Efficient Graph Clustering Algorithms using Compressive Sensing

by

Daniel Mckenzie

(Under the direction of Ming-Jun Lai)

Abstract

Clustering graph-based data is a core problem in contemporary data science. In particular, as data sets grow in size and dimension, efficient algorithms are needed that can go beyond the $O(n^2)$ run time of classical algorithms such as spectral clustering. In this dissertation we propose a new paradigm that rephrases the problem of finding clusters in graphs as a compressive sensing problem. This enables us to use fast algorithms originally developed for sparse recovery (in particular, greedy algorithms such as Orthogonal Matching Pursuit or Subspace Pursuit) for the clustering problem. We propose two new algorithms, and several variations thereof, in this paradigm, which we deem "Cluster Pursuit" algorithms. In particular, `SingleClusterPursuit` takes a small set of seed vertices and returns a good cluster containing them in $O(d_{\max} n \log(n))$ time, where $d_{\max}$ is the maximum vertex degree of the graph, while `DynamicClusterPursuit` efficiently updates an existing cluster in an evolving network. We further prove that `SingleClusterPursuit` is able to recover a large fraction of a given cluster for graphs drawn from a well-known probabilistic model of graphs with communities, namely the stochastic block model.

In an additional chapter, we study the related problem of turning Euclidean data into graph data, so that graph-based clustering algorithms such as those discussed above can

be used. We analyze the use of *power weighted shortest path distances* to measure the distance between such data points, and show that this can lead to significant improvements in classification accuracy on both real and synthetic data sets.

INDEX WORDS: Spectral Graph Theory, Random Graph Theory, Compressive Sensing, Semi-supervised Learning, Unsupervised Learning, Clustering, Path Distances, Manifold Hypothesis

Efficient Graph Clustering Algorithms using Compressive Sensing

by

Daniel Mckenzie

B.Sc (hons), University of Cape Town, 2010

M.Sc, University of Cape Town, 2014

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2019

Efficient Graph Clustering Algorithms using Compressive Sensing

by

Daniel Mckenzie

Approved:

Major Professor: Ming-Jun Lai

Committee: Malcolm Adams
Juan Gutierrez
Alexander Petukhov
Qing Zhang

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
May 2019

Acknowledgments

Firstly I would like to thank my advisor, Ming-Jun Lai, for his guidance and for his uncanny ability to suggest a way around a problem whenever I got stuck. I would not have been able to finish this dissertation without his assistance (and indeed, as he was the one to suggest using compressive sensing in graph clustering, arguably would not have started it.). Thanks also to my committee for many useful discussions, and in particular Qing Zhang and Alex Petukhov for teaching several graduate courses that really influenced the way I approach mathematics.

Thanks to the Mathematics Department of the University of Georgia for providing a supportive working environment (and a paycheck) for the last six years. In addition to my advisor and committee, I'd like to thank Neil Lyall for many interesting conversations about the profession of mathematics, Lisa Townsley for teaching me to teach and Laura Ackerley for always genuinely caring. Thanks to my fellow graduate students, in particular Jason Joseph, Andrew Maurer, Ernest Guico and Eric Perkerson, who over the years have become friends and housemates, and from the start made me feel welcome in a new city. Thanks also to my friends in South Africa, and in particular the Hobo Child crew, for a WhatsApp group that never ceases to amuse.

A special thank you to my parents, Andrew and Judy, for our Sunday morning Skypes, their constant support and, on at least one occasion, dropping everything to come visit when the stress of graduate school became a bit too much. Thank you to my siblings—Carla and Joel—for repeatedly reminding me that mathematics probably isn't the most important thing in the world. Thank you to my Athens family, and in particular Ross and Teresa,

for being surrogate parents, for hosting me for numerous Thanksgivings, Christmases and Fourth-of-July's, and for teaching me a thing or two about the South.

Finally, thank you to my wonderful fiancée Amanda, for all that you've done for me. I can honestly say that getting a PhD would not have been possible without your love and support, your willingness to pick up the slack when dissertation writing consumed all of my time and your unwavering belief in me. Thanks also for agreeing to move across the country with me; I am beyond excited for our next adventure!

Table of Contents

Page

Acknowledgments iv

Chapter

1 Introduction 1

1.1 Overview of this Dissertation 1

1.2 Graphs 1

1.3 Clustering Algorithms 7

1.4 Compressive Sensing 13

2 Semi-Supervised Cluster Pursuit 19

2.1 Cluster Extraction as Compressive Sensing 19

2.2 Concentration in Random Graphs 22

2.3 Finding good supersets 25

2.4 Extracting C_1 from Ω 30

2.5 Computational Complexity 41

2.6 Numerical Results 43

3 Dynamic Cluster Pursuit 58

3.1 Overview of Dynamic Clustering 59

3.2 Probabilistic Models of Dynamic Random Graphs 62

3.3 Dynamic cluster pursuit 65

3.4 Experimental Results 71

4 Shortest path distances for clustering Euclidean data 77

4.1 Power Weighted Shortest Path Distances 79
4.2 Prior Work 82
4.3 Analysis of shortest path distances in the multi-manifold setting 84
4.4 A fast algorithm for computing K-nearest neighbors in the p-WSPD 90
4.5 Experimental Results 96
Bibliography 104

Chapter 1

Introduction

1.1 Overview of this Dissertation

The structure of this dissertation is as follows. In this Chapter (*i.e.* Chapter 1) we collect some preliminary material, which is likely well-known to experts. In particular, we discuss basic notions from random graph theory, spectral graph theory and compressive sensing. We also survey the literature on various clustering problems for graphs, and highlight recent progress made in the theory of clustering for the stochastic block model.

