achieve when $\psi$ is a (global) isometry [Bernstein et al. (2000)].
2.1.1 Isomap

In this section, we provide a detailed explanation of the Isomap algorithm. Isomap has been used for dimensionality reduction since the last decade. This algorithm is closely related to the well-known Multidimensional Scaling (MDS), which aims to detect underlying dimensions that can explain dissimilarities between objects. There are two main steps in this algorithm, which are:

1. Forming a $q$-nearest-neighbor graph of all the points in high dimensional space, and then estimating the geodesic distance between each pair of points using shortest-path distance.

2. Applying MDS to the distance matrix to find the embedded points in low dimensional space.
Algorithm 1   Isomap

**Input:** $x_1, \ldots, x_n \in \mathbb{R}^D$, $q$

**output:** $u_1, \ldots, u_n \in \mathbb{R}^d$

**Estimate** the geodesic distances (distances along a manifold) between all pairs of points using shortest-path distances on the neighborhood graph. Form a dissimilarity matrix $D$ in which its elements are the shortest-path distances between pairs.

**Apply** MDS on $D$.

Broadly speaking, Isomap is able to retrieve the parametrization of a manifold under the following assumptions [Cayton (2005)]:

1. An isometric embedding to $\mathbb{R}^D$ for manifold exists.

2. The parameter space of the manifold is convex (a geodesic between pairs of points lies within the parameter space). In other words, there is no hole in the parameter space of the manifold.

3. Every part of the manifold is sampled sufficiently.

4. The manifold is compact as a topological space.

In order to find geodesic distances, we assume that the data points are coming from a $d$-dimensional manifold embedded in $\mathbb{R}^D$ ambient space. The key assumption for Isomap is that there exists an isometric chart. This means if $x_i$, $x_j$ are points that lie on the manifold and $\delta(x_i, x_j)$ is their geodesic distance, then there is a chart $f : \mathcal{M} \rightarrow \mathbb{R}^d$ such that:

$$\|f(x_i) - f(x_j)\| = \delta(x_i, x_j).$$

If $x_i$ and $x_j$ are close to each other on a smooth manifold, then we are able to find a good approximation of $\delta(x_i, x_j)$ using the Euclidean distance between these points in the original high-dimensional space. As the distance increases, the Euclidean approximation gets worse. For the points that are far from each other in the original high-dimensional space, the corresponding Euclidean distance
can not approximate the geodesic distance accurately. Due to the above problem, estimating distances between distant points demands a new technique. To solve this problem, the Isomap algorithm forms a q-nearest-neighbor graph. Then, it uses a shortest-path algorithm on the graph to estimate the geodesic distances between points. After finding the geodesic distances between points, the Isomap algorithm looks for points whose Euclidean distances are equal to their geodesic distances. Because the manifold is isometrically embedded, there exist points that hold this condition and moreover are unique up to translation and rotation. Multidimensional Scaling [Cox and Cox (2008)] is a classical technique that could potentially find these points. The output of classical MDS Algorithm 2 gives us the position of points in $d$-dimensional space. Figure 2.2 illustrates the application of MDS on a matrix of distances for five points.

**Algorithm 2** Classical Multidimensional Scaling

<table>
<thead>
<tr>
<th>Input: $D \in \mathbb{R}^{n \times n}$ ($D_{ii} = 0, D_{ij} \geq 0$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set $B := -\frac{1}{2}QDQ$, where $B = I - \frac{1}{n}11^T$ is the centering matrix.</td>
</tr>
<tr>
<td>Compute the spectral decomposition of $B : B = U\Lambda U^T$.</td>
</tr>
<tr>
<td>Form $\Lambda_+$ by setting $[\Lambda_+]<em>{ij} := max{\Lambda</em>{ij}, 0}$.</td>
</tr>
<tr>
<td>Set $X := U\Lambda_+^{\frac{1}{2}}$.</td>
</tr>
<tr>
<td>Return $[X]_{n \times d}$.</td>
</tr>
</tbody>
</table>

![Figure 2.2: MDS on matrix of distances](image-url)
Figure 2.3 illustrates how Isomap approximates the Euclidean distance using the geodesic distance (shortest-path) on the graph.

![Figure 2.3](image)

Figure 2.3: The geodesic distances on the graph (in red) approximate the Euclidean distance (in blue) [Silva and Tenenbaum (2002)].

There are two variants of Isomap. The first variant is called C-Isomap which assigns different weights on neighborhoods depending on the density of data points. Isomap and C-Isomap have a computational complexity of $O(N^3)$, which is problematic for large datasets. To overcome this issue, the second variant selects a small set of $M$ landmark points, where $M$ is much smaller than $N$. It uses the geodesic distances between each landmark and the rest of the points. Classical MDS is applied to the $M \times M$ matrix of distances in order to find the coordinate of landmark points in $\mathbb{R}^d$. Finally, the rest points are embedded using triangulation. The computational complexity of landmark Isomap is $O(dMN \times \log(N + M^3))$.

### 2.2 Constrained shortest-path distance

Neighborhood graphs play a central role in manifold learning, exploiting the fact that smooth submanifolds are locally flat. Recall that a neighborhood graph is a graph with vertices $x_1, \ldots, x_N$. We consider two types of neighborhood structure [Maier et al. (2009)]:

- **$\varepsilon$-ball.** $x_i$ and $x_j$ are connected if $\|x_i - x_j\| \leq \varepsilon$, where $\| \cdot \|$ denotes the Euclidean norm.

- **$q$-nearest-neighbor.** $x_i$ and $x_j$ are connected if $x_j$ is among the $q$-nearest neighbors of $x_i$ (in the Euclidean metric), or vice-versa.
The central idea in this thesis is the use of constrained shortest-path distances in a neighborhood graph. The paths are constrained in order to control their smoothness. In Problem 1, the constrained shortest-path distances are used to estimate geodesic distances reliably, even when the surface self-intersects, that allows us to mimic Isomap. In Problem 2, we consider the fact that the constrained and unconstrained shortest-path distances are similar for points belonging to the same submanifold, while usually different for points belonging to different submanifolds.

The angle of an ordered triplet of points \((x, y, z)\) in \(\mathbb{R}^D\) is defined as

\[
\angle(x, y, z) = \angle(xar{y}, yar{z}) = \cos^{-1} \left( \frac{\langle z - y, z - y \rangle}{\|z - y\|} \right) \in [0, \pi],
\]

and its curvature is defined as

\[
\text{curv}(x, y, z) = \begin{cases} 
(R(x, y, z))^{-1}, & \text{if } \angle(x, y, z) < \frac{\pi}{2}, \\
\infty, & \text{otherwise,}
\end{cases}
\]

where \(R(x, y, z)\) is the radius of the circle passing through \(x, y, z\) — with \(R(x, y, z) = \infty\) if \(x, y, z\) are aligned (i.e., collinear).

**Lemma 1.** For three vertices \(x, y, z\) that are not aligned, we have

\[
R(x, y, z) = \frac{\sqrt{\|x - y\|^2 + \|z - y\|^2 + 2\|x - y\|\|z - y\| \cos \angle(x, y, z)}}{\sin \angle(x, y, z)}.
\]

In particular,

\[
\text{curv}(x, y, z) \geq \cos \left( \frac{1}{2} \angle(x, y, z) \right) \frac{1}{\|x - y\| \vee \|z - y\|}.
\]

**Proof.** Let \(x, y, z\) be \(d\)-dimensional points which form vertices of a triangle. The circumradius \(R(x, y, z)\) is then defined by

\[
R(x, y, z) = \frac{\|x - y\| \|z - y\| \|x - z\|}{2 \|(x - y) \times (y - z)\|} = \frac{\|x - z\|}{2 \sin \angle(x, y, z)}
\]

\[
= \frac{\sqrt{((x - y) - (z - y))^T((x - y) - (z - y))}}{2 \sin \angle(x, y, z)}
\]

\[
= \frac{\sqrt{\|x - y\|^2 + \|z - y\|^2 - 2\langle x - y, z - y \rangle}}{2 \sin \angle(x, y, z)}.
\]
where × denotes the cross product of two vectors.

We studied two constraints:

- **Angle constraint.** For $\theta \in [0, \pi]$, we say that a sequence of points $(z_1, \ldots, z_m)$ is $\theta$-angle constrained if the angles between successive segments are all bounded by $\theta$, meaning

  $$\angle(z_{t-1}, z_t, z_{t+1}) \leq \theta, \quad \forall t = 2, \ldots, m - 1. \tag{2.5}$$

- **Curvature constraint.** For $\kappa > 0$, we say that a sequence $(x_{i1}, \ldots, x_{im})$ is $\kappa$-curvature constrained if the (discrete) curvatures formed by pairs of successive segments are all bounded by $\kappa$, meaning

  $$\text{curv}(x_{it-1}, x_{it}, x_{it+1}) \leq \kappa, \quad \forall t = 2, \ldots, m - 1. \tag{2.6}$$

There are other definitions of discrete curvature, as discussed in (Hoffmann, 2009, Sec. 2.2). Our definition of curvature in (2.2) makes it consistent with the usual definition of curvature for smooth curves, in the following sense.

**Lemma 2.** Consider a curve $\gamma : (a, b) \rightarrow \mathbb{R}^D$ which is twice continuously differentiable and is parametrized by arc length, so that $\|\gamma(t)\| = 1$ for all $t \in (a, b)$. For $s \in (a, b)$, the curvature of $\gamma$ at $s$ is defined as $\|\gamma(s)''\|$. Then when $s$ is fixed, we have

$$\text{curv}(\gamma(r), \gamma(s), \gamma(t)) \rightarrow \|\gamma(s)''\|, \quad r, t \rightarrow s.$$  

**Proof.** Using the definition of curvature at (2.2) and the proof of Lemma 1, we have:

$$\text{curv}(\gamma(r), \gamma(s), \gamma(t)) = \frac{2 \|(\gamma(r) - \gamma(s)) \times (\gamma(t) - \gamma(s))\|}{\|\gamma(r) - \gamma(s)\| \|\gamma(t) - \gamma(s)\| \|\gamma(t) - \gamma(r)\|}.$$  

Now using the Taylor formula, the fact that $r$ and $t$ tend to $s$, and by considering $r - s = \Delta_1, t - s = \Delta_2$, we have:
curv(\gamma(r), \gamma(s), \gamma(t)) = \\
\frac{2 \left\| (\gamma(s)' \Delta_1 + \frac{1}{2} \gamma(s)'' \Delta_1^2 + o(\Delta_1^2)) \times (\gamma(s)' \Delta_2 + \frac{1}{2} \gamma(s)'' \Delta_2^2 + o(\Delta_2^2)) \right\|}{\left( \gamma(s)' \Delta_1 + \frac{1}{2} \gamma(s)'' \Delta_1^2 + o(\Delta_1^2) \right) \left( \gamma(s)' \Delta_2 + \frac{1}{2} \gamma(s)'' \Delta_2^2 + o(\Delta_2^2) \right) - \gamma(s)' \Delta_1 - \frac{1}{2} \gamma(s)'' \Delta_1^2 - o(\Delta_1^2)}
\\
\frac{2 \gamma(s)' \times \gamma(s)' \Delta_1 \Delta_2 + \frac{1}{2} \gamma(s)' \times \gamma(s)'' \Delta_1^2 \Delta_2 - \frac{1}{2} \gamma(s)' \times \gamma(s)'' \Delta_2^2 \Delta_2 + o(\Delta^2)}{\left( \gamma(s)' \Delta_1 + \frac{1}{2} \gamma(s)'' \Delta_1^2 + o(\Delta_1^2) \right) \left( \gamma(s)' \Delta_2 + \frac{1}{2} \gamma(s)'' \Delta_2^2 + o(\Delta_2^2) \right) - \gamma(s)' \Delta_1 - \frac{1}{2} \gamma(s)'' \Delta_1^2 - o(\Delta_1^2)}
\\
(2.7)

Our goal is to compute the limit of (2.7) when \Delta_1 and \Delta_2 \to 0. For the denominator, using Cauchy-Schwarz and triangle inequalities, we have:

\left\| (\gamma(s)' \Delta_1 + \frac{1}{2} \gamma(s)'' \Delta_1^2 + o(\Delta_1^2)) \right\| \leq
\\
\left( \gamma(s)' \Delta_1 + \frac{1}{2} \gamma(s)'' \Delta_1^2 + o(\Delta_1^2) \right)
\\
(\gamma(s)' \Delta_2 + \frac{1}{2} \gamma(s)'' \Delta_2^2 + o(\Delta_2^2))
\\
\gamma(s)'(\Delta_2 - \Delta_1) + \frac{1}{2} \gamma(s)''(\Delta_2^2 - \Delta_1^2) + o(\Delta_2^2) - o(\Delta_1^2)
\\
(2.8)

Since the curve is parametrized by arc length, \|\gamma(r)\| = \|\gamma(s)\| = \|\gamma(t)\| = 1. Due to the fact that the cross product of a vector with itself is zero, we have \gamma(s)' \times \gamma(s)' = 0, thus the first term in the nominator of (2.8) would be zero. On the other hand, for the curve parametrized by arc length, \gamma(s)' is orthogonal to \gamma(s)'', which means \|\gamma(s)' \times \gamma(s)''\| = \|\gamma(s)''\| \|\gamma(s)''\| = \|\gamma(s)' \times \gamma(s)''\| = 0. In (2.8), the limit of middle
term is equal to the limit of lower and upper bounds when $\Delta_1$ and $\Delta_2 \to 0$. Using all these facts, we can write the limit of (2.7) as

$$
\lim_{r,t \to s} \text{curv}(\gamma(r), \gamma(s), \gamma(t)) = \lim_{\Delta_1, \Delta_2, \Delta \to 0} \frac{2 \left\| \frac{1}{2} \gamma(s)^{''}(\Delta_1 \Delta_2^2 - \Delta_1^2 \Delta_2) + o(\Delta)^3 \right\|}{\left\| \Delta_1 \Delta_2 (\Delta_2 - \Delta_1) + o(\Delta)^3 \right\|}
$$

Using the fact that $\Delta = \max(\Delta_1, \Delta_2)$ and $o(\Delta)^3$ is negligible compare to other terms in the nominator and the denominator.

The proof sketch is shown in Figure 2.4.

**Figure 2.4:** Consistency of the discrete curvature.

**Lemma 3.** If $x, y, z$ be three $d$-dimensional points which form vertices of a triangle such that $x$ and $z$ belong to the annulus neighborhood of point $y$, then angle con-
straint \( \angle(x, y, z) < \theta \) where \( \theta < \pi/2 \) implies curvature constraint \( \text{curv}(x, y, z) < \kappa \) where \( \kappa = \frac{2\sin(\theta)}{\sqrt{\frac{\varepsilon^2}{2} (1 + \cos(\theta))}} \).