In Chapter 2 we introduce the idea of clustering as a compressive sensing problem, and present the main algorithmic contribution of this work, namely the `SingleClusterPursuit` algorithm. In Chapter 3 we present the `DynamicClusterPursuit` algorithm, which uses similar ideas to track the evolution of a cluster in a network which is changing with time. Finally, in Chapter 4 we discuss some work on the problem of creating an auxiliary graph, G, from a Euclidean data set, $\mathcal{X} \subset \mathbb{R}^D$ so that graph-based clustering algorithms can be used on G in order to cluster $\mathcal{X}$.

1.2 Graphs

In this dissertation we shall mostly restrict our attention to simple and undirected graphs G with vertex set V and edge set E. There are two exceptions to this: in Chapter 3 we consider dynamic graphs while in §4.4 we briefly consider directed graphs. In all cases our graphs will be finite, hence we shall identify the vertex set V with $[n] := \{1, \ldots, n\}$ and

denote an edge between vertices i and j as $\{i, j\}$. A shall denote the adjacency matrix of G, defined as $A_{ij} = 1$ if $\{i, j\} \in E$ and $A_{ij} = 0$ otherwise. We shall allow our graphs to have non-negatively weighted edges, in which case $A_{ij} \geq 0$ will denote the weight of the edge $\{i, j\}$. By convention, an edge weight of zero is the same as not having that particular edge. By d_i we mean the degree of the i-th vertex, computed as $d_i = \sum_j A_{ij}$. By $\mathrm{vol}(S)$, for any $S \subset V$ we mean the sum of degrees of all vertices in S, *i.e.* $\mathrm{vol}(S) = \sum_{i \in S} d_i$. For quantities such as d_i (and later λ_i and r_i) that are indexed by $i \in [n]$, let $d_{\max} := \max_i d_i$ and similarly $d_{\min} := \min_i d_i$. Denote by D the diagonal matrix whose (i, i) entry is d_i.

Definition 1.2.1 (Laplacians of graphs)**.** The *normalized, random walk* Laplacian is defined as $L = I - D^{-1}A$. We shall simply refer to it as *the Laplacian*. The *normalized, symmetric* Laplacian is: $L^{\mathrm{sym}} := I - D^{-1/2}AD^{-1/2}$. The unnormalized Laplacian, sometimes called the combinatorial Laplacian, is $L^{\mathrm{comb}} = D - A$.

Also related is the random walk *transition probability matrix* $P = AD^{-1}$, to be explored further in Theorem 2.3.2. We note that the spectra of L, L^{sym} and P are related:

Lemma 1.2.2. *For any matrix B with real eigenvalues let $\lambda_i(B)$ denote the i-th smallest eigenvalue, counted with multiplicity. Then $\lambda_i(L) = \lambda_i(L^{sym})$ and $\lambda_{n-i}(P) = 1 - \lambda_i(L)$*

Proof. Observe that $L = D^{-1/2}L^{\mathrm{sym}}D^{1/2}$, hence L and L^{sym} have the same spectrum. Similarly $P = D^{1/2}\left(I - L^{\mathrm{sym}}\right)D^{-1/2}$ hence P and $I - L^{\mathrm{sym}}$ have the same spectrum. Thus if λ is the i-th smallest eigenvalue of L^{sym} it is the i-th largest (and hence the $(n-i)$-th smallest) eigenvalue of $I - L^{\mathrm{sym}}$. □

For any $S \subset V$, we denote by G_S the induced sub-graph with vertices S and edges all $\{i, j\} \in E$ with $i, j \in S$. By A_{G_S} (resp. L_{G_S} and P_{G_S}) we shall mean the adjacency matrix (resp. Laplacian or transition probability matrix) of the graph G_S. Note that while A_{G_S} is a submatrix of A, L_{G_S} and P_{G_S} *are not submatrices of L and P* (the factor of D^{-1} gets in the way). Let us now give an admittedly vague definition of a cluster:

Definition 1.2.3. A cluster in a graph G is any subset of vertices $C \subset V$ such that there are many edges between vertices in C and few edges between C and $V \setminus C$.

Clusters are also frequently referred to as communities, particularly when the graph in question is derived from a social network. We shall say that a partition of V, namely $\mathbf{C} = \{C_1, \ldots, C_k\}$, is a *clustering* of V if each C_a forms a cluster. We do not claim that there is a unique clustering for any given graph. Throughout this dissertation we shall reserve the letter "k" to denote the number of clusters in a graph and "n" for the number of vertices.

There are many measures of "goodness" of cluster. In Definition 1.2.4 we collect several such measures. Before doing so, let us introduce the notation $E(S_1, S_2)$ to denote the set of all edges between two sets $S_1, S_2 \subset V$. If G is a weighted graph, $E(S_1, S_2)$ will denote the sum of the weights of all edges between S_1 and S_2: $E(S_1, S_2) = \sum_{\substack{i \in S_1, j \in S_2 \\ \{i,j\} \in E}} A_{ij}$.

Definition 1.2.4. Let $C \subset V$ be a cluster of G:

1. The *cut* of C is defined as $E(C, V \setminus C)$.
2. The *ratio cut* of C is defined as $\dfrac{E(C, V \setminus C)}{|C||V \setminus C|}$
3. The *normalized cut* of C is defined as $\dfrac{E(C, V \setminus C)}{\mathrm{vol}(C)\mathrm{vol}(V \setminus C)}$
4. The *conductance* of C is defined as $\dfrac{E(C, V \setminus C)}{\mathrm{vol}(C)}$

where for any finite set S, by $|S|$ we mean its cardinality. There are also measures, for example modularity, that refer to a baseline model of random graphs without clusters, such as the Erdős - Rènyi model to be introduced in the next section. We remark that optimizing different quantities selects different types of clusters. For example, minimizing ratio cut or normalized cut yields balanced clusters, where $|C| \sim |V \setminus C|$ while minimizing conductance yields smaller, tighter clusters. One can easily extend some of these measures to reflect the goodness of a clustering, $\mathbf{C}$:

Definition 1.2.5. Let $\mathbf{C}$ be a clustering of G:

1. The ratio cut of $\mathbf{C}$ is defined as $\frac{1}{2}\sum_{a=1}^{k}\frac{E(C_a, V \setminus C_a)}{|C_a|}$.

2. The normalized cut of $\mathbf{C}$ is defined as $\frac{1}{2}\sum_{a=1}^{k}\frac{E(C_a, V \setminus C_a)}{\mathrm{vol}(C_a)}$.

1.2.1 Random Graphs

Before proceeding, it is useful to establish two probabilistic models of graphs. Technically, a probabilistic model is a probability distribution, μ, on the set of all graphs on n vertices, with respect to which graphs are sampled. As the graphs in question all have the same vertex set, what μ really indicates is which edges to include. So, we shall follow the convention, typical in the literature, of not specifying μ explicitly, but rather specifying the probability of including each potential edge $\{i, j\}$ in a graph drawn from a particular model.

Definition 1.2.6. We say $G = (V, E)$ is drawn from the Erdős - Rènyi model on n vertices with parameter p (written $G \sim \mathrm{ER}(n, p)$) if $V = [n]$ and $\mathbb{P}[\{i, j\} \in E] = p$ for $i, j \in V$, with all such probabilities being independent.

If $G \sim \mathrm{ER}(n, p)$ we shall frequently abuse terminology slightly and refer to G as "an Erdős - Rènyi graph". Such graphs are extremely homogeneous in the sense that for any subset $S \subset V$ the number of edges between S and $V \setminus S$ is almost always close to its expected value, $p|S||V \setminus S|$. As such, Erdős - Rènyi graphs are very unlikely to contain clusters. In order to benchmark the performance of clustering algorithms (as well as to guide their development) it is useful to have a probabilistic model of graphs with clusters.

Definition 1.2.7. Let $\mathbf{n} = (n_1, \ldots, n_k)$ be a vector of positive integers, and let P be a $k \times k$ symmetric matrix with $P_{ab} \in [0, 1]$ for all a, b. We say a graph $G = (V, E)$ is drawn from the Stochastic Block Model (written $G \sim \mathrm{SBM}(\mathbf{n}, P)$) if there exists a partition $V = C_1 \cup C_2 \ldots \cup C_k$ with $|C_a| = n_a$ such that any vertices $i \in C_a$ and $j \in C_b$ are connected by an edge with probability P_{ab}, and all edges are inserted independently.

The first usage of the term "Stochastic Block Model" is usually attributed to Holland et al. [49] although there are certainly earlier instances of researchers studying similar models of graphs with planted clusters. For a thorough and modern overview, we recommend [2]. We remark that in [2] and elsewhere, a slightly more general definition is given where it is only required that the expected value of $|C_a|$ is n_a, but the above shall suffice for our purposes. In §1.3.3 we discuss recent work characterizing the values of the parameters p and q for which the clusters are recoverable given a single $G \sim \mathrm{SBM}(\mathbf{n}, P)$.

In the special case where all the n_a are equal, $P_{aa} = p$ for all a and $P_{ab} = q$ for all $a \neq b$ we say that G is drawn from the *Symmetric Stochastic Block Model*, and write $G \sim \mathrm{SSBM}(n, k, p, q)$. In this case the clusters are all of size $n_0 := n/k$.

Definition 1.2.8. For any two functions $f(n)$ and $g(n)$ we say that:

- $f(n) = O(g(n))$ if there exists constants C and K such that for all $n \geq K$ we have $f(n) \leq Cg(n)$.
- $f(n) = o(g(n))$ if for every $\epsilon > 0$ there exists a constant K such that for all $n \geq K$ we have $|f(n)| \leq \epsilon g(n)$.

Alternatively, observe that $f(n) = o(g(n))$ if and only if the ratio $f(n)/g(n)$ goes to zero as $n \to \infty$. In this dissertation we shall frequently use $o(1)$ to denote a quantity that goes to zero as $n \to \infty$. By $O(1)$ we shall denote a quantity which is eventually bounded.

Definition 1.2.9. If a certain property holds for graphs drawn from any probabilistic model of graph on n vertices with probability $1 - o(1)$, we say that this property holds *almost surely*, or a.s. [1]

[1] Technically, what we mean here is that the property in question holds *asymptotically almost surely*, or a.a.s. We adopt a slight abuse of terminology, common in the random graph theory literature, in referring to this phenomenon without the word "asymptotically".