Proof. We consider \( x \) and \( z \) two points in the annulus neighborhood of point \( y \) which means that \( \|x - y\| \geq \varepsilon/2 \) and \( \|z - y\| \geq \varepsilon/2 \). Also \( \angle(x, y, z) < \theta \) and \( \theta < \pi/2 \) implies \( \cos(\angle(x, y, z)) > \cos(\theta) \) and \( \sin(\angle(x, y, z)) < \sin(\theta) \), using these results, (2.3) and (2.2) the proof is complete.

Other definitions of curvature do not always enjoy this consistency. For example, those definitions based on angle defect or Steiner’s formula [Bauer et al. (2010)] are of the form \( f(\angle(x, y, z)) \), for some continuous function \( f \), and thus are not consistent since \( \angle(\gamma(r), \gamma(s), \gamma(t)) \to 0 \) when \( \gamma \) is differentiable and \( r, t \to s \).

We note that, for an \( \varepsilon \)-ball graph, the curvature constraint implies an angle constraint. Indeed, if (2.6) holds, and in addition \( \|x_{t-1} - x_t\| \vee \|x_{t+1} - x_t\| \leq \varepsilon \), then we get that \( \angle(z_{t-1}, z_t, z_{t+1}) \leq \theta := 2 \cos^{-1}(\kappa \varepsilon) \) from (2.4).

Figure 2.5 shows three D-dimensional points \( x, y, z \) which form vertices of a triangle such that \( x \) and \( z \) belong to the annulus neighborhood of point \( y \). Under above assumption, the angle constraint \( \angle(x, y, z) < \theta \), where \( \theta < \pi/2 \) implies curvature constraint \( \text{curv}(x, y, z) < \kappa \) where \( \kappa = \frac{2\sin(\theta)}{\sqrt{\frac{\varepsilon^2}{2} (1 + \cos(\theta))}} \).

In Section 2.6 we analysis the correctness of our algorithm for simple case of two curves with intersection based on the angle constraint.

### 2.3 Approximate constrained shortest-path algorithm

Central to our methodology is the computation of curvature constrained shortest-path distances in the neighborhood graph. In order to compute constrained shortest-path distances we propose a simple modification of Dijkstra’s algorithm. Then input of the algorithm are the neighborhood graph, angle or curvature constraint, and source node. The proposed algorithm computes the constrained
shortest-path distance between the source node and all the graph points considering that if a path among three consecutive points violates the constraint then this path would be considered as infinity. In other words, there is no path from source node to that point. The main difference of our algorithm with Dijkstra’s algorithm is that we need to check the constraint during Dijkstra’s process when we consider the neighboring nodes of our current node. Suppose the current node is $x$ and nodes $y$ and $z$ are the only neighbors of node $x$ such that these two nodes in Dijkstra’s process are not visited yet and the weight from node $x$ to node $y$ is less than weight of node $x$ to node $z$. Moreover, if the path containing nodes $parent[x], x, y$ violates the constraint but the path containing the nodes $parent[x], x, z$ does not, we consider $parent[x], x, z$ as the right path. However, there is no guarantee such a path always exists, which means in this case there is no path from source node to a given point in the graph. This simple modification allows to compute the shortest-path with the same computational complexity as Dijkstra’s algorithm, $O(\Delta N \log N)$ per source point where $\Delta$ is the maximum degree of the graph. In the case of q-nearest-neighbor graph with parameter $q$, we have $(qN)/2$ edges. Therefore, the complexity would be $O(qN \log N)$. See Algorithm 3.
Algorithm 3  Constrained Dijkstra’s Algorithm

**Input:** weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, source node $s$, constraint on pairs of edges $\sim$.

**Output:** constrained shortest-path distances from $s$ to all other nodes.

Initialize $p = s$, $t = s$ and $m = 0$.

For $i \in \mathcal{V} \setminus \{s\}$: $\text{dist}[i] = \infty$, $\text{cost}[i] = \infty$, $\text{parent}[i] = $ undefined, $\text{temp}[i] = 0$, $\text{path}[i] = []).$

for $j = 1$ to $N$ do
  $\text{dist}[i] = w_{ti}$ for each $i \in \text{neighbors}(t)$.

  for $i = 1$ to $N$ do
    if $\text{dist}[i] + m < \text{cost}[i]$ then
      if $\angle(i, t, p) < \theta$ or $t == s$ then
        $\text{parent}[i] = t$ and $\text{cost}[i] = \text{dist}[i] + m$.
      end if
    end if
  end for

$I$ is the vertex in $\mathcal{G}$ with min $\text{cost}[i]$. Exclude $I$ from the next search.

if $\text{parent}[I] = \text{defined}$ then
  Update path[I] by appending vertex $I$ to the end of path[parent[I]].
end if

$\text{dist}[i] = \infty$ for all $i = 1, ..., N$.

$t = I$ , $p = \text{parent}[t]$.

if $p = \text{undefined}$ then
  break
end if

end for

2.4 Exact constrained shortest-path algorithm

The exact constrained Shortest-Path Algorithm can be implemented by applying the classical Dijkstra’s algorithm to the graph $\mathcal{G}_c$ with the same nodes and edges of the original graph $\mathcal{G}$, where two edges $(i, j)$ and $(j, k)$ in $\mathcal{G}$ are neighbors in $\mathcal{G}_c$ if
Algorithm 4  Build a new graph where the nodes of new graph are the edges of original graph

**Input:** weighted undirected graph \( \mathcal{G} = (\mathcal{V}, \mathcal{E}, w) \), constraint on pairs of edges \( \sim \).

**Output:** Edge graph.

Form a new undirected graph \( \mathcal{G}_e = (\mathcal{V}_e, \mathcal{E}_e, w_e) \) where \( \mathcal{V}_e \), nodes in the new graph, are the edges of original graph \( \mathcal{G} \).

for all \( x \in \mathcal{V} \) and all pairs \( y, z \in \text{neighbors}(x) \) do

if \( \angle(y, x, z) < \text{constraint} \) then

\( x, z \in \mathcal{V}_e \) and \( x, y \in \mathcal{V}_e \) are two connected nodes in \( \mathcal{G}_e \) with weight 

\[ w(y, z) = \frac{w(x, z) + w(x, y)}{2} \]

where \( w(y, z) \in w_e \).

end if

end for

Algorithm 5  Constrained shortest-path algorithm

**Input:** weighted undirected graph \( \mathcal{G} = (\mathcal{V}_e, \mathcal{E}_e, w_e) \), source node \( s \), destination node \( d \).

**Output:** constrained shortest-path distance from \( s \) to \( d \).

Add a source nodes \( s_e \) to \( \mathcal{V}_e \) and also add a set of edges between \( s_e \) and all the nodes in graph \( \mathcal{G}_e \) of form \( s, i \) where \( i \in \mathcal{G} \) and \( w(s, s_e) = \frac{1}{2}w(s, i) \).

Add a destination nodes \( d_e \) to \( \mathcal{V}_e \) and also add a set of edges between \( d_e \) and all the nodes in graph \( \mathcal{G}_e \) of form \( d, i \) where \( i \in \mathcal{G} \) and \( w(d, d_e) = \frac{1}{2}w(d, i) \).

Apply regular Dijkstra’s algorithm to the undirected graph \( \mathcal{G}_e = (\mathcal{V}_e, \mathcal{E}_e, w_e) \) with source node \( s_e \) and destination node \( d_e \).
2.4.1 Complexity of the exact constrained shortest-path algorithm

The time complexity of Dijkstra’s algorithm on the new graph is \( O(\log(N_e)E_e) \), where \( N_e \) and \( E_e \) are the number of nodes and edges in the new edge graph \( G_e \), respectively, and \( N \) and \( E \) are the number of nodes and edges in the original graph, respectively. For \( q \)-nearest-neighbor graph with parameter \( q \) we have \( (qN)/2 \) edges and \( N_e = (qN)/2 \) is the number of nodes in \( G_e \), the maximum number of edges in this graph for every node happens when every node is connected to all of its neighbors which is \( q \). Hence, the total number of edges for the new graph in the worst case (no constraint on the path) would be \( E_e = (((qN)/2)q)/2 \), which gives the complexity \( O(\log((qN)/2)(((qN)/2)q)/2) = O((q^2N/4)\log(qN/2)) \). Approximately, the complexity for the single source and single destination is \( O(Nq^2) \).
2.5 Constrained shortest-path algorithm for manifold learning

Having introduced the constrained shortest-path distances, we propose our manifold learning algorithm based on landmark Isomap [Silva and Tenenbaum (2002)], namely smoothness-constrained Isomap, where geodesic distances are estimated using constrained shortest-path distances in a neighborhood graph. The algorithm is given in Algorithm 6. This algorithm can learn self-intersect manifold and normal manifold successfully. See Figure 2.7. For the self intersecting manifold, there is a post-processing step to keep up spurious points. Those are the points on or very close to the intersection. We find the number of points inside the ε neighborhood of each point and compute the average number over all the points. Next, we compute the number of neighbors inside ε neighborhood of every point on the unfolded surface. If the number of neighbors is less than 85% of the average number then we remove that point from the unfolded surface.

Algorithm 6 Smoothness-Constrained Isomap

Input: data \((x_i)\); parameters \(q, L, \theta\)

Build \(q\)-nearest-neighbor graph

Choose \(L\) landmarks at random

for \(\ell = 1\) to \(L\) do

For each landmark \(\hat{x}_\ell\), compute the \(\theta\)-constrained shortest-path distance to all the data points \(x_i\) using Algorithm 3, obtaining \(\text{dist}_G(\hat{x}_\ell, x_i)\) for \(i \in [N]\).

end for

Apply MDS to the distance matrix \(\text{dist}(\hat{x}_\ell, \hat{x}_{\ell'}) : \ell, \ell' \in [L]\).

Embed the remaining points using a triangulation.

2.6 Theoretical support

To obtain theoretical guarantees, we need to extend the assumptions Isomap needs in order to recover an isometric embedding [Bernstein et al. (2000)]. Specifically, assume that we have a surface \(S = \psi(\Omega)\), where \(\Omega\) is a compact convex set of
$\mathbb{R}^d$ with non-empty interior, and $\psi : \Omega \rightarrow \mathbb{R}^D$ is twice continuously differentiable and a local isometry.

For an integer $m \geq 1$ and a point $u \in \Omega$, let $D^m_u g$ denote the differential form of $\psi$ of order $m$ at $u$. In particular, when $\psi = (\psi_1, \ldots, \psi_D)$, then $D_u \psi = (\partial_j \psi_i(u)) \in \mathbb{R}^{D \times d}$ (the Jacobian matrix) and $D^2_u \psi = (\partial_k \partial_j \psi_i(u)) \in \mathbb{R}^{D \times D \times d}$, where $\partial_j$ denotes the partial derivative with respect to the $j$th variable. By assumption, $D_u \psi$ and $D^2_u \psi$ exists and are continuous on $\Omega$.

The assumption that $\psi$ is a local isometry can be specified in several ways. Here it means that, for all $u \in \Omega$, $D_u \psi$ is an orthogonal matrix (i.e., its $d$ column vectors are orthonormal in $\mathbb{R}^D$). To be more specific, we define the following two metrics on $\mathcal{S}$:

- **Intrinsic metric.** This is the metric inherited from the ambient space $\mathbb{R}^D$. 

---

Figure 2.7: Simulated data illustrating the problem of manifold learning. Left: input data; Right: output from our method.
For \(x, x' \in \mathcal{S}\), it is defined as follows:

\[
\delta_{\mathcal{S}}(x, x') = \inf \left\{ T : \exists \vartheta : [0, T] \rightarrow \mathcal{S}, \text{ 1-Lipschitz, with } \vartheta(0) = x \text{ and } \vartheta(T) = x' \right\}.
\] (2.9)

(A function \(\vartheta : [0, T] \rightarrow \mathbb{R}^D\) is 1-Lipschitz if \(\|\vartheta(t) - \vartheta(t')\| \leq |t - t'|\) for all \(t, t' \in [0, T]\).) Since our assumptions imply that \(\mathcal{S}\) is compact, the infimum in (2.9) is attained, and in fact, \(\delta_{\mathcal{S}}(x, x')\) is the length of the shortest continuous path on \(\mathcal{S}\) passing through \(x\) and \(x'\).

**Image metric.** This is the metric image of the Euclidean metric on \(\mathbb{R}^d\) via the parametrization function \(\psi\). For \(x, x' \in \mathcal{S}\), it is defined as follows:

\[
\delta_{\psi}(x, x') = \inf \left\{ \|u - u'\| : u, u' \in \Omega, \text{ with } \psi(u) = x \text{ and } \psi(u') = x' \right\}.
\] (2.10)

(In general, \(\|u - u'\|\) would be replaced by \(\delta_{\Omega}(u, u')\), but they both coincide here since \(\Omega\) is assumed to be convex.)

The following could be taken as a (useful) definition of local isometry in our context.

**Lemma 4.** There is \(\varepsilon_0 > 0\) such that

\[
\delta_{\psi}(\psi(u), \psi(u')) = \|u - u'\|, \quad \forall u, u' \in \Omega : \|u - u'\| \leq \varepsilon_0.
\] (2.11)

We define two shortest-path distances as below:

**Unconstrained shortest-path distance:** Given \(x_1, \ldots, x_N\), and leaving them implicit, let \(\mathcal{G}_\varepsilon\) denote their \(\varepsilon\)-ball graph. For \(x, x' \in \mathcal{S}\), define the corresponding unconstrained shortest-path distance as:

\[
\Delta_{\varepsilon}(x, x') = \inf \|x - x_{i_1}\| + \|x_{i_1} - x_{i_2}\| + \cdots + \|x_{i_{\ell-1}} - x_{i_{\ell}}\| + \|x_{i_{\ell}} - x'\|,
\] (2.12)

\[
\max \left[ \|x - x_{i_1}\|, \|x_{i_1} - x_{i_2}\|, \cdots, \|x_{i_{\ell-1}} - x_{i_{\ell}}\|, \|x_{i_{\ell}} - x'\| \right] \leq \varepsilon.
\] (2.13)

where the infimum is over all possible tuples of indices \(i_1, \ldots, i_{\ell} \in [N], \ell \geq 1\).