Determining which properties hold almost surely is one of primary interest in the study of random graphs. It frequently turns out that there is a "threshold phenomenon" whereby if a parameter (for example, p in the Erdős - Rènyi model) is above a certain value then a property holds almost surely, but if it is below this value then the property does not hold, almost surely. For example, graphs drawn from $\mathrm{ER}(n, p)$ are connected, almost surely, if $p = c\log(n)/n$ with $c > 1$ and have two or more components, almost surely, if $c < 1$. In §2.2 we exploit such phenomena to ensure that the degrees and eigenvalues of Erdős - Rènyi graphs are suitably concentrated near their expected value.

1.2.2 Spectral Graph Theory

Spectral graph theory refers to the study of a graph G via the eigenvalues and eigenvectors of a matrix associated to G, typically the adjacency matrix A or one of the Laplacian matrices described in Definition 1.2.1. A good reference is [24]. Of primary interest to us is the following theorem:

Theorem 1.2.10. *Let $C_1, \ldots, C_k$ denote the connected components of a graph G. Then the cluster indicator vectors $\mathbf{1}_{C_1}, \ldots, \mathbf{1}_{C_k}$ are all eigenvectors of L associated to the eigenvalue 0. Equivalently, $\mathbf{1}_{C_1}, \ldots, \mathbf{1}_{C_k}$ is a basis for the kernel of L.*

Proof. See proposition 4 of [87]. □

To be clear, for any $S \subset V = [n]$, the vector $\mathbf{1}_S \in \mathbb{R}^n$ is defined as $(\mathbf{1}_S)_i = 1$ if $i \in S$ and $(\mathbf{1}_S)_i = 0$ otherwise. Eigenvectors of L are also of practical interest. In fact, finding them is the key step in the popular spectral clustering family of algorithms (see [78, 73, 87]). Theorem 1.2.10 is frequently used in the theoretical justification of the success of such algorithms [73, 87].

Given a probabilistic model of graphs, such as the two introduced in §1.2.1, it is an interesting question to determine the distribution of the eigenvalues of L for graphs drawn

from this model. Pursuing this question leads one quickly to deep questions on random matrices and sums of almost independent random variables. We return to this problem in §2.2.

1.3 Clustering Algorithms

We shall refer to the problem of partitioning the vertex set V of a graph into clusters $C_1, \ldots, C_k$ such that $V = \cup_a C_a$ and $C_a \cap C_b = \emptyset$ for all $a \neq b$ as the *partitional clustering problem*. We shall reserve the word "clustering" for a set of clusters $\mathbf{C} = \{C_1, \ldots, C_k\}$ satisfying these two properties (*i.e.* they partition V). The theory of partitional clustering is well developed, and there are many different algorithmic approaches to this problem. We refer the reader to the comprehensive surveys [39, 70, 52] for an overview and historical perspective. An obvious variation of this problem is to relax the condition that the clusters be disjoint, *i.e.* to allow for *overlapping clusters*. The theory behind this is less developed, and we refer the reader to [1] for further information.

Arguably the most well-known approach to partitional clustering is *spectral clustering*, introduced by Fiedler [37] for the two-cluster case, and popularized for the k cluster case by the papers of Shi and Malik [78], and Ng, Jordan and Weiss [73]. We refer the interested reader also to the excellent survey article [87]. Pseudocode for the algorithm of [73] is presented as Algorithm 1, while pseudocode for the algorithm of [78] is presented as Algorithm 2.

Algorithm 1 Normalized Spectral Clustering according to [73], as presented in [87]

Input: Adjacency matrix $A \in \mathbb{R}^{n \times n}$, number of clusters k.
(1) Computed the (symmetric) normalized Laplacian $L^{\text{sym}} \in \mathbb{R}^{n \times n}$
(2) Find the k eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ of L^{sym} corresponding to the k smallest eigenvalues.
(3) Let $U \in \mathbb{R}^{n \times k}$ be the matrix with columns $\mathbf{u}_1, \ldots, \mathbf{u}_k$.
(4) Let $T \in \mathbb{R}^{n \times k}$ be the matrix formed by scaling the rows of U to norm 1. Let $\mathbf{r}_i \in \mathbb{R}^k$, for $i = 1, \ldots, n$ denote the i-th row of T.
(5) Cluster the points $\{\mathbf{r}_1, \ldots, \mathbf{r}_n\} \subset \mathbb{R}^k$ into k clusters $B_1, \ldots, B_k$ using k-means.
Output: Clusters $C_1, \ldots, C_k$ with $C_a = \{i : \mathbf{r}_i \in B_a\}$

Algorithm 2 Normalized Spectral Clustering according to [78], as presented in [87]

Input: Adjacency matrix $A \in \mathbb{R}^{n\times n}$, number of clusters k.
(1) Computed the (random-walk) normalized Laplacian L.
(2) Find the k eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_k$ of L^{sym} corresponding to the k smallest eigenvalues.
(3) Let U be the matrix with columns $\mathbf{u}_1, \dots, \mathbf{u}_k$. Let $\mathbf{r}_i \in \mathbb{R}^k$, for $i = 1, \dots, n$ denote the i-th row of U.
(4) Cluster the points $\{\mathbf{r}_1, \dots, \mathbf{r}_n\} \subset \mathbb{R}^k$ into k clusters $B_1, \dots, B_k$ using k-means.
Output: Clusters $C_1, \dots, C_k$ with $C_a = \{i : \mathbf{r}_i \in B_a\}$

We note that there has also been significant progress, on both the algorithmic and theoretical side, on the problem of turning Euclidean data $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ into graphical data for the purpose of applying a graph-based clustering algorithm. We explore this issue further in Chapter 4, but for now let us mention that in [73] it is recommended that from $\mathcal{X}$ a weighted graph is formed with $V = [n]$ and $A_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/\sigma^2)$ for a user specified parameter σ.