**Constrained shortest-path distance:** Given \(x_1, \ldots, x_N\), and leaving them
implicit, let $G_{\varepsilon}$ denote their $\varepsilon$-ball graph. For $x, x' \in S$, define the corresponding constrained shortest-path distance as:

$$\Delta^\kappa_{\varepsilon}(x, x') = \inf \|x - x_{i_1}\| + \|x_{i_1} - x_{i_2}\| + \cdots + \|x_{i_{\ell - 1}} - x_{i_{\ell}}\| + \|x_{i_{\ell}} - x'\|,$$  \hspace{0.5cm} (2.14)

Subject to:

$$\max \left[ \|x - x_{i_1}\|, \|x_{i_1} - x_{i_2}\|, \cdots, \|x_{i_{\ell - 1}} - x_{i_{\ell}}\|, \|x_{i_{\ell}} - x'\| \right] \leq \varepsilon. \hspace{0.5cm} (2.15)$$

and

$$\max \left[ \text{curv}(x, x_{i_1}, x_{i_2}), \text{curv}(x_{i_1}, x_{i_2}, x_{i_3}), \ldots, \text{curv}(x_{i_{\ell - 2}}, x_{i_{\ell - 1}}, x_{i_{\ell}}), \text{curv}(x_{i_{\ell - 1}}, x_{i_{\ell}}, x') \right] \leq \kappa. \hspace{0.5cm} (2.16)$$

where the infimum is over all possible tuples of indices $i_1, \ldots, i_{\ell} \in [N], \ell \geq 1$.

Our central result is the following. As the sampling gets denser and denser, and the neighborhood radius of the graph is chosen appropriately, the unconstrained shortest-path distance $\Delta_{\varepsilon}(x, x')$ approaches the intrinsic distance $\delta_S(x, x')$, while the constrained shortest-path distance $\Delta^\kappa_{\varepsilon}(x, x')$ approaches the image distance $\delta_\psi(x, x')$. Moreover, the convergence is uniform if the sampling is uniformly dense. We quantify the density of the sampling using:

$$\varepsilon_x = \sup_{x \in S} \min_i \|x - x_i\|. \hspace{0.5cm} (2.17)$$

In terms of sampling, it is standard to generate points uniformly at random in $S$, which is identical to sampling $u_1, \ldots, u_N$ uniformly at random from $\Omega$ and then defining $x_i = \psi(u_i)$. (This is because $\psi$ is a local isometry.)

Bernstein et al. (2000) or Arias-Castro and Pelletier (2013, Sec. 3.2) show that when $x_1, \ldots, x_N$ are sampled uniformly at random on $S$ and under same assumption on $\Omega$ and $\psi$, we have $\varepsilon_x \leq \frac{1}{\beta} \left( \frac{\log N}{N} \right)^{1/D}$ with probability less than $1 - e^{-\beta N}$.

Assume that we have a surface $S = \psi(\Omega)$, where $\Omega$ is a compact convex set of $\mathbb{R}^d$ with non-empty interior, and $\psi: \Omega \to \mathbb{R}^D$ is twice continuously differentiable and a local isometry. Given sample points $x_1, \ldots, x_N \in S$, take $\varepsilon, \varepsilon_x, \psi$ and limited number of points at the intersection, then we can show

$$\frac{\Delta_{\varepsilon}(x, x')}{\delta_S(x, x')} \hspace{0.5cm} (2.18)$$
Figure 2.8: The proof sketch that shortest-path on the graph approximates the intrinsic metric.

and

$$\frac{\Delta^S_\varepsilon(x, x')}{\delta_\psi(x, x')}$$

are bounded from above and below. Figure 2.8 shows the proof sketch.

Although we do not provide a rigorous proof which needs more tools and assumptions on the manifold, we explain our procedure for finding the upper bound and lower bound for (2.18).

Let $T = \delta_S(x, x')$ be the length of an arc on the surface connecting $x$ and $x'$ and $\delta_S(x, x') \geq \varepsilon$, then split this arc into $q$ arcs of length $T_1 = \varepsilon/2$ and one arc of length $T_{q+1} \leq T_1$. We study the path $x, \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_{q-1}, \tilde{x}_q, x'$ on the surface where $\delta_S(\tilde{x}_j, \tilde{x}_{j+1}) = T_1$ for $j = 1, \ldots, q - 1$, $\delta_S(x, \tilde{x}_1) = T_1$ and $\delta_S(\tilde{x}_q, x') = T_{q+1}$. Suppose $x_{ij}$ is the closet point on the graph $G_\varepsilon$ to the point $\tilde{x}_j$ on the surface $S$.

Using density of sampling in (2.17) we have $\|x_{ij} - \tilde{x}_j\| \leq \varepsilon_x$.

Using above facts for $j = 1, \ldots, q - 1$ we get:
\[\|x_{i+1} - x_i\| \leq \|x_{i+1} - \bar{x}_{j+1}\| + \|\bar{x}_{j+1} - \bar{x}_j\| + \|x_j - \bar{x}_j\| \leq\]

\[\|x_{i+1} - \bar{x}_{j+1}\| + \delta_S(\bar{x}_{j+1}, \bar{x}_j) + \|x_j - \bar{x}_j\| \leq \varepsilon_x + T_1 + \varepsilon_x = 2\varepsilon_x + T_1\]

Also for the first piece:

\[\|x - x_i\| \leq \|x - \bar{x}_1\| + \|x_i - \bar{x}_1\| \leq\]

\[\delta_S(x, \bar{x}_1) + \|x_i - \bar{x}_1\| \leq T_1 + \varepsilon_x\]

And for the last piece:

\[\|x' - x_i\| \leq \|x' - \bar{x}_q\| + \|x_i - \bar{x}_q\| \leq\]

\[\delta_S(x', \bar{x}_q) + \|x_i - \bar{x}_q\| \leq T_{q+1} + \varepsilon_x\]

Combining all these results and using \(T_1 = \varepsilon/2\) and the fact that \(\Delta_\varepsilon(x, x')\) in (2.12) is the shortest-path on the graph that connects \(x\) and \(x'\) we have:

\[\Delta_\varepsilon(x, x') \leq \|x - x_i\| + \|x_i - x_{i_2}\| + \cdots + \|x_{i_{q-1}} - x_{i_q}\| + \|x_{i_q} - x'\| \leq\]

\[(q - 1)(T_1 + 2\varepsilon_x) + T_1 + \varepsilon_x + T_{q+1} + \varepsilon_x =\]

\[qT_1 + T_{q+1} + 2q\varepsilon_x =\]

\[T + 2q\varepsilon_x \leq T + 4T\varepsilon_x/\varepsilon = T(1 + 4\varepsilon_x/\varepsilon) = \delta_S(x, x')(1 + 4\varepsilon_x/\varepsilon)\]

By choosing \(\lambda_2 = 4\varepsilon_x/\varepsilon\) we achieve the upper bound. For the case where \(T = \delta_S(x, x') \leq \varepsilon\) using the same analysis we get:

\[\Delta_\varepsilon(x, x') \leq \delta_S(x, x') + 2\varepsilon_x = T(1 + 2\varepsilon_x/T)\]

Choosing \(\lambda_2 = 2\varepsilon_x/T\) we achieve the upper bound.
In order to find the lower bound we use the central assumption given for Isomap [Bernstein et al. (2000)], called the regularity assumption indicates that there is a non-decreasing function $c : [0, \infty) \to [0, \infty)$ such that $c(r) \to 0$ when $r \to 0$, where, for all $x_1, x_2 \in S$,

$$\delta_S(x_1, x_2) \leq (1 + c(\|x_1 - x_2\|)) \|x_1 - x_2\|.$$ 

Arias-Castro and Pelletier (2013) gave a proof for this assumption. This condition holds when $S$ is a compact, smooth submanifold with smooth boundary and to tubular neighborhood of such a set.

Let $\Delta_\varepsilon(x, x')$ be the (unconstrained) shortest-path distance that satisfies (2.15) and $x_i, x_{i1}, \ldots, x_{iq}, x'$ are the points in this path. We consider a self intersecting surface. We split

$$x, x_{i1}, x_{i2}, \ldots, x_{iq}, x'$$

on the surface in two sets:

$$x, x_{i1}, \ldots, x_{im} \cup x_{im+k}, \ldots, x_{iq}, x'$$

which is not within $\varepsilon/2$ of intersection and

$$x_{im+1}, \ldots, x_{im+k-1}$$

which is within $\varepsilon/2$ of intersection. On the first set since the regularity condition holds we get:

$$\delta_S(x, x_{i1}) + \cdots + \delta_S(x_{im-1}, x_{im}) + \delta_S(x_{im+k}, x_{im+k+1}) + \cdots + \delta_S(x_{iq}, x') \leq (\|x - x_{i1}\| + \cdots + \|x_{im-1} - x_{im}\| + \|x_{im+k} - x_{im+k+1}\| + \cdots + \|x_{iq} - x'\|)

(1 + c(max \left\{ \|x - x_{i1}\|, \ldots, \|x_{im-1} - x_{im}\|, \|x_{im+k} - x_{im+k+1}\|, \ldots, \|x_{iq} - x'\| \right\}))$$
The regularity assumption is not valid on the second set.

Assuming:

\[ h = \max \left\{ \|x - x_1\|, \ldots, \|x_{im} - x_i\|, \|x_{im+k} - x_{im+k+1}\|, \ldots, \|x_i - x'\| \right\} \]

and \( \lambda_1 = c(h) \), now using the fact that:

\[ \delta_S(x_{im}, x_{im+1}) + \cdots + \delta_S(x_{im+k-1}, x_{im+k}) \leq k\varepsilon/2 \]

we have:

\[ \delta_S(x, x') \leq \delta_S(x, x_1) + \cdots + \delta_S(x_{im-1}, x_{im}) + \delta_S(x_{im}, x_{im+1}) + \cdots + \delta_S(x_{im+k-1}, x_{im+k}) + \delta_S(x_{im+k}, x_{im+k+1}) + \cdots + \delta_S(x_{q}, x') \leq \]

\[ \left( \|x - x_1\| + \cdots + \|x_{im-1} - x_{im}\| + \|x_{im} - x_{im+1}\| + \cdots + \|x_{im+k-1} - x_{im+k}\| + \|x_{im+k} - x_{im+k+1}\| + \cdots + \|x_q - x'\| \right) \left( 1 + \lambda_1 \right) + k\varepsilon/2 \]

\[ \leq \left( \|x - x_1\| + \cdots + \|x_{im-1} - x_{im}\| + \|x_{im} - x_{im+1}\| + \cdots + \|x_{im+k} - x_{im+k+1}\| + \cdots + \|x_q - x'\| \right) (1 + \lambda_1) + k\varepsilon/2 = \Delta \varepsilon(x, x')(1 + \lambda) + k\varepsilon/2 \]

Since \( h \leq 2\varepsilon + \varepsilon/2 \), when \( \varepsilon \to 0 \) then \( \varepsilon_x \to 0 \) and \( h \to 0 \). Assuming \( \lambda_1 = c(h) \to 0 \). Using the inequality:

\[ \frac{\delta_S(x', x')}{1 + \lambda_1} - \frac{k\varepsilon/2}{1 + \lambda_1} \leq \Delta \varepsilon(x, x') \quad (2.20) \]

when \( \lambda_1 \to 0 \):

\[ 1 - \lambda_1 \leq \frac{1}{1 + \lambda_1} \]

and

\[ (1 - \lambda_1) - \frac{k\varepsilon/2}{\delta_S(x, x')} \leq \frac{\Delta \varepsilon(x, x')}{\delta_S(x, x')} \]
Using:

$$\delta_S(x, x') = q \frac{\varepsilon}{2} + T_{q+1} \geq q \frac{\varepsilon}{2}$$

we have:

$$\left(1 - \lambda_1\right) - \frac{k}{q} \leq \frac{\Delta_\varepsilon(x, x')}{\delta_S(x, x')}$$

When $k << q$ (the number of points at the intersection is negligible compare to the number of points on the arc) the lower bound at (2.18) is established. We can extend this idea of finding the lower bound to the case of manifolds with multiple self intersections.

A similar approach can be taken for (2.19), but we also need to show that the set of points $x, \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_{q-1}, \tilde{x}_q, x'$ satisfy the curvature constraint.

When $\psi$ is injective, it is a global isometry and $\mathcal{S}$ is a bona fide manifold, and the metrics $\delta_S$ and $\delta_\psi$ would be the same. Working in that specific case, Bernstein et al. (2000) proved an approximation result similar to (2.18). As we see, this result still holds even when $\psi$ is only a local isometry, but this is not useful for solving manifold learning problem, where the goal is to recover the image distances (before applying MDS). This is where the constrained shortest-path distances come in, and the approximation (2.19) is exactly what is needed to justify the performance of Algorithm 6.

We also mention that a similar result holds for $q$-nearest-neighbor graph, when $q$ is chosen appropriately. When the points are sampled uniformly the $q$-nearest-neighbor graph is close to the $\varepsilon$-ball graph using the fact $k \asymp N\varepsilon^d \gg \log N$, where $N\varepsilon^d$ is the order of the degrees.

### 2.7 Implementation and numerical experiments

We compare our algorithm with state-of-the-art manifold learning techniques for learning a self-intersect manifold. Our method unfolds the curve successfully, while the other methods fail. See Figure 2.9.

In another experiment shown in Figure 2.10 we use a curved surface with a hole in 3D and compare our algorithm with state-of-the-art manifold learning
techniques. Our method unfolds the surface whereas in other methods the original surface is distorted because of scale and translation of points in lower dimensional space.
Figure 2.10: Results of manifold learning algorithms on a non-compact manifold with a hole. (a) non-compact manifold; (b) our method; (c) LLE; (d) HLLE; (e) LTSA; (f) MVU. Our algorithm preserves the similarity of manifold.
Chapter 3

Manifold clustering

3.1 Manifold clustering problem

We use similar ideas to tackle the related problem of multi-manifold clustering.

**Problem 2** (Manifold clustering). Given a sample $x_1, \ldots, x_n \in \mathbb{R}^D$ sampled from $S_1 \cup \cdots \cup S_K$, where for each $k$, $S_k$ is a smooth, but possibly self-intersecting manifolds, label each point according to the manifold it belongs to. Figure 3.1.

Subspace clustering (where the surfaces are affine) is an important special case with a number of applications in computer vision [Vidal (2011); Ma et al. (2008)]. It has been generalized to the case of higher-order algebraic surfaces [Ho et al. (2003); Epstein et al. (1995)]. The literature on general multi-manifold clustering is smaller. Most methods are designed for the case where the Manifolds do not intersect [Polito and Perona (2001); Ng et al. (2002); Elhamifar and Vidal (2011)], while others work only when the surfaces that intersect have different intrinsic dimension or density [Gionis et al. (2005); Haro et al. (2007)]. The method of Arias-Castro et al. (2011) is only able to separate intersecting curves. Methods that purposefully aim at resolving intersections are even fewer. Souvenir and Pless (2005) implement some variant of K-means where the centers are surfaces. Guo et al. (2007) propose to minimize a (combinatorial) energy that includes local orientation information, using a tabu search. The state-of-the-art lies with methods based on local PCA. An early proposal was the elaborate multi scale spectral
method of Kushnir et al. (2006), while the clustering routine of Goldberg et al. (2009) — developed in the context of semi-supervised learning — inspired the works of Wang et al. (2011), Gong et al. (2012) and Arias-Castro et al. (2013). See also the related method of Wang et al. (2014).

![Figure 3.1: Simulated data illustrating the problem of multi-manifold clustering. Left: 3D data. Right: output from our method.](image)

Of course, the labels are determined up to a permutation of \{1, \ldots, K\}. We propose a markedly different approach based on comparing unconstrained and constrained shortest-path distances in a neighborhood graph. The resulting algorithm is comparable to the state-of-the-art methods on standard synthetic data. However, it is simpler and faster, requires less parameters than the other methods suggested in the literature, and can also handle surfaces of different intrinsic dimensions seamlessly, instead of requiring that the surfaces be of same dimension. Moreover, it comes with theoretical guarantees; we mention that Arias-Castro et al. (2013); Wang et al. (2014) obtain similar guarantees, but the analysis in these papers is substantially more complicated. The method is described in Algorithm 9.

### 3.2 Algorithms for multi-manifold clustering

In this section, we elaborate on three of them, namely K-Manifolds (KM) Souvenir and Pless (2005), Spectral Curvature Clustering (SCC) Chen and Lerman (2009), and Spectral Multi-Manifold Clustering (SMMC) Wang et al. (2011) of-
ffered for Multi-Manifold Clustering. Furthermore, we use them as benchmarks in our experiments. our choice was dictated by performance, code availability and relevance to our particular setting.

3.2.1 K-Manifolds

Souvenir and Pless (2005) suggest an algorithm that mimics K-means, replacing centroid points with centroid submanifolds. The method starts like Isomap by building a neighborhood graph and computing shortest-path distances within the graph. After randomly initializing a $K$-by-$n$ weight matrix $W = (w_{ki})$, where $w_{ki}$ represent the probability that point $i$ belongs to the $k_{th}$ cluster, it alternates between an M-Step and an E-Step. In the M-Step, for each $k$, the points are embedded in $\mathbb{R}^K$ using a weighted variant of multidimensional scaling using the weights $(w_{ki} : i = 1, \ldots, n)$. In the E-Step, for each $k$ and $i$, the normal distance of point $x_i$ to the cluster $k$ is estimated as

$$\delta_{ki} = \frac{\sum_j w_{kj}(d(x_i, x_j) - d_c(x_i, x_j))}{\sum_j w_{kj}},$$

where $d(x_i, x_j)$ denotes the shortest-path distance in the neighborhood graph and $d_k(x_i, x_j)$ denotes the Euclidean distance in the $k^{th}$ embedding, between points $x_i$ and $x_j$. The weights are then updated as $w_{ki} \propto \exp(-d_{ki}^2/\sigma^2)$ such that $\sum_k w_{ki} = 1$ for all $i$, where $\sigma^2$ is chosen automatically Algorithm 7.

3.2.2 Spectral multi-manifold clustering

Wang et al. (2011) is a spectral method using a dissimilarity that factors in the Euclidean distance and the discrepancy between the local orientation of the data. The surfaces are assumed to be of same dimension $d$ known to the user. First, a mixture of probabilistic principal component analyzers Tipping and Bishop (1999) are fitted to the data, approximating the point cloud by a union of $d$-planes. This is used to estimate the tangent subspace at each data point. The dissimilarity between two points is then an increasing function of their Euclidean distance and the principal angles between their respective affine subspaces. These dissimilarities
Algorithm 7  K-Manifolds

Input: $x_1, ..., x_n$, K number of clusters.

Build a neighborhood graph and computing shortest-path distances within the graph.

Randomly initializing a $K$-by-$n$ weight matrix $W = (w_{ki})$, where $w_{ki}$ represent the probability that point $i$ belongs to the $k^{th}$ cluster.

Build a neighborhood graph and computing shortest-path distances within the graph.

It alternates between an M-Step and an E-Step.

In the M-step, for each $c$, the points are embedded in $R^K$ by applying a node-weighted MDS where seeks to minimize the following function (In total $k$ embedding): $\sum_{ij} w_{ki} w_{kj} (\|y_i - y_j\|^2 - D(i,j))^2$ where $D$ is the distance matrix between nodes in the graph and $y_j$ is a K-vector.

In the E-Step, for each $k$ and $i$, the normal distance of point $x_i$ to the cluster $k$ is estimated as $\delta_{ki} = \frac{\sum_j w_{kj}(d(x_i,x_j)-d_k(x_i,x_j))}{\sum_j w_{kj}}$ where $d(x_i,x_j)$ denotes the shortest-path distance in the neighborhood graph and $d_k(x_i,x_j)$ denotes the Euclidean distance in the $k^{th}$ embedding, between points $x_i$ and $x_j$.

The weights are then updated as $w_{ki} \propto \exp(-\delta_{ki}^2/\sigma^2)$ such that $\sum_K w_{ki} = 1$ for all $i$, where $\sigma^2$ is chosen automatically.
are fed to the spectral graph partitioning method of Ng et al. (2002). It is on the
analysis that spectral clustering methods are able to work well when the affinity
values of the points belonging to different clusters are relatively low, which is
able to handle intersections. In this model, the data are assumed to lie on or
close to multiple smooth low-dimensional manifolds, where some data manifolds
are separated but some are intersected. Then, local geometric information of
the sampled data is incorporated to construct a suitable affinity matrix. Finally,
spectral Multi-Manifold method is applied to this affinity matrix to group the data. Spectral
Multi-Manifold Clustering is shown in Algorithm 8.

**Algorithm 8 Spectral Multi-Manifold Clustering**

*Input:* $x_1, ..., x_n$, $K$ number of clusters, dimension of the manifolds $d$, number
of mixture models $M$, number of neighbors $q$, tuning parameter $o$.

*Train* $M d$-dimensional local linear manifolds by using mixture of probabilis-
tic principal component analyzers (MPPCA) to approximate the underlying
manifolds.

*Determine* the local tangent space of each point.

*Compute* pairwise affinity between two local tangent.

*Compute* the affinity matrix $W \in \mathbb{R}^{n \times n}$ where the similarity between two
neighboring points is a decreasing function of the principal angles between their
respective affine subspaces and is zero otherwise.

*Compute* the diagonal matrix $E$ with $E_{ii} = \sum_j w_{ij}$.

*Extract* the first $k$ generalized eigenvectors $u_1, ..., u_K$ of $(E - W)u = \lambda Eu$.

*Apply* $K$-means to cluster the row vectors of $U$ in $\mathbb{R}^K$.

### 3.2.3 Spectral curvature clustering

Consider the problem of modeling a collection of data points with a union of
subspaces, as illustrated in Figure 3.2. Let $\{x_j \in \mathbb{R}^D\}_{j=1}^N$ be a given set of points
drawn from an unknown union of $n \geq 1$ linear or affine subspaces $\{S_i\}_{i=1}^N$ of
unknown dimensions $d_i = dim(S_i), 0 < d_i < D, i = 1, ..., n$. The subspace can be
describe as:
Figure 3.2: A set of sample points in $\mathbb{R}^3$ drawn from a union of three subspaces.

$$s_i = \{ x_j \in \mathbb{R}^D : x = \mu_i + U_i y \}$$

where $\mu_i \in \mathbb{R}^D$ is an arbitrary point in subspace $S_i$ ($\mu_i = 0$ for linear subspaces), $U_i \in \mathbb{R}^{D \times d_i}$ is a basis for subspace $S_i$ and $y \in \mathbb{R}^{d_i}$ is a low-dimensional representation for point $x$. The goal of subspace clustering is to find the number of subspaces $n$, their dimensions $\{d_i\}_{i=1}^n$, the subspace bases $\{U_i\}_{i=1}^n$, the points $\{\mu_i\}_{i=1}^n$ (in the case of affine subspaces), and the segmentation of the points according to the subspaces. Majority of subspace clustering methods face challenges like, there is a strong coupling between data segmentation and model estimation. The distribution of the data inside the subspaces is generally unknown. The relative position of the subspaces can be arbitrary. The data can be corrupted by noise, missing entries, outliers, etc. Finding a model selection criteria that favors a small number of subspaces of small dimension.

Chen and Lerman (2009) proposed a spectral method for subspace clustering — the setting where the underlying surfaces are affine. We will compare our method to theirs when the surfaces are affine, and also when the surfaces are curved. The latter is done as a proof of concept, for it will be very clear that it cannot handle curved surfaces, like any other method for subspace clustering we know of. The procedure assumes that all subspaces are of same dimension $d$, which is a parameter of the method. For each $(d + 2)$-tuple, $x_{i_1}, \ldots, x_{i_{d+2}}$, it
computes a notion of curvature $C_{i_1, \ldots, i_{d+2}}$ which measure how well approximated this $(d + 2)$-tuple is by an affine subspace of dimension $d$. After reducing the tensor $C = (C_{i_1, \ldots, i_{d+2}} : i_t = 1, \ldots, N)$ spectral graph partitioning \cite{Ng et al. (2002)} is applied.

### 3.3 Constrained shortest-path for multi-manifold clustering

Our algorithm is quite distinct from all the other methods for multi-manifold clustering we are aware of, although it starts by building a $q$-nearest-neighbor graph like many others. The idea is very simple and amounts to clustering together points that are connected by an angle-constrained path in the neighborhood graph.

**Definition.** For an angle $\theta \in [0, \pi]$, we say that a path $(x_{i_1}, \ldots, x_{i_m})$ is $\theta$-constrained if $\angle(\overrightarrow{x_{i_t-1}x_{i_t}}, \overrightarrow{x_{i_t}x_{i_{t+1}}}) \leq \theta$ for all $t = 2, \ldots, m - 1$.

The rationale is the following. Take two surfaces $S_1$ and $S_2$ intersecting at a strictly positive angle. Then for ‘most’ pairs of data points $x_{i_1} \in S_1$ and $x_{i_2} \in S_2$, a path in the graph going from $x_{i_1}$ to $x_{i_2}$ has at least one large angle between two successive edges, on the order of the incidence angle between the surfaces; while for ‘most’ pairs of data points $x_{i_1}, x_{i_2} \in S_1$, there is a path with all angles between successive edges relatively small.

To speedup the implementation, we select $M$ landmarks (with $M$ slightly larger than $K$) at random among the data points and only identify what data points are connected to what landmark via a $\theta$-constrained path in the graph. $M$ and $\theta$ are parameters of the algorithm.

Let $\xi_{i\ell} = 1$ if point $i$ and landmark $\ell$ are connected that way, and $\xi_{i\ell} = 0$ if not. We use $\xi_i := (\xi_{i\ell} : \ell = 1, \ldots, M)$ as feature vectors that we group together and cluster using hierarchical clustering with complete linkage.
3.4 Computational complexity

The algorithm is quite fast. Building a symmetric $q$-nearest-neighbor graph using cover trees [Beygelzimer et al. (2006)] takes order $O(qN \log N)$, where the implicit constant depends exponentially on the intrinsic dimensions of the surfaces and linearly on the ambient dimension $D$. The angle-constrained pathfinder routine is a simple variant of Dijkstra’s algorithm, whose implementation by Fibonacci heaps runs in $O(qN \log N)$. Hence, calling this routine once for each landmark costs $O(qMN \log N)$. Grouping the feature vectors $O(FMN)$ and then clustering them by complete linkage costs $O(F^2 \log F)$, where $F$ is the (data-dependent) number of distinct feature vectors $\xi_i$, often of the order of $K$ in our experiments.

\begin{algorithm}
\textbf{Input:} data $(x_i)$; parameters $q, K, M, \theta$

Build $q$-nearest-neighbor graph

Choose $M$ landmarks are random

\textbf{for} $i = 1$ to $n$ \textbf{do}

For each landmark $\hat{x}_\ell$, identify which points $x_i$ it is connected to via a $\theta$-constrained path in the graph, and set $\xi_{\hat{x}_\ell} = 1$ if so, and $\xi_{\hat{x}_\ell} = 0$ otherwise.

\textbf{end for}

Group and then apply hierarchical clustering to the feature vectors $\xi_1, \ldots, \xi_n$ to find $K$ clusters, where $\xi_i := (\xi_{\hat{x}_\ell} : \ell = 1, \ldots, M)$.

\end{algorithm}

3.5 Implementation and numerical experiments

3.5.1 Numerical experiments

In this section, we compare our Algorithm 9 to the methods described in previous section on various synthetic and real-world datasets.
Synthetic data

The synthetic datasets we generated are similar to those appearing in the literature. We have applied our method in 8 synthetic data including: Three Planes (TP), Two Spirals (TS1), Five Segments (FS), Dollar-Sign and Plane and Roll (DPR), Roll and Plane (RP), Cone and Plane (CP), Two Spheres (TS2), Rose Curve and Circle (RC). Figure 3.3 shows the performance of our algorithm on eight synthetic datasets. The misclustering rates for our method, and the other three methods, are presented in Table 3.1, where we see that our method achieves a performance at least comparable to the best of the other three methods on each dataset. We implemented path-based clustering using both curvature constraint and angle constraint with annuals graph. To compute the accuracy of clustering we remove a few ambiguous points close by intersection. Spectral Curvature Clustering (SCC) works well on linear manifolds (as expected) while it fails when there is curvature Figure 3.4 a. K-Manifolds fails in the more complicated examples Figure 3.4 c,d. We found that this algorithm is very slow since it has to compute the shortest-path between all the points, so that we could not apply it to some of the largest datasets. We mention that it assumes that clusters intersect, and otherwise does not work properly. Our method and Spectral Multi-Manifolds Clustering (SMMC) perform comparably on most datasets, but SMMC fails in the Rose Curve and Circle example Figure 3.4 b. We note that K-Manifold, SCC and SMMC all require that all surfaces are of same dimension, which is a parameter of these methods, why our method does not need knowledge of the intrinsic dimensions of the surfaces and can operate even when these are different.

Clustering of 2D image data

We apply our method on the COIL-20 dataset which includes 1440 gray-scale images of 20 objects shown in Figure 3.5. Each object contains 72 images taken by a camera at different angles. The original resolution of each image is $128 \times 128$. We first projected the dataset onto the top 10 principal components. In Figure 3.6 we show the output of our algorithm and those of SCC and SMMC on first 5 objects in a 3D plot where the points are represented by their first 3 principal components.
Figure 3.3: Result of our method on 8 synthetic datasets.
Table 3.1: Clustering accuracy on synthetic data.