1.3.1 Cluster Extraction

A more refined clustering problem, as compared to partitional clustering, is the problem of finding a single cluster, C, given a few representative members, $\Gamma \subset C$, which we refer to as *seed vertices*. This was first proposed in the computer science literature by Spielman and Teng [80, 81] under the name *local graph clustering*. Their motivation was to extend clustering techniques to graphs, G, that were too large for conventional partitional clustering techniques, and as such they were interested in algorithms with complexity scaling at most linearly in the number of vertices in G. Their proposed algorithm, `Nibble`, finds a cluster C in time $O(|C|\log^6(n)/\phi^4)$ (here ϕ is the conductance of C defined in Definition 1.2.4) and is a motivating example of a *diffusion based local clustering algorithm*. Further example of such algorithms include `PageRank-Nibble` [8] and heat kernel based approaches [25, 54] which we discuss further in §2.3.

Another related line of inquiry, presented in the statistics literature, is the problem of *cluster extraction*. This was motivated in [96] by the task of distinguishing a cluster, C, in some graph G, from a background of vertices that do not belong in any cluster. This was further developed in [89], where the algorithm `ESSC` is proposed. Note that in both the aforementioned papers the emphasis is on finding *significant* clusters, *i.e.* subsets $C \subset V$ with a significant difference between in-cluster and out-of-cluster connectivity, and in particular `ESSC` will return the empty set if it determines that no such cluster exists. Moreover, neither of the two approaches mentioned above requires a set of seed vertices. However, this additional flexibility is gained at the cost of slower run time, and in practice we have found both approaches to be unsuitable for large graphs.

A final family of approaches to this problem are the *local spectral methods*. These pursue the idea, first expressed by Mahoney, Orecchia and Vishnoi [61], of finding a local analogue of the second eigenvector of the Laplacian that is so important in spectral clustering. In the work of Bindel, He, Hopcroft, Kloster, Li and Shi [46, 58, 59, 79] the approach is to first subsample a graph $G_s \subset G$ that has much fewer vertices than G but that is very likely to contain the cluster C containing the seed vertices Γ, and then use spectral-type methods to extract C from G_s.

For consistency, we shall refer to all algorithms mentioned in this section as cluster extraction algorithms. We emphasize that any cluster extraction algorithm can, in principle, be iterated to yield a clustering of the entire vertex set V. That is, if one proceeds by removing all clusters previously found from G and then extracting the next one, one ends up with a partition of V. On the other hand, not removing already-found clusters allows for overlapping clusters. Thus cluster extraction generalizes both the partitional and overlapping clustering problems.

1.3.2 Semi-Supervised Clustering

Yet another variant on the classical clustering problem is *semi-supervised clustering*. In this set-up it is assumed that the cluster memberships of a small subset of the vertices of G are known *a priori*. For consistency with §1.3.1 we shall denote these vertices (the "labeled data" in the parlance of machine learning) as $\mathbf{\Gamma} = \{\Gamma_1, \ldots, \Gamma_k\}$ where $\Gamma_a \subset C_a$. Clearly, one would hope to be able to use this labeled data to achieve a more accurate clustering. Many such algorithms take a *variational approach*, whereby a clustering $\mathbf{C}$ is sought which minimizes a function of the form

$$E(\mathbf{C}) = R(\mathbf{C}) + F(\mathbf{C})$$

Here $R(\cdot)$ is a regularizing term, which penalizes partitions that are not smooth—that is, partitions which place vertices i and j with high affinity (*i.e.* large value of A_{ij}) into different clusters. $F(\cdot)$ is a fidelity term which penalizes clusterings that divide a set of labelled data Γ_a into multiple clusters. Various choices for the precise form of R and F are possible.

One family of variational semi-supervised clustering algorithms are the MBO-type schemes [64, 65, 63, 51], where the regularizing term is a discrete analog of the Merriman, Bence and Osher (MBO) energy that is used in approximating the mean curvature motion of interfaces. Another example is the regional force algorithms introduced in [94].

1.3.3 Partitional Clustering Algorithms and the Stochastic Block Model

We have introduced a large number of algorithms for several variants of the clustering problem, but we have not yet discussed how well they work. In order to make statements about how likely a certain algorithm is to succeed, one needs a probabilistic model of graphs with clusters, such as the stochastic block model introduced in §1.2.1. Following the conventions established in the survey article of Abbe [2], we distinguish between three types of recovery. Note that these refer to the *partitional clustering* problem.