<table>
<thead>
<tr>
<th>Data set</th>
<th>K-Manifolds</th>
<th>SCC</th>
<th>SMMC</th>
<th>PBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three Planes</td>
<td>95.1%</td>
<td>98.8%</td>
<td>99.3%</td>
<td>93.0%</td>
</tr>
<tr>
<td>Two Spirals</td>
<td>95.2%</td>
<td>54.8%</td>
<td>99.6%</td>
<td>98.2%</td>
</tr>
<tr>
<td>Five Segments</td>
<td>59.1%</td>
<td>94.9%</td>
<td>99.6%</td>
<td>98.1%</td>
</tr>
<tr>
<td>Dollar-Sign, Plane and Roll</td>
<td>46.1%</td>
<td>-</td>
<td>99.4%</td>
<td>97.5%</td>
</tr>
<tr>
<td>Roll and Plane</td>
<td>53.5%</td>
<td>-</td>
<td>95.6%</td>
<td>94.6%</td>
</tr>
<tr>
<td>Cone and Plane</td>
<td>-</td>
<td>-</td>
<td>99.3%</td>
<td>94.9%</td>
</tr>
<tr>
<td>Two Spheres</td>
<td>-</td>
<td>-</td>
<td>95.5%</td>
<td>97.9%</td>
</tr>
<tr>
<td>Rose Curve and Circle</td>
<td>62.9%</td>
<td>-</td>
<td>64.7%</td>
<td>99.4%</td>
</tr>
</tbody>
</table>

Figure 3.4: Examples where the other methods fail. (a) SCC, (b) SMMC, (c, d) K-Manifolds.

Path-Based Clustering seems closer to the ground truth. We also tested our method on the three very similar objects 3, 6 and 19. The algorithm is 99% accurate
(misclusters only 2 images out of 216) bringing a significant improvement over the state-of-the-art result of 70% reported in Wang et al. (2011). Lastly, we evaluated our method on the whole dataset obtaining an 83.6% accuracy, improving on the 70.7% accuracy reported in Wang et al. (2011). Here we used the top 20 principal components. Since in this case we have 20 different classes, we increased the number of landmarks to 100 to make sure we sampled that at least a few landmarks from each class.

![Figure 3.5: The 20 objects from the COIL-20 database.](image)

In another experiment we use a subset of the Caltech101 dataset[1] as our real data consisting 3500 images categorized in 10 different classes (see Figure 3.7). We utilized a Bag-of-words model of SIFT features to represent the content of each image. The feature vectors have the size of 128. Then, we applied PCA on the feature vectors to get smaller features with the size of 10. This was mainly done for speeding up the computations. Here again we applied the used methods and measured the clustering accuracy. However, we did this experiment with a different number of classes, namely 2, 4, 6, 8, and 10. Precisely, each time we selected K classes from the dataset, applied the methods and measured the accuracy. The clustering results for different K are presented in Table 3.2.

---

Figure 3.6: Output of three different methods on the first 5 objects of COIL-20. Points are represented by their 3 top principal components.

Figure 3.7: Some sample images of Caltech dataset
### Table 3.2: Clustering accuracy of multiple subjects for the Caltech dataset

<table>
<thead>
<tr>
<th>K</th>
<th>SCC</th>
<th>SC</th>
<th>KM</th>
<th>SMMC</th>
<th>PBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.6846</td>
<td>± 0.1417</td>
<td>0.7456</td>
<td>± 0.1569</td>
<td>0.7706 ± 0.1518</td>
</tr>
<tr>
<td>4</td>
<td>0.5694</td>
<td>± 0.0890</td>
<td>0.5621</td>
<td>± 0.1063</td>
<td>0.5598 ± 0.0996</td>
</tr>
<tr>
<td>6</td>
<td>0.4898</td>
<td>± 0.0514</td>
<td>0.4927</td>
<td>± 0.0643</td>
<td>0.4139 ± 0.0936</td>
</tr>
<tr>
<td>8</td>
<td>0.4074</td>
<td>± 0.0751</td>
<td>0.4122</td>
<td>± 0.0111</td>
<td>0.4252 ± 0.0489</td>
</tr>
<tr>
<td>10</td>
<td>0.4074</td>
<td>± 0.0751</td>
<td>0.4122</td>
<td>± 0.0111</td>
<td>0.4252 ± 0.0489</td>
</tr>
</tbody>
</table>

**Clustering of human motion sequences**

In computer vision, clustering of human motion sequences into different classes of activities performed by a subject is referred to as temporal segmentation. In this section, we test our algorithm on a sequence of video frames including different activities performed by a subject. We choose 4 mixed actions from subject 86, trial number 9 of the CMU MoCap dataset. The data consists in a temporal sequence of 62-dimensional representation of the human body via markers in $\mathbb{R}^3$. One motion sequence of 4794 frames and corresponding result of path-based multi-manifold clustering are given in Figure 3.8. Four activities are labeled from 1 to 4. We do not label the frames where the subject switches from one action to another because of the uncertainty about the true activity.
Segmentation of video sequences

In this section we consider the problem of partitioning a video sequence into different scenes. We consider the same video sequence used in Chen and Lerman (2009); Vidal et al. (2005). The video is an interview from Fox News containing 135 image frames of size $294 \times 413$; a sample is depicted in Figure 3.9. Firstly we change each RGB image frame to the gray scale intensity image, then resize it to an $74 \times 104$ image. After concatenating all pixels of each image and putting into a vector of size 7696, we construct a matrix of size $135 \times 7696$ where each row represents a frame of the original video sequence. Applying our algorithm on this matrix we get a perfect clustering (100%). We repeated the experiment, this time projecting the data onto the top 10 principal components as done in Chen and Lerman (2009); Vidal et al. (2005), obtaining a matrix of size $135 \times 10$. We still get a 100% accuracy, for an even wider range of parameters.
Figure 3.9: The first, 56th and last frame (135th) of Fox news video sequence.

![Output of SMMC and PBC](image)

Figure 3.10: Example of noisy data.

**Robustness to noise**

We speak of noise when the data points are not necessarily sampled exactly on the surface(s). This is an important issue as most (if not all) methods are quite sensitive to noise. We performed a simple experiment evaluating the sensitivity of our multi-surface clustering method (Algorithm 9). The setting is that of two intersecting curves in the plane. We report the mis-clustering error rate for our method and SMMC as a function of the noise level in Figure 3.11. The Noise model is chosen as $\alpha U[0,1]$ where we increase $\alpha$ from zero to .35. We note that the method is somewhat robust up to about 20% of noise ($\alpha = .2$), when the error rate starts to increase sharply.
Figure 3.11: Left: Two intersecting curve with added uniform noise. Right: The performance of our algorithm and SMMC as a function of noise level.

**Robust to outliers**

In another experiments to test the performances of our method in presence of outlier we draw a uniform random sampling from a unit square, then sample from a curve inside the square with a different rate. By applying Algorithm 3 to this data, we compute the longest of constrained shortest-path between all pairs as it is illustrated in Figure 3.12. That path belongs to the points on the curve. In this experiment the total number of points is 1600 with 100 of them on the curve. There are some applications in robotics for finding the correct path.

Figure 3.12: Robustness of proposed method to outliers. Left: A curve (100 points) is embedded in a crowd of outlier points (1600 points). Right: The proposed method could automatically recover the curve from outliers.
The choice of constraint

The automatic choice of tuning parameters is rather challenging in the context of unsupervised learning. For example, methods like cross-validation are typically not applicable. This is surely true in manifold learning and multi-manifold clustering. Our approach shares this difficulty with others. The choice of neighborhood radius and the number of clusters, for example, is common with other methods. What sets our approach apart is the use of a constraint on the paths. We discuss its choice in some detail. Consider choosing of the angle constraint (working with an annulus graph) in the context of multi-surface clustering. On the one hand, a large angle causes more points from different manifolds to be connected via constrained paths and thus makes the feature vectors weaker. On the other hand, a small angle could cause points on the same surface not to be connected by a constrained path thus leading to disconnecting points on the same surface. In Figure 3.13 we look at the example of two intersecting spheres. We chose a landmark point from one of the sphere and for any other point, we computed the maximum angle along an unconstrained shortest-path joining the point to the landmark. We then plot the distributions of that angle among the two groups. We observe they are fairly well separated. This offers some hope that we can use this maximum angle to tune the angle constraint. The idea would be to optimize a form of clustering measurement (in dimension one where the maximum angle lives). This strategy would work in this simple example. Figure 3.14 shows the sensitivity of six synthetic datasets to the angle constraint.
Figure 3.13: Left: Two intersecting Spheres. Right: The distribution of maximum angle along an unconstrained shortest-path between all the points and a given landmark.

Figure 3.14: The sensitivity of six synthetic datasets to the angle constraint.
Chapter 4

Summary and conclusion

4.1 Summary

Having presented the background of shortest constrained path and the practical framework of manifold leaning and clustering, it is valuable to summarize the presented material and mention some idea for possible future work. The main problem addressed in this thesis is manifold learning and clustering by a novel constrained shortest-path. Former methods in the area of manifold learning to find a low-dimensional representation of data are not applicable for every datasets like those with a hole. In our proposed approach, we used a new distance to determine the shortest-path and then used this path for manifold learning and clustering.

In chapter 2 we first provided a review of related work in the area of dimensionality reduction and manifold learning. Specifically, we categorized the existing techniques into two main group. The first group contains linear dimensionality reduction techniques such as PCA and LDA. The second group contains non-linear techniques specially recently proposed manifold learning techniques such as LLE, LE, and Isomap. We provided a brief introduction for these approaches.

The proposed manifold learning has been introduced in chapter 2 followed by introducing our constrained shortest-path. In this chapter, we first state the manifold learning problem and then we provide details of the popular algorithms such as Isomap, LLE. Later, we provide detailed information about constrained shortest-path distances in two different graphs, namely $c$-ball and q-nearest-neighbor graphs.
Exact and approximate algorithms for constrained shortest-path are provided in order to reduce the computational complexity of exact shortest-path algorithm. Having introduced our shortest-path algorithm, we introduced a novel manifold learning algorithm based on defined distances. We provided all its required technical support and proofs. Finally, we provided its implementation and numerical experiments.

Chapter 3 deals with the problem of manifold (surface) clustering by first stating the problem. The we provided a short review of state-of-the-art techniques in this area such as K-Manifolds, Spectral Multi-Manifold Clustering (SMMC), Spectral Curvature Clustering (SCC) and Spectral Clustering. In the following, we proposed our constrained shortest-path for multi-manifold clustering. The computational complexity of the proposed algorithm and its technical support are provided to discuss the details of the algorithm. Finally, we presented its implementation and numerical experiments.

### 4.2 Conclusion and future work

Our presented methodology for manifold learning and manifold clustering using constrained shortest-path has shown promising results when applied to both synthetic and real datasets. However, there are issues that could be considered as possible future work. For example, we assumed that the data is coming from a smooth manifold which could not happen in reality. Therefore, we suggest to extend this idea so that we could handle scenarios when the data has some discontinuity. On the other side, we see a lot of applications that the proposed algorithms could be used. For instance, we can use them in the area of active learning by selecting those data points for training that are consistent with the constraints. Additionally, we could use them in the area of background subtraction, where we could assume that the foreground is low-dimensional manifold embedded in background. There are many applications in computer vision in which the proposed algorithms could be used.
Acknowledgments

This work was partially supported by a grant from the National Science Foundation (DMS-09-15160). Part I of thesis, in part, consists primarily of previously published material. Part of the content first were published in two Journals. Babaeian, Amir, Alireza Bayestehtashk, and Mojtaba Bandarabadi. "Multiple manifold clustering using curvature constrained path." PloS one 10, no. 9 (2015): e0137986. Babaeian, Amir, Mohammadreza Babaee, Alireza Bayestehtashk, and Mojtaba Bandarabadi. "Nonlinear subspace clustering using curvature constrained distances." Pattern Recognition Letters 68 (2015): 118-125. I want to thank my friends and the co-authors of my papers, Alireza Bayestehtashk, Mojtaba Bandarabadi and Mohammadreza babae. The author of this dissertation was the the primary author of both papers.
Part II

Community Detection In Large Graphs; Analysis, Design And Implementation
Chapter 5

Introduction

Network analysis and community detection techniques have been studied for a while [Fortunato (2010)]. The main objective of graph clustering is to group data points based on similar properties they share. Due to the massive scale of modern data, using a distributed parallel implementation is a necessary task. Using such a parallel system, we can run the software in multiple processors in different machines where the the processor can communicate using the distributed implementation [Dean and Ghemawat (2008); Lämmler (2008)]. Using such a system we are able to mine graphs with millions of nodes and billions of edges. In this chapter we discuss a distributed graph clustering algorithm called message passing and its implementation in a multi cluster distributed system.

5.1 Motivation

Many complicated systems are represented using network structures. In recent years, due to the size of available data, the power of computers, as well as distributed computing frameworks like Hadoop [Dean and Ghemawat (2008)] and Spark [Zaharia et al. (2010)], developing the computational techniques that help to better understand network structure and are able to run at scale has been an important challenge. In general, we are looking for algorithms that scale linearly with the size of networks. Since Spark’s multi-stage in-memory primitives provide excellent performance for iterative algorithms [Xin et al. (2013)], it is a very
suitable platform to implement linearly scalable community detection algorithms. The Spark platform allows the user to load the data into the clusters memory and query machines repeatedly. The Spark platform allows us to design a distributed graph clustering algorithm, and its implementation will help to analyze networks with over billions of edges and millions of nodes.

5.2 Problem statement

In this thesis, we propose, design, implement and analyze a message passing algorithm for finding communities in large networks. The proposed algorithm uses the network structure and does not need any information about nodes in the graph. An initial label will be assigned to every node, then nodes start sending messages (label information to their neighbors). After a few iterations, each community member converges to the same label, thus forming a community.

5.3 Contribution

The contributions of this thesis are comprised as:

• Proposing an algorithm that can be run on distributed machines and is able to find communities in very large graphs.

• Implementing our algorithm in the Spark framework where it’s multi-stage in-memory primitives provide a faster performance.

• Performing various tests on random and real networks to check the performance of the proposed method in terms of accuracy and time complexity.

• Checking the community detection accuracy of the algorithm using a few modularity measures.
5.4 Overview of community detection algorithms

During the last few years, several community detection algorithms have been developed. These include traditional algorithms like graph partitioning and hierarchical clustering [Friedman et al. (2001)], divisive algorithms [Girvan and Newman (2002)], modularity-based methods [Newman (2004)] and methods based on statistical inference [Winkler (1972)].

Although a brief overview of these algorithms is out of scope of this thesis, we refer the interested reader to [Fortunato (2010)]. The main focus of this thesis is not on community detection but rather on scalable and parallel implementation of a message passing algorithm using an open source cluster-computing network called Spark. A universal definition of community is not agreed upon. For example, [Fortunato (2010)] introduces three main definitions of community called local, global, and vertex similarity based communities. They divide the community detection algorithms into four main groups: divisive algorithms, modularity-based methods, spectral algorithms, methods based on statistical inference. Most of the community detection algorithms presume the network is formed by many disjoint communities. Intuitively, a community in a network is a collection of nodes which resemble each other but are different to other nodes in the network. There would be more edges between nodes belonging to the same community versus nodes in different communities [Newman (2003); Girvan and Newman (2002); Wasserman and Faust (1994); Fortunato (2010)]. For example, communities in social networks represent people that possibly share the same interests or same beliefs. Communities in Protein-Protein Interaction (PPI) networks are subject of intense investigations in biology and bioinformatics, as the interactions between proteins are fundamental for each process in the cell and can define the role of the proteins [Guberman et al. (2011)]. Depending on our definition of community, we can design different algorithms for community detection.

The majority of community detection algorithms assume that the communities in the network do not overlap with each other. In other words, there are not common nodes among different communities [Raghavan et al. (2007)]. In general, this may not be a correct assumption depending on the network under consideration.
For example, some communities may overlap, such as communities in which people belong to more than one social group. Figure 5.1 shows the Zachary Karate Club network, which is a network of friendships among 34 members of a karate club (Zachary, 1977). As illustrated in the figure, the graph is sparse, having an average node degree of around 5. In general, the majority of real networks we work with are sparse.

Another challenge of community detection techniques stems from the massive scale of modern data, which render many available methods impractical. Thus, we need to find algorithms that can run at scale and can be distributed across machines. The message passing algorithm proposed by Raghavan et al. (2007) is one of the fastest algorithms for non-overlapping community detection with a near linear time complexity. There are other versions of this algorithm proposed by Gregory (2010); Leung et al. (2009) for overlapping communities.

In this project, we focus on distributed randomized message passing algorithms for non-overlapping communities. Our algorithm can run on graphs with over a hundred million nodes and billions of edges.

Figure 5.1: The Zachary Karate Club network, an example of a sparse network.
5.5 Structure of part II

The second part of this thesis is divided into 7 main blocks. Each of these parts contains thematically related sections and all parts build upon each other. The outline is as follows:

5. **Introduction**: We give a general information about the motivation, problem statement, and contribution of this thesis.

6. **Related works**: We briefly review message passing algorithms and different variants of that for community detection of disjoint graphs and graphs with overlapping communities. We also propose a distributed version of [Gregory (2010)].

7. **Distributed randomized message passing algorithm**: We present the randomized distributed message passing algorithm.

8. **The stochastic block model**: We introduce a random network model called simple stochastic block model used in our experimental results.

9. **Implementation of the algorithm in Spark**: We discuss our design for the message passing algorithm, explain how to use the MapReduce and Spark frameworks for our implementation, and talk about the measurement of the clustering results.

10. **Experimental results**: We perform multiple numerical experiments on random graphs and empirically analyze the performance of our algorithm under different parameter choices. We discuss a randomized-parallel message passing algorithm with dynamic update probability as well. The experimental results on the stochastic block model show how the number of iterations change with different parameter choices and test the sensitivity of our algorithm to these parameters. For the reminder of the section, we test our algorithm on real networks with millions of nodes and billions of edges. We discuss about Spark settings, cluster optimization, and the running time on real networks. Also, we compare the result of our algorithm with a few real networks with ground truth. Lastly, we measure the clustering accuracy using a few modularity measures.

11. **Summary and Conclusion**: We summarize the key results of this thesis and give closing remarks. Additionally, we provide possible directions for future work.
Chapter 6

Related work

The main idea behind the message passing algorithm [Raghavan et al, 2007] is that every node in the graph receives a message from it’s neighbors, which is the label identifier. We describe the message passing algorithm below:

\begin{algorithm}
\caption{Message passing algorithm}
\begin{algorithmic}
\Input graph $\mathcal{G}_N = (\mathcal{V}, \mathcal{E})$ where $N$ is the number of nodes in the graph.
\Assign a set of indices $1, \ldots, N$ to the nodes in the graph.
\Initialize all the nodes $1, \ldots, N$ in the graph with a set of distinct labels $L = l_1, \ldots, l_N$.
\While $l_1, \ldots, l_N$ does change 
\For $i = 1$ to $N$ do
\Update label $l_i$ using the most frequent label of it’s neighbors.
\EndFor
\EndWhile
\Output communities $C_1, \ldots, C_K$ where the nodes in each community have the same label in $L$.
\end{algorithmic}
\end{algorithm}

The algorithm divides nodes in the graph into the groups that share similar labels. The time complexity of the proposed algorithm for each iteration is $O(|\mathcal{E}|)$ where $|\mathcal{E}|$ is the number of edges in the graph. [Raghavan et al, 2007] reported that the algorithm converges in 5 iterations in most cases and detects the correct community structure. [Leung et al, 2009] empirically analyzed the algorithm on a
Zhang and Moore (2014) employs recent methods based on statistical physics in order to detect statistically important communities. Their algorithm uses message passing but does not scale very well with the number of clusters. The largest dataset they test their algorithm on was obtained from Google Web data and consists of about 900,000 nodes and 5 million edges. They don’t provide detailed information about their implementation or how their algorithm scales to billions of edges. From their implementation, it appears they take advantage of a parallel machine learning system called graph lab Low et al. (2012, 2014). Spectral clustering Ng et al. (2001) is another method for community detection and graph clustering but the running time of these algorithms is cubic on the number of nodes, which makes them unsuitable for graphs with over 1000 nodes and impossible for graphs with millions of nodes. In recent work based on spectral clustering, Tsironis et al. (2013) propose an approximate algorithm based on k-means implemented in MapReduce. They apply their method on the Amazon dataset, which is the smallest dataset in our real data and it took them 2 days to extract 100 clusters. Clearly, their methods cannot be applied to large or even mid size data. We are particularly interested in algorithms that can scale efficiently with the size of data by adding extra processors where the time wasted for communication between processors does not increase drastically as the size of the data increases. We exclude spectral clustering in our studies and solely focus on various message passing algorithms and their specific implementation in Spark.

There is a variant of Algorithm 10 that decreases the weight of messages propagating from the original label Leung et al. (2009). One drawback of Algorithm 10 is that the algorithm is not able to detect overlapping communities. Gregory (2010) proposed another variant of the algorithm which is able to detect overlapping communities. They keep more than one community identifier in each iteration of the algorithm. They do asynchronous updating and their termination condition is based on the minimum number of vertices labelled with each community iden-
tifier since the number of identifiers last reduced. The number of communities identifier is $N$ at the first iteration. We introduce a distributed implementation of [Gregory (2010)] methods in Algorithm 11 though in this project our focus is on non-overlapping communities.

The maximum number of communities each node in the graph could belong to is denoted by $\nu$. The neighbors of node $i$ in the initialization part of Algorithm 11 are denoted by $1, \ldots, n_i$. Actually, for each node $i$ in the graph we form a uniform probability distribution on the labels of it’s neighbors. Algorithm 11 runs synchronously. For cases with $\nu < 2$, Algorithm 11 is identical to synchronized version of Algorithm 10.
**Algorithm 11**  Message passing algorithm with overlapping communities

**Input:** graph \( \mathcal{G}_N = (\mathcal{V}, \mathcal{E}) \) where \( N \) is the number of nodes in the graph, threshold \( 1/\nu \). A set of distinct labels \( L = l_1, ..., l_N \).

**Assign** a set of indices \( 1, ..., N \) to the nodes in the graph.

**Initialize** each node \( i \in 1, ..., N \) in the graph with a set of pairs \( S^i = [(l_1, 1/n_i), (l_2, 1/n_i), ..., (l_{n_i}, 1/n_i)] \) (set of label-probability pairs). The number of neighbors of node \( i \) is \( n_i \). The set of labels assigned to node \( i \) is denoted by \( L_i = \bigcup_{k=1}^{n_i} l_k \).

**From** a set of \( S = [S^1, ..., S^N] \)

**while** set \( S \) does change **do**

  **Form** another set \( \tilde{S} = S \).

  **for** \( j = 1 \) to \( N \) in set \( \tilde{S} \) **do**

    **Pick** a node \( i \) in the graph randomly from the set of unvisited nodes.

    **Update** set \( S^i \) in \( \tilde{S} \) as follow:

    If there are at least two pairs in \( S^i \in S \) with probability equal or greater than \( 1/\nu \) keep \( \nu \) pairs with highest probability. If all the probabilities are less than \( 1/\nu \) or there is one pair where it’s probability is equal or greater than \( 1/\nu \) just keep the pairs with highest probability (break the tie randomly).

    **Normalize** set \( S^i \) to form a valid probability distribution.

    **Mark** node \( i \) as visited.

  **end for**

  \( S = \tilde{S} \)

  **for** \( j = 1 \) to \( N \) on set \( S \) **do**

    **Form** \( L^i_{ne} = \bigcup_{k=1}^{\text{ne}} L_k \) as the set of labels assigned to all the neighbors of node \( i \). Set of neighbors of node \( i \) is denoted by \( \text{ne} \).

    **Reinitialize** \( S^i = [(l_1, p_1), (l_2, p_2), ..., (l_{\text{ne}}, p_{\text{ne}})] \) where \( p_k \) for \( k \in 1, ..., |\text{ne}| \) is the sum over the components of neighbors of node \( i \) that have label \( l_k \).

    **Normalize** set \( S^i \) to form a valid probability distribution.

  **end for**

**end while**

**Output:** \( S \).
Chapter 7

Distributed randomized message passing algorithm

As can be seen from Algorithm 10, the size of our available data make it infeasible to sort graph nodes based on their ID in every iteration. Furthermore, when there are tens of millions of nodes, we need to divide them into different sets and store them into different machines or partitions. Before we discuss the actual implementation of the algorithm including memory management and machine communication, we introduce a randomized version of the message passing algorithm that can be implemented in distributed systems. We update the current state of each node in the graph with probability $\alpha$. The randomized message passing algorithm is presented in Algorithm 12. The key difference between Algorithm 12 and Algorithm 10 is that Algorithm 12 can be implemented in distributed systems.
Algorithm 12  Randomized-nonsequential message passing algorithm

**Input:** graph $\mathcal{G}_N = (\mathcal{V}, \mathcal{E})$ where $N$ is the number of nodes in the graph, parameter $\alpha$.

**Assign** a set of indices $1, ..., N$ to the nodes in the graph.

**Initialize** all the nodes $1, ..., N$ in the graph with a set of distinct labels $L = l_1, ..., l_N$.

**Form** another set $\tilde{L} = L$.

**while** set $L$ does change **do**

  **for** $j = 1$ to $N$ in set $\tilde{L}$ **do**

    **Pick** a node $i$ in the graph randomly from the set of unvisited nodes in current iteration.

    **Update** the label of index $i$ in set $\tilde{L}$ with probability $\alpha$ using the most frequent label of its neighbors in $L$.

    **Mark** node $i$ as visited.

  **end for**

  $L = \tilde{L}$

**end while**

**Output:** communities $C_1, ..., C_K$ where the nodes in each community have the same label in $L$.

Since we want to update all the nodes at once at the end of each iteration, we use another array $\tilde{L}$ to keep track of updated nodes. This indicates that in each iteration of the algorithm the nodes won’t be updated sequentially so the ordering is not important. In other words, there is no need to keep the order of nodes to make it feasible to implement Algorithm 12 in a distributed system. At the end of each iteration we update the current state of all the nodes $L$ with $\tilde{L}$. For the extreme case of $\alpha = 1$ every node will get updated with the most frequent label of its neighbors. The higher $\alpha$ gives us a quicker convergence to a steady state. The choice of $\alpha = 1$ may present challenges. To illustrate the importance of $\alpha$ in Algorithm 12, we show the change of labels in one iteration of Algorithms 10 and 12 in Figure 7.4. This is a simple graph with four nodes and two communities. The left figure shows the change of labels in one iteration of Algorithm 10 in
the order of $A, B, C, D$. As it illustrated in Figure 7.1 the algorithm converges in one iteration and detects correct communities. The right portion figure shows the change of labels in one iteration of Algorithm 12 with $\alpha = 1$ in the order of $A, B, C, D$. Algorithm 12 with $\alpha = 1$ is identical to synchronized updating of nodes in each iteration. Nodes switch their labels in each iteration and never converge to the correct communities. Thus the choice of correct $\alpha$ is important to make a good trade off between the accuracy of community detection results and the speed of convergence. In Chapter 10 we discuss this in more detail, and empirically compare the performance of our algorithm on detecting communities for different $\alpha$ values. Similarly we can define an update probability $\alpha$ for Algorithm 11 and design a distributed implementation of that for community detections with overlapping communities. However in this project our focus is on Algorithm 12 which is for non-overlapping communities.

Figure 7.1: Left: Change of labels in one iteration of Algorithm 10. Right: Change of labels in one iteration of Algorithm 12 with $\alpha = 1$. 
The stochastic block model

We evaluate the performance of message passing algorithms on the simplest model of a network with observable communities, the so-called stochastic block model (SBM), originally introduced in social science [Holland et al. (1983)]. The stochastic block model is a generative model for random graphs where the edges within and between communities are connected with different probabilities. Formally, a network is here modeled as an undirected graph $G = (\mathcal{V}, \mathcal{E})$, with vertex set $\mathcal{V} = [N] := \{1, \ldots, N\}$ and edge set $\mathcal{E}$ lists the (unordered) pairs of nodes that are neighbors in the graph. Such a graph is equivalently represented by its adjacency matrix, denoted $A = (A_{ij} : i, j \in [N])$, which is symmetric with entries in $\{0, 1\}$ and 0’s on the diagonal; $A_{ij} = 1$ indicates that nodes $i$ and $j$ are (direct) neighbors, meaning $(i, j) \in \mathcal{E}$. The SBM is a model of random graphs, best defined via adjacency matrices. Let $K$ be an integer with $1 \leq K \leq N$, and let $\mathbf{P} = (p_{ab} : a, b \in [K])$ a symmetric matrix with coefficients in $[0, 1]$. Also, let $n_1, \ldots, n_K$ be positive integers such that $N = n_1 + \cdots + n_K$. The vertex set $[N]$ is randomly partitioned into $K$ subsets of sizes $n_1, \ldots, n_K$, representing the communities in the network. For a node $i \in [N]$, let $z_i = a$ if $i$ belongs to community $a \in [K]$; also, let $\mathbf{z} = (z_1, \ldots, z_N)$. We will also use the notation $i \in \{a\}$ to indicate that $z_i = a$, meaning that node $i$ belongs to group $a$. In the model, two nodes $i \in \{a\}$ and $j \in \{b\}$ are connected with probability $p_{ab}$ independently. Note that when the $p_{ab}$’s are all equal, the SBM in the Erdős-Rényi model. For simplicity, we will highlight the special case where there are $K$ communities of
equal size $n_1 = \cdots = n_K = n$, and for some $p_1 \neq p_0$, $p_{aa} = p_1$ and $p_{ab} = p_0$ for all $a \neq b$ in $[K]$. We call this the simple SBM with parameters $(n, K, p_1, p_0)$. This model corresponds to the four-parameter SBM of Rohe et al. (2011), though our parametrization is different. For concreteness, we assume that $p_1 > p_0$, though the results do not require that.