Definition 1.3.1. Let $\mathcal{A}$ be a partitional clustering algorithm. For any input $G \sim \text{SBM}(\mathbf{n}, P)$ let $\mathbf{C}^{\#} = (C_1, \ldots, C_k)$ denote the output. We say that:

1. $\mathcal{A}$ solves the *exact recovery problem* if $\mathbb{P}\left[C_a^{\#} = C_a \text{ for } a = 1, \ldots, k\right] = 1 - o(1)$.
2. $\mathcal{A}$ solves the *almost exact recovery problem* if $\mathbb{P}\left[\frac{\sum_{a=1}^{k} |C_a \setminus C_a^{\#}|}{n} = o(1)\right] = 1 - o(1)$
3. $\mathcal{A}$ solves the *α-partial recovery problem* if $\mathbb{P}\left[\frac{\sum_{a=1}^{k} |C_a \setminus C_a^{\#}|}{n} \leq \alpha\right] = 1 - o(1)$ for some $\alpha \in (0, 1)$.

In other words, almost exact recovery asks for the fraction of misclassified vertices to go to zero, while α-partial recovery asks for the fraction of misclassified vertices to remain bounded by α. Not all values of α are appropriate here—choosing α too high would mean that even randomly assigning vertices to clusters would solve α-partial recovery. We refer to [2] for further discussion on this. For future reference we define a version of almost exact recovery suited to the cluster extraction problem:

Definition 1.3.2. Let $\mathcal{A}$ be a cluster extraction algorithm. For any input $G \sim \text{SBM}(\mathbf{n}, P)$ and $\Gamma \subset C_a$ let $C_a^{\#}$ denote the output of $\mathcal{A}$. We say that $\mathcal{A}$ solves the *almost exact cluster extraction problem* if $\mathbb{P}\left[\frac{|C_a \triangle C_a^{\#}|}{|C_a|} = o(1)\right] = 1 - o(1)$

Here $\triangle$ denotes the symmetric difference, defined for any two sets A, B as $A \triangle B = (A \setminus B \cap A) \cup (B \setminus A \cap B)$.

Recently, there has been a flurry of activity in determining for which parameter values the problems 1–3 in Definition 1.3.1 are solvable. An excellent overview of this work can be found in [2]. One can summarize this line of research with the following theorems:

Theorem 1.3.3. *Suppose that $a > b \geq 0$, where a and b may depend on n. Almost exact recovery is possible for SSBM$(n, k, a/n, b/n)$ if and only if:*

$$\frac{(a-b)^2}{k(a + (k-1)b)} \to \infty \text{ as } n \to \infty$$

This result was proved for the case $k = 2$ by Mossel, Neeman and Sly in [67] and later extended by the same authors in [68] to arbitrary k. In between the publication of these two papers, Abbe and Sandon [4] proved a slightly more general result on almost exact recovery for the (not necessarily symmetric) stochastic block model. Both groups of researchers provided new algorithms as part of their proof. The algorithm of Mossel, Neeman and Sly has computational complexity $O(n \log^2(n))$ although by their own admission the implied constant is very large, and they do not implement this algorithm. The algorithm of Abbe and Sandon, called `SphereComparison` runs in time $O(n^{1+o(1)})$ although the version presented in [4] required *a priori* knowledge of the parameters k, a and b. A later version, `AgnosticSphereComparison`, presented in [5], does not require such advanced knowledge and also runs in time $O(n^{1+o(1)})$.

Theorem 1.3.4. *Suppose that $a > b \geq 0$, where a and b may depend on n. Exact recovery for SSBM$(n, k, a\log(n)/n, b\log(n)/n)$ is possible if and only if*

$$\frac{1}{k}\left(\sqrt{a} - \sqrt{b}\right)^2 > 1$$

The $k = 2$ case of this Theorem is proved in [67] and also [3]. The case of $k > 2$ is covered in [4], although again knowledge of the parameters k, a and b is required. This obstacle is removed in [5], and the algorithm presented there has complexity $O(n^{1+\epsilon})$ for any fixed $\epsilon > 0$ (although the smaller the ϵ the less accurate the algorithm for smaller values of n). Essentially, what Abbe and Sandon propose is a two-step algorithm where in the first step `AgnosticSphereComparison` is used to find a good clustering of G with a small number of vertices misclassified. A second algorithm, `AgnosticDegreeProfiling` then "cleans up" this preliminary clustering. Although a single numerical experiment on a moderately sized graph (~ 1000 vertices) with this algorithm is alluded to in [5], to the best of the author's knowledge no extensive benchmarking of this approach has been done.

1.4 Compressive Sensing

We shall say that a vector $\mathbf{x} \in \mathbb{R}^N$ is *sparse* if it has few non-zero entries relative to its length. We follow the convention of defining the "0 quasi-norm" as $\|\mathbf{x}\|_0 = |\{x_i : \ x_i \neq 0\}|$. Motivated by problems of signal acquisition ("sensing") and compression for storage or transmission arising in signal processing, it is frequently of interest to consider the problem:

$$\text{argmin}\{\|\mathbf{x}\|_0 : \quad \mathbf{x} \in \mathbb{R}^N, \ \Phi\mathbf{x} = \mathbf{y}\} \tag{1.1}$$

$\Phi \in \mathbb{R}^{m \times N}$ is called the sensing matrix and typically $m < N$, making the linear system $\Phi\mathbf{x} = \mathbf{y}$ underdetermined. Key to the development of `SingleClusterPursuit` and `DynamicClusterPursuit` will be considering the situation where $m > N$, but Φ has a large kernel.

Problem (1.1) is NP-Hard [71] which means that, assuming P $\neq$ NP, there does not exist a polynomial time algorithm to solve it. Instead, one could consider the convex relaxation of (1.1):

$$\text{argmin}\{\|\mathbf{x}\|_1 : \quad \mathbf{x} \in \mathbb{R}^N, \ \Phi\mathbf{x} = \mathbf{y}\} \tag{1.2}$$

or the related problem:

$$\text{argmin}\,\{\|\Phi\mathbf{x} - \mathbf{y}\|_2 : \quad \|\mathbf{x}\|_0 \leq s\} \tag{1.3}$$

The origin of the field of compressive sensing (also called compressed sensing, or compressive/compressed sampling) was the realization that if $\Phi \in \mathbb{R}^{m \times N}$ is a random matrix and $\mathbf{y} = \Phi\mathbf{x}^*$ with $\|\mathbf{x}^*\|_0 = s$ such that $m = O(s \log(N/s))$ then both Problems 1.2 and 1.3 have a unique solution and moreover this solution coincides with the solution to (1.1), *i.e.* $\mathbf{x}^*$. The papers [18] and [35], both published in 2006, are generally credited as the first to explicitly make this connection. Three main questions now arise:

1. Are there efficient and accurate algorithms for solving problems (1.2) and (1.3)?

2. From what probability distribution should one draw Φ?

3. Is this set-up robust to noise? That is, suppose one acquires a signal polluted with some random noise: $\mathbf{y} = \Phi\mathbf{x}^* + \mathbf{e}$. If $\|\mathbf{e}\|_2$ is small, can one guarantee that the solution to (1.2) (resp. (1.3)) is close to $\mathbf{x}^*$?

We address the first question in the next subsection, the second briefly in §1.4.2 and the third in §1.4.3.

1.4.1 Algorithmic Approaches

As (1.2) is a convex problem, one may essentially use any convex optimization algorithm to solve it, although some perform better than others. We refer to [41] for further discussion. In Chapter 3 we shall use the ℓ_1 minimization algorithm described in [56] to solve (1.2). As in the literature, we shall refer to problem (1.2) as *basis pursuit*. Algorithms for Problem (1.3) may roughly be divided into two camps: thresholding algorithms (for example iterative hard thresholding [14] and hard thresholding pursuit [40]) and greedy algorithms. We shall explore greedy algorithms further in this section, but first we introduce two important concepts used in the analysis of compressive sensing algorithms.

Definition 1.4.1. Denote the i-th column of Φ as $\varphi_i \in \mathbb{R}^m$, so that $\Phi = [\varphi_1, \ldots, \varphi_N]$. Define the coherence of Φ, written $\mu(\Phi)$, as:

$$\mu(\Phi) := \max_{\substack{i,j\in[N] \\ i\neq j}} \frac{|\langle \varphi_i, \varphi_j \rangle|}{\|\varphi_i\|_2 \|\varphi_j\|_2}$$

Note that $\mu(\Phi) \in [0, 1]$. In many sources when coherence is defined it is assumed that the columns φ_i are normalized, in which case dividing by $\|\varphi_i\|_2\|\varphi_j\|_2$ is unnecessary.

Definition 1.4.2. The s Restricted Isometry Constant (s-RIC) of $\Phi \in \mathbb{R}^{m\times N}$, written $\delta_s(\Phi)$, is defined to be the smallest value of $\delta > 0$ such that, for all $\mathbf{x} \in \mathbb{R}^n$ with $\|\mathbf{x}\|_0 \leq s$, we have:

$$(1-\delta)\|\mathbf{x}\|_2^2 \leq \|\Phi\mathbf{x}\|_2^2 \leq (1+\delta)\|\mathbf{x}\|_2^2.$$

If $\delta_s(\Phi) < 1$ we often say that Φ has the *Restricted Isometry Property* (RIP).

Most proofs of correctness of compressive sensing algorithms rely on the RIP. For example:

Theorem 1.4.3 (Cai and Zhang, 2013 [17]). *Suppose that* $\|\mathbf{x}^*\|_0 = s$ *and that* $\mathbf{y} = \Phi\mathbf{x}^*$. *Then* $\mathbf{x}^*$ *is the unique solution to problem* (1.2) *with this data provided that* $\delta_s(\Phi) < 1/3$. *Moreover this bound is sharp in the sense that there exist* $\Phi \in \mathbb{R}^{m\times N}$ *with* $\delta_s(\Phi) \geq 1/3$ *and s-sparse* $\mathbf{x}^* \in \mathbb{R}^N$ *such that* $\mathbf{x}^*$ *is* not *the unique solution to problem* (1.2) *with data* Φ *and* $\mathbf{y} = \Phi\mathbf{x}^*$.

The archetypal greedy algorithm is Orthogonal Matching Pursuit (`OMP`), also referred to as the Orthogonal Greedy Algorithm, and analyzed comprehensively in [85]. This approach builds up a candidate support set $S^{\#}$ one index at a time by adding, on each iteration, the index i such that the inner product of φ_i and the current residual is maximized. `OMP` is presented formally as Algorithm 3. Note that if the sparsity of the target vector $\mathbf{x}^*$ is known then one may set $\ell = s$, otherwise we recommend taking an overestimate. The attribution of this algorithm is tricky, as it was discovered independently, and under many different names, by many researchers working in several different fields (see chapter 3 of [41]). A more refined greedy algorithm that we shall have use for later is `SubspacePursuit`, presented as Algorithm 4 and introduced by Dai and Milenkovic in [33]. Instead of adding one new index at the j-th iteration, `SubspacePursuit` considers a set of ℓ new indices and adds them to a stored set of ℓ indices, $S^{(j-1)}$, to obtain an active set of 2ℓ indices $\hat{S}^{(j)}$. An optimal 2ℓ sparse approximation to $\mathbf{y}$, namely $\mathbf{u}$, with support contained in $\hat{S}^{(j)}$ is then computed. Finally the iteration is completed by defining $S^{(j)}$ to be the indices of the ℓ largest-in-magnitude entries of $\mathbf{u}$, and this is stored for the next iteration. The number of iterations, J, is a user specified hyperparameter, but our analysis in Theorem 2.4.6 will show that $O(\log(N))$ iterations suffices. As for `OMP`, ℓ is a sparsity parameter, to be set equal to the sparsity of the target vector $\mathbf{x}^*$ if this is known. In line (3) of the "for" loop of Algorithm 4, $\mathcal{L}_s(\cdot)$ and