Figure 8.1 shows the simple stochastic block model with parameters $n = 100, K = 3, p_1 = .09, p_0 = .005$ used in our experiments. The real graphs we work with are consistent with stochastic block model in terms of sparsity and the number of edges within nodes in each community and in-between communities.

Figure 8.1: Simple stochastic block model with parameters $n = 100, K = 3, p_1 = .09, p_0 = .005$
Chapter 9

Implementation of the algorithm in Spark

In this section we discuss the design and implementation of Algorithm 12 in the Spark framework, and give an explanation about the main functions used in our implementation.

9.1 MapReduce and Spark

MapReduce has been used in a lot of big data applications where the data is distributed in several machines on cluster. In the MapReduce framework users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model. In the Map step, we break down the problem into the smaller problems. Then, the smaller problems are distributed across different machines. We call this machines the worker nodes. In the Reduce step, all the results of the worker nodes get combined with each other, forming the final result. The Map and Reduce steps, as well as the intermediate steps, are shown in Figure 9.2.
Using functional programming language helps to implement the parallel algorithm easily and run them on a series of machines [Dorband et al. (2003)]. The run-time system helps to partition the data efficiently and schedule the execution over the cluster. It also manages the machine flop and takes care of communication across all the machines. This provides inexperienced programmers with an easier task of using parallel algorithms in order to solve large scale problems. In applications where we reuse the data throughout the different iterations of the algorithm (for example iterative algorithms in machine learning) MapReduce would be inefficient since it needs to reload the data from stable storage in every iteration of the algorithm. As it can be seen from Algorithm 12 we need to reuse the working set of our data across multiple parallel operations. The algorithm spends more than 90% of its time to read from and write into the HDFS file system. Hadoop is an open source implementation of MapReduce. In order to gain faster performance, we use the Spark framework to overcome the challenge of rewriting the result into the disk in every step of the iterative algorithm. The big difference
here is that rather than writing the output of the iterative algorithm on the disk in every iteration and reading the result from the disk, we keep the result of the previous iteration in the distributed memory system (RAM). Having said that, if the size of the data is very large where we can not keep the intermediate result on RAM we need to write the result on the disk. Figure 9.2 shows the main difference between MapReduce and Spark for iterative algorithms. We still take advantage of the automatic parallelization, scalability and fault tolerance offered by Hadoop. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDDs) [Zaharia et al. (2012)]. An RDD is the Spark abstraction of data distributed over different machines and is suitable for interactive, iterative algorithms. RDD provides a bunch of APIs other than just Map and Reduce. These APIs like groupBy, aggregate, flatMap and filter make it very easy for us to do operations on the large graph [Zaharia et al. (2012)]. We can create RDDs using different datasets, e.g. text data. For example, groupBy is applied to an RDD with the list of items and creates a new RDD where the list of items are grouped using some key in the new RDD. Depending on the specific algorithm and the amount of shuffling between different partitions, some of these operations could be more or less costly.

In the following subsection, we discuss the implementation of our algorithm in detail, where we break down the implementation into three functions. We call them Send, Reduce and Update. We explain how we take advantage of Spark for efficient and fast implementation of these three functions.

[Malewicz et al. (2010)] introduces a graph algorithm on MapReduce. In their method the nodes and edges are preserved in the same machine and messages fly over different machines.

### 9.2 Design and implementation of the message passing algorithm

In order to do the operations in the distributed system efficiently we use a monoid. A monoid is an algebraic structure studied in abstract algebra, which is
used in programming languages McBride and Paterson (2008). We first define a monoid.

**Definition:** Given a type $T$, a binary operation $Op : (T, T) \rightarrow T$, and an instance $Zero : T$, with properties specified below, the triple $(T, Op, Zero)$ is called a monoid.

Two properties a monoid must satisfy are:

**Neutral element,**

$$Zero \; Op \; a == a \; Op \; Zero == a \quad (9.1)$$

**Associativity,**

$$(a \; Op \; b) \; Op \; c == a \; Op \; (b \; Op \; c) \quad (9.2)$$

Associativity allows us to do the calculation in any order and that is very important in distributed systems where we need to do the same operation in various machines. This will help us to do the operations in several partitions/machines in parallel and aggregate the result of machines to get the final answer. In the actual implementation of Algorithm 12 in every iteration we use three functions called: $Send : (V, E, S, W) \rightarrow M$
\[ \text{Reduce} : (M, M) \to M \]
\[ \text{Update} : (S^\text{old}_{id}, M) \to S^\text{new}_{id} \]

\( V \) is the set of vertex IDs and \( E \) is the set of pairs of source and destination IDs of vertices for every edge in the graph, \( S \) is the current states (labels) of the vertices and \( W \) is the set of edge weights. \( M \) are the messages sent to the destination nodes. In fact, the input of the algorithm is a list of all the nodes and all the edges. Send function knows the IDs of source and destination nodes. This is the abstraction we use for a directed graph, while for an undirected graph we duplicate the number of edges by switching incoming and outgoing edges. Edge weights are either 0 or 1 in our graphs, but the implementation is more general and can be used for weighted graphs. The Send function gets called \(|E|\) times which is equal to the number of edges in the graph. You can think of the Send function as the Map function in the MapReduce paradigm. For every node in the graph, the Reduce function reduces the incoming messages from all the neighboring nodes to one message by taking advantage of monoid properties. This function takes two messages with the same ID of destination node and produce a single message. This reduction is done first per partition, then all the messages related to all partitions are combined at their destination node forming the final message, which is then used to update state of the destination node. Assume the Reduce function acts like + operator, using monoid property \( M_1 + (M_2 + M_3) = (M_1 + M_2) + M_3 \), then if the nodes that sent \( M_1 \) and \( M_2 \) messages are in the same partition we do \( M_1 + M_2 \) first, versus if the nodes that sent \( M_2 \) and \( M_3 \) are in the same partition we do \( M_2 + M_3 \) first. Figure 9.3 shows the Send, Reduce and Update steps of the actual implementation on a simple graph. The Update function updates the current state of each node in the graph with probability \( \alpha \). This function is called \(|V|\) times, the number of nodes in the graph.
Send

\[ V = [1 \ 2 \ 3 \ 4 \ 5 \ 6] \]
\[ E = [ [1 \ 2] \ [1 \ 3] \ [1 \ 4] \ [3 \ 2] \ [3 \ 4] \ [3 \ 5] \ [4 \ 2] \ [5 \ 2] \ [6 \ 2] ] \]

Reduce

Partition 1:
\[ \text{Reduce}: ([A, 1], [A, 1]) = [A, 2] \]

Partition 2:
\[ \left\{ \begin{align*}
\text{Reduce}: ([B, 1], [A, 1]) &= [[B, 1] \ [A, 1]] \\
\text{Reduce}: ([[B, 1] \ [A, 1]], [A, 1]) &= [[A, 2] \ [B, 1]]
\end{align*} \right. \]

Final Reduce:
\[ \text{Reduce}: ([[A, 2] \ [B, 1]], [A, 2]) = [[A, 4] \ [B, 1]] \]

Update

\[ \text{Update}: (C, [A, 4] [B, 1]) = \begin{cases} 
A & \text{With probability } \alpha \\
C & \text{With probability } 1 - \alpha 
\end{cases} \]

Figure 9.3: Send, Reduce and Update steps on a simple graph. In this example, Reduce function computes the messages sent to the node with ID = 2.
9.3 Measurement of the clustering result

Since the data and the number of communities are very large, due to computational limitations, there will not be an exact measurement to compute the clustering accuracy. For example, if the number of communities exceeds a few hundred, the in-distributed system sorting would be costly and impractical. Furthermore, exact measurement requires computing all possible permutations between group IDs in the ground truth and clustering result – which is infeasible. Thus, we need to find an approximate algorithm in order to measure the accuracy of the clustering algorithm. We have designed a very simple algorithm for this matter. First, in the ground truth table we group the nodes by community ID, where we change the community ID to the minimal node ID of that cluster. We do the same for clustering result table. Then, we check every node of each community in the ground truth table, and we find the corresponding community ID of these nodes in the clustering result table. For example, lets say \[ 1 \ 2 \ 3 \ | \ 4 \ | \ 5 \ 6 \] is our clustering result and \[ 1 \ 2 \ | \ 3 \ 4 \ | \ 5 \ 6 \] is our ground truth where \(|\) separates communities in each array. Using the minimum node ID, we get \[ 1 \ 1 \ 1 \] and \[ 1 \ 1 \ 3 \] for the first community, \[ 4 \] and \[ 3 \] for the second community and \[ 5 \ 5 \] and \[ 5 \ 5 \] for the third community for the clustering result and the ground truth respectively. Then we compute the $F_1$ score between each of these pairs (per community). The final $F_1$, which is a clustering accuracy, would be the average of all three $F_1$ scores. However, this measurement causes the following problem: if the minimal node ID is wrong, the community ID will be wrong for all the nodes in that community. So, in order to make our algorithm more robust, we apply a hash function $f$ to hash every node ID and community ID in both the ground truth and clustering result tables in each experiment:

$$f(id) = (P \times id) \mod C,$$

where $id$ is the ID of a given node, $P$ is a chosen prime number and $C$ is a constant and mod is the modulus. In our experiments we apply the hash function to the nodes IDs using 10 different prime numbers. In order to avoid the hash
collision, we also set the constant to be a large number. We get 10 different results using this approach. We choose the best $F_1$ score per community and then take the average as our final $F_1$ score as the measurement of our clustering result. This simple approach makes our result more robust.
Chapter 10

Experimental results

10.1 Analysis of parameters of message passing algorithm

In order to evaluate the performance of the community detection algorithm, we use three different approaches. The first approach is to run the algorithm on available real world networks and compare the clustering result with ground truth. In reality, we do not know the actual communities, or the ground truth itself may not be very well defined, since they are solely based on external criteria. Furthermore, some of these ground truths are for overlapping communities which make it hard to compare with the result of our clustering algorithms, especially for large graphs where a node could belong to multiple communities. There are cases in our real networks where nodes belong to over 600 communities. The second approach uses a modularity measure to compare the performance of the community detection algorithm on real or synthetic networks under different choices of parameters. The third approach is to generate a random network with known communities and evaluate the performance of the community detection algorithm on those. In order to do that, we use a simple stochastic block model. The misclustering error of such a method $\hat{z}$ for small number of communities $K$ is defined as

$$R_A(\hat{z}) = \min_{\sigma \in S_K} \frac{1}{N} \sum_{i=1}^{N} \{ \sigma(\hat{z}_i) \neq z_i \}, \quad (10.1)$$
where $\mathcal{S}_K$ denotes the set of permutations of $[K]$. This is the labeling error up to permutation of labels. Because computing the error (10.1) when $K$ is large is computationally intractable, for $K > 5$ we used the following alternate definition:

$$
\tilde{R}_\Lambda(\hat{z}) = \frac{1}{(N-1)^2} \sum_{i \neq j} \{ z_i = z_j, \hat{z}_i \neq \hat{z}_j \} + \{ z_i \neq z_j, \hat{z}_i = \hat{z}_j \}. 
$$

(10.2)

In Table 10.1 we fix $K = 3, n = 100, p_1 = .18$ and change $p_0$ from .01 to .05 and run the algorithm with different $\alpha$ and report the number of times out of 50 experiments the algorithm gives us the correct community structure. When $p_0$ increases, in-between community edges increase, so the choice of $\alpha$ would be an important task here. It can be observed that there is no linear relation between $\alpha$ and the hardness of the community detection algorithm. Real networks in general are very sparse with few outgoing edges between communities and the ratio of in-between community edges to within community edges is very small.

Table 10.1: Number of times out of 50 experiments the algorithm returns the correct community structure

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>.1</th>
<th>.2</th>
<th>.3</th>
<th>.4</th>
<th>.5</th>
<th>.6</th>
<th>.7</th>
<th>.8</th>
<th>.9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_0 = .01$</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>$p_0 = .02$</td>
<td>45</td>
<td>45</td>
<td>45</td>
<td>40</td>
<td>46</td>
<td>45</td>
<td>49</td>
<td>44</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>$p_0 = .03$</td>
<td>36</td>
<td>38</td>
<td>43</td>
<td>40</td>
<td>37</td>
<td>37</td>
<td>40</td>
<td>45</td>
<td>40</td>
<td>35</td>
</tr>
<tr>
<td>$p_0 = .04$</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>$p_0 = .05$</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Choosing a fixed parameter $\alpha$ in every iteration of the algorithm may not be the best choice. Instead, at the beginning we would like to start with smaller $\alpha$ and after a few iterations where some nodes are assigned to the correct labels, we want the nodes with correct label in each community to send a message to their neighbors with higher probability. In order to do that, we modify Algorithm 12 to Algorithm 13 by introducing a dynamic randomization weight $\eta$:...
Algorithm 13  Randomized-nonsequential message passing algorithm with dynamic probability of update

**Input:** graph \( G_N = (V, E) \) where \( N \) is the number of nodes in the graph, parameter \( \alpha \), and \( \eta \).

**Assign** a set of indices \( 1, \ldots, N \) to the nodes in the graph.

**Initialize** all the nodes \( 1, \ldots, N \) in the graph with a set of distinct labels \( L = l_1, \ldots, l_N \).

**Form** another set \( \tilde{L} = L \).

\( j = 0 \) where \( j \) is the current iteration of the algorithm.

**while** set \( L \) does change **do**

**for** \( i = j \) to \( N \) in set \( \tilde{L} \) **do**

Pick a node \( i \) in the graph randomly from the set of unvisited nodes in the current iteration.

**Update** the label of index \( i \) in set \( \tilde{L} \) with probability \( \eta^j \alpha \) using the most frequent label of it’s neighbors in \( L \).

Mark node \( i \) as visited.

**end for**

\( L = \tilde{L} \)

\( j = j + 1 \)

**end while**

**Output:** communities \( C_1, \ldots, C_K \) where the nodes in each community have the same label in \( L \).

On one hand, we don’t want to start with high probability of update because of the problems mentioned in Figure 7.1. On the other hand, with small \( \alpha \) the algorithm takes more iterations to converge. Using Algorithm 13, as the number of iterations increases, the probability of update increases. For example, for iteration 20 the probability of update for the nodes in the graph is \( 1.05^{20} \times .1 = .26 \) resulting in a faster update by it’s neighbors. In large networks with billions of nodes, the number of iterations until the algorithm converge to the final result can be large, so the speed is very important to us. In general, our experimental results show \( \eta \) should be less than 1.05 because larger \( \eta \) results in a higher probability of update.
in starting iterations. Thus the nodes move toward wrong labels assigned by their neighbors, then propagate those wrong labels with higher speed as the number of iterations increases, resulting in poor communities.

In Table 10.2 we report the number of iterations the message passing algorithm takes to detect correct communities on the stochastic block model with parameters $K = 3, p_1 = .18, p_0 = .01$ and $n$ the number of nodes in each community varies from $2^6$ to $2^{11}$. We do this experiment with a fixed probability of update, Algorithm 12 with $\alpha$ at .1 and 1 and Algorithm 13 with two different $\eta = 1.03$ and $\eta = 1.05$. We report the number of iterations the algorithm takes in order to converge to the correct communities, $R_A(\hat{z}) = 0$, for each probability of update. In this experiment, we stop when there is no further change of the labels. The experimental results show a dynamic probability of update increases convergence speed while we still obtain the perfect community detection on our stochastic block model. Every experiment is repeated 10 times and the rounded average of the number of iterations is reported.

Table 10.2: The number of iterations message passing Algorithms 12 and 13 take to detect perfect communities. The number of nodes in each community is $n$.

<table>
<thead>
<tr>
<th>n</th>
<th>$\alpha = .1$</th>
<th>$\alpha = .1, \eta = 1.03$</th>
<th>$\alpha = .1, \eta = 1.05$</th>
<th>$\alpha = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>50</td>
<td>34</td>
<td>29</td>
<td>8</td>
</tr>
<tr>
<td>128</td>
<td>57</td>
<td>36</td>
<td>30</td>
<td>8</td>
</tr>
<tr>
<td>126</td>
<td>60</td>
<td>32</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>512</td>
<td>60</td>
<td>40</td>
<td>32</td>
<td>5</td>
</tr>
<tr>
<td>1024</td>
<td>69</td>
<td>42</td>
<td>34</td>
<td>5</td>
</tr>
<tr>
<td>2048</td>
<td>70</td>
<td>44</td>
<td>36</td>
<td>6</td>
</tr>
</tbody>
</table>

In Table 10.3 we do similar experiments, but this time we fix the number of nodes in each community at $n = 300$. The number of communities varies from $2^1$ to $2^4$. We also tested for $K = 2^5$ and a higher number of communities. When $K$ gets very large, the message passing algorithm fails. Our result shows for $K = 2^5$ all the nodes in the graph converge to a unique label. Intuitively, as $K$ increases the number of in-between community edges increase while the number of within community edges is fixed. The message passing algorithm breaks down in these scenarios since the number of neighbors outside the community of the node exceeds
the number of neighbors within its community. As has been mentioned in other references and as our result confirms, Algorithms 12 and 13 are suitable for sparse graphs which is the case for many real world networks.

Table 10.3: The number of iterations Algorithms 12 and 13 take to detect perfect communities. The number of communities is $K$.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$\alpha = 0.1$</th>
<th>$\alpha = 0.1, \eta = 1.03$</th>
<th>$\alpha = 0.1, \eta = 1.05$</th>
<th>$\alpha = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>34</td>
<td>30</td>
<td>25</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>33</td>
<td>27</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>41</td>
<td>39</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>44</td>
<td>42</td>
<td>34</td>
<td>5</td>
</tr>
</tbody>
</table>

10.2 Real networks

10.2.1 Experimental setup

In Table 10.4 we report the experimental set ups for all 6 real datasets. The last column is the final number of communities resulting from Algorithm 12. These numbers are less than the number of communities reported in the ground truth. This is because in ground truth every node belongs to multiple communities while in the result of the message passing algorithm, every node belongs to just one community. For example in the Orkut dataset, every node could belong to up to 500 communities.

For every dataset, we have tried three different experiments using different $\alpha$. As we know, when $\alpha$ is larger, the algorithm converges faster. As a result, for $\alpha = 0.2$, we set the number of iterations to 40, for $\alpha = 0.5$, to 20, and for $\alpha = 1$, to 10. For Amazon, DBLP, Youtube and the Live Journal datasets – graphs with less 10 million nodes and less than 50 million edges – we store the intermediate result of each iteration in memory only because these datasets are relatively small. If the graph is in memory, we can take advantage of Spark’s in-memory calculation, which is the main benefit of Spark over MapReduce.

On the contrary, for the Orkut and Friendster datasets, we store the graph on disk because they are too large to fit into memory, so reading and writing in each
Table 10.4: Experimental set up and result in Spark for the real datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No nodes</th>
<th>No edges</th>
<th>α</th>
<th>No of iterations</th>
<th>Storage level</th>
<th>No of partitions</th>
<th>Time per iteration</th>
<th>No of final communities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>334,863</td>
<td>925,872</td>
<td>.2</td>
<td>40</td>
<td>Memory Only</td>
<td>1</td>
<td>15s</td>
<td>27,210</td>
</tr>
<tr>
<td></td>
<td>.5</td>
<td></td>
<td></td>
<td>20</td>
<td>Memory Only</td>
<td>1</td>
<td>15s</td>
<td>25,757</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>10</td>
<td>Memory Only</td>
<td>1</td>
<td>15s</td>
<td>36,147</td>
</tr>
<tr>
<td>DBLP</td>
<td>317,080</td>
<td>1,049,866</td>
<td>.2</td>
<td>40</td>
<td>Memory Only</td>
<td>1</td>
<td>13s</td>
<td>28,842</td>
</tr>
<tr>
<td></td>
<td>.5</td>
<td></td>
<td></td>
<td>20</td>
<td>Memory Only</td>
<td>1</td>
<td>13s</td>
<td>26,779</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>10</td>
<td>Memory Only</td>
<td>1</td>
<td>14s</td>
<td>32,571</td>
</tr>
<tr>
<td>Youtube</td>
<td>1,134,890</td>
<td>2,987,624</td>
<td>.2</td>
<td>40</td>
<td>Memory Only</td>
<td>10</td>
<td>10s</td>
<td>65,575</td>
</tr>
<tr>
<td></td>
<td>.5</td>
<td></td>
<td></td>
<td>20</td>
<td>Memory Only</td>
<td>10</td>
<td>10s</td>
<td>59,866</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>10</td>
<td>Memory Only</td>
<td>10</td>
<td>10s</td>
<td>125,556</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>3,997,962</td>
<td>34,681,189</td>
<td>.2</td>
<td>40</td>
<td>Memory Only</td>
<td>100</td>
<td>25s</td>
<td>120,840</td>
</tr>
<tr>
<td></td>
<td>.5</td>
<td></td>
<td></td>
<td>20</td>
<td>Memory Only</td>
<td>100</td>
<td>25s</td>
<td>103,155</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>10</td>
<td>Memory Only</td>
<td>100</td>
<td>25s</td>
<td>138,913</td>
</tr>
<tr>
<td>Orkut</td>
<td>3,072,441</td>
<td>117,185,083</td>
<td>.2</td>
<td>40</td>
<td>Disk Only</td>
<td>5000</td>
<td>3.1min</td>
<td>15,488</td>
</tr>
<tr>
<td></td>
<td>.5</td>
<td></td>
<td></td>
<td>20</td>
<td>Disk Only</td>
<td>5000</td>
<td>3.1min</td>
<td>13,946</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>10</td>
<td>Disk Only</td>
<td>5000</td>
<td>3.1min</td>
<td>15,282</td>
</tr>
<tr>
<td>Friendster</td>
<td>65,608,366</td>
<td>1,806,067,135</td>
<td>.2</td>
<td>40</td>
<td>Disk Only</td>
<td>9000</td>
<td>22min</td>
<td>534,096</td>
</tr>
<tr>
<td></td>
<td>.5</td>
<td></td>
<td></td>
<td>20</td>
<td>Disk Only</td>
<td>9000</td>
<td>22min</td>
<td>808,341</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>10</td>
<td>Disk Only</td>
<td>9000</td>
<td>22min</td>
<td>798,453</td>
</tr>
</tbody>
</table>

Iteration happens in disk. We partition our data and distribute and these partitions are distributed across all the machines. We can do further optimization by sending neighboring nodes to the same partition. This might increase the speed since nodes receive messages from their neighbors avoiding between machines queries, though in our implementation we ignored this part. For small datasets, we just use one partition; however, for large datasets, we set the number of partitions to a comparably larger number. In this way, the reduce step takes less time.

10.2.2 Measurement using $F_1$ score

Some of the real networks already have a ground truth. We use the Stanford large network dataset collection (SNAP) [Yang and Leskovec (2015)] to evaluate the community detection algorithm. We run our algorithm on 6 different networks: Amazon.com, DBLP collaboration network, Youtube social network, LiveJournal social network, Orkut social network and Friendster social network. We can compare the community detection result of our algorithm with the ground truth. That gives us a criteria to evaluate the performance of the proposed algorithm on those networks. Nodes belonging to the same group share similar functions. These functions vary in different datasets. For example, this functions could be all the friends...
in a social circle, people who publish in the same venue in DBLP networks, or the category number in the amazon dataset. But as we will see in the rest of this section these ground truths may not be defined properly and might have nodes with many overlapping communities. In Table 10.5 we compute the accuracy of community detection using the $F_1$ measure computed between ground truth and community detection result for various $\alpha$. We use 10 different hash functions explained in the previous section to map the cluster IDs to different numbers. As mentioned before, we use these approximate measures because we can not simply sort the user ID in memory specifically for networks with over a million edges. Main functions have a monoid property mentioned before and can be implemented efficiently in Spark. Here we used the top 5000 communities in the ground truth and compute the average $F_1$ on them. There were some repeated rows in the ground truth which we removed with preprocessing. As seen from Table 10.5 in general the average $F_1$ score with smaller $\alpha$ is higher. However, it takes more iterations to converge. In order to make a trade-off between accuracy and speed, we can use Algorithm 13 with a dynamic update probability. In the ground truth file related to the Orkut and Youtube data, we noticed many overlapping communities (a single node can belong to over 600 communities), so reporting $F_1$ score for that data is not meaningful. Also we run the algorithm with $\alpha = 1$ on the Orkut data for about 40 iterations and the final number of communities was 698. This means that in cases with many overlapping communities, the message passing Algorithm 13 breaks down and converges to communities with bigger size.

Table 10.5: Average $F_1$ score on detected communities by applying Algorithm 12.

<table>
<thead>
<tr>
<th>Average $F_1$ Score</th>
<th>$\alpha = .2$</th>
<th>$\alpha = .5$</th>
<th>$\alpha = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>0.836687</td>
<td>0.831211</td>
<td>.748115</td>
</tr>
<tr>
<td>DBLP</td>
<td>0.425071</td>
<td>0.422366</td>
<td>0.405542</td>
</tr>
<tr>
<td>Live Journal</td>
<td>0.467967</td>
<td>0.472411</td>
<td>0.444953</td>
</tr>
</tbody>
</table>
10.2.3 Measurement using modularity measures

As we discussed, the ground truth may not be defined properly. Furthermore, in the real world defining those ground truths is subjective, so we should look for a better measure of the goodness of a community detection algorithm. We use two measures of goodness to evaluate the performance of the algorithm. We define two community goodness metrics that will explain the that good communities are both compact and well connected internally while being relatively well-separated from the rest of the network \cite{Yang and Leskovec 2015}. In order to compute those measures let’s first define some notation. Let $G = (V, E)$ be an undirected graph and $C$ is a community (set of nodes) in that network. Let $e_c = |\{(u, v) \in E, u \in C, v \in C\}|$ and $e_\bar{c} = |\{(u, v) \in E, u \in C, v \notin C\}|$ are respectively internal and external edges in $C$. One of the measures we use is the separability measure \cite{Fortunato 2010; Shi and Malik 2000}, which is given by:

$$g(C) = \frac{e_c}{e_\bar{c}}. \tag{10.3}$$

Smaller $g(C)$ means that good communities have fewer edges towards the nodes outside that community and more edges within the nodes inside the community. Another measure is the density of edges within each community which can be computed using:

$$d(C) = \frac{e_c}{n_c(n_c - 1)/2}, \tag{10.4}$$

where $n_c = |C|$ is the number of nodes in community $C$.

In another experiment, we first sort the communities in decreasing order based on their separability and density measures for the output of our community detection algorithm with parameter $\alpha = .2, .5, 1$ and the ground truth. Figure \ref{fig:separability} shows the separability measure on the Amazon, DBLP, Youtube, Live Journal and Friendster datasets. We put a cut off point at $g(C) = 4$ to illustrate the decrease in separability for different parameters more clearly. In the ground truth data, there are communities with $e_\bar{c} = 0$ and in those cases the separability measure would be infinity. Instead of infinity, we use an arbitrary number bigger than the maximum separability computed with parameters $.2, .5, 1$. The separability measure
for $\alpha = .2$ and $\alpha = .5$ decreases with a slower rate for all the datasets, meaning a better community detection. There is no significant difference between $\alpha = .2$ and $\alpha = .5$, but a choice of $\alpha = .5$ will result in a faster convergence. Figure 10.2 shows the density measure on four datasets mentioned above. A similar behavior is observed for density, though the rate of decrease for ground truth is different in the four datasets. For example, in the Youtube data we have a rapid decrease in ground truth which could be another reason that ground truth defined by the similarity function mentioned earlier might not be the best choice.
Figure 10.1: Separability measure on the Amazon, DBLP, Youtube, Live Journal and Friendster datasets.
Figure 10.2: Density measure on the Amazon, DBLP, Youtube, Live Journal and Friendster datasets.
Chapter 11

Summary and conclusion

11.1 Summary

In this project, we proposed a new distributed community detection algorithm for large networks. In Chapter 5 we discussed the motivation for community detection in large networks and stated the problem. In chapter 6 we surveyed related work for community detection and the use of message passing algorithms to detect communities. In Chapter 7 we proposed a distributed randomized message passing algorithm for community detection. The proposed algorithm can be implemented in distributed systems and run on graphs with millions of nodes and billions of edges. We analyzed our algorithm on a few random and real networks to provide insight about the choice of parameter. We discussed the implementation of the algorithm on the Spark platform and how its set up depends on the size of the dataset. We also analyzed the performance of the algorithm in terms of speed and accuracy using given ground truths and a few modularity measures like separability and density of detected communities. Experiments show that the proposed algorithm is efficient and linearly adaptable to the scale of our data by adding extra machines as the size of our networks increase.
11.2 Conclusion and future work

Experiments show that the proposed randomized message passing algorithm is efficient and linearly adaptable to the scale of our data by adding extra machines as the size of the networks increase. The randomization parameter can change the result of the community detection algorithm and the number of iterations the algorithm needs to converge. Checking the result of the community detection algorithm using separability and density measures on a few real networks and comparing with the ground truth show that a higher randomization parameter gives a better community detection result. The choice of the best randomization parameter can be a challenging task and may require further study of network topology. Also we used a specific implementation of our algorithm in the Spark framework where it’s multistage in-memory primitives provide a faster performance. In future work, one can use the randomized message passing algorithm method for overlapping communities where every node can belong to more than one community.

Acknowledgments

I want to thank Prof. Ery Arias-Castro for introducing me to the stochastic block model and the Spark community for their advice regarding the implementation of the algorithm in Spark.
Bibliography