$\mathcal{H}_s(\cdot)$ are thresholding operators:

$$\mathcal{L}_s(\mathbf{v}) := \{i \in [n] : \ v_i \text{ among } s \text{ largest-in-magnitude entries in } \mathbf{v}\}$$

$$\mathcal{H}_s(\mathbf{v})_i := \begin{cases} v_i & \text{if } i \in \mathcal{L}_s(\mathbf{v}) \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 3 Orthogonal Matching Pursuit, or `OMP`, as presented in [41]

Inputs: measurement matrix Φ, measurement vector $\mathbf{y}$ and sparsity parameter ℓ
Initialization: $S^{(0)} = \emptyset$, $\mathbf{x}^{(0)} = \mathbf{0}$
for $j = 1$ to ℓ **do**
$i_j := \underset{i \in [N]}{\text{argmax}} \left\{ \left| \Phi^\top \left(\mathbf{y} - \Phi \mathbf{x}^{(j-1)} \right)_i \right| \right\}$
$S^{(j)} = S^{(j-1)} \cup \{i_j\}$
$\mathbf{x}^{(j)} = \underset{\mathbf{z} \in \mathbb{R}^N}{\text{argmin}} \left\{ \|\mathbf{y} - \Phi \mathbf{z}\|_2 : \ \text{supp}(\mathbf{z}) \subset S^{(j)} \right\}$
end for
Output: The ℓ sparse vector $\mathbf{x}^\# = \mathbf{x}^{(\ell)}$

Algorithm 4 `SubspacePursuit`, as presented in [33]

Inputs: measurement matrix Φ, measurement vector $\mathbf{y}$, sparsity parameter ℓ and number of iterations J.

Initialization:
(1) $S^{(0)} = \mathcal{L}_s(\Phi^\top \mathbf{y})$.
(2) $\mathbf{x}^{(0)} = \text{argmin}_{\mathbf{z} \in \mathbb{R}^N} \{\|\mathbf{y} - \Phi \mathbf{z}\|_2 : \ \text{supp}(\mathbf{x}) \subset S^{(0)}\}$
(3) $\mathbf{r}^{(0)} = \mathbf{y} - \Phi \mathbf{x}^{(0)}$
for $j = 1 : J$ **do**
(1) $\hat{S}^{(j)} = S^{(j-1)} \cup \mathcal{L}_\ell \left(\Phi^\top \mathbf{r}^{(j-1)} \right)$
(2) $\mathbf{u} = \underset{\mathbf{z} \in \mathbb{R}^N}{\text{argmin}} \{\|\mathbf{y} - \Phi \mathbf{z}\|_2 : \ \text{supp}(\mathbf{x}) \subset \hat{S}^{(j)}\}$
(3) $S^{(j)} = \mathcal{L}_\ell(\mathbf{u})$ and $\mathbf{x}^{(j)} = \mathcal{H}_\ell(\mathbf{u})$
(4) $\mathbf{r}^{(j)} = \mathbf{y} - \Phi \mathbf{x}^{(j)}$
end for

1.4.2 Matrices which satisfy the RIP

As should be clear from the previous subsection, good matrices for compressive sensing are those that satisfy the RIP. As alluded to earlier, a significant advance in the development of compressive sensing was the realization that random matrices satisfy the RIP, with

overwhelming probability. Specifically, we mention the following results of Baraniuk et al, Mendelson et al and Candes et al. Recall that we say that a random variable X is *subgaussian* if there exist constants σ and C such that $\mathbb{P}[|X| \geq t] \leq Ce^{-\sigma t^2}$ for all $t > 0$. Example 1.4.4 presents two standard examples of subgaussian random variables.

Example 1.4.4. 1. If X takes the values $+1$ and -1 with equal probability we call X a *Rademacher random variable*. Such an X is trivially subgaussian.

2. If $X \sim \mathcal{N}(0,1)$, *i.e.* the normal distribution, then X is subgaussian.

Theorem 1.4.5. *Suppose that $\Phi \in \mathbb{R}^{m \times N}$ is such that $\Phi = \frac{1}{\sqrt{m}}\tilde{\Phi}$ with each entry $\tilde{\Phi}_{ij}$ an independent subgaussian random variable. For any $\delta \in (0,1)$ there exist constants C_1, C_2 depending only on δ, such that for any $s \leq C_1 m \log(N/s)$ Φ has $\delta_s(\Phi) \leq \delta$ with probability greater than $1 - 2e^{-C_2 m}$.*

Proof. The theorem as stated comes from [10] but was first proved by [62]. See also [18]. □

1.4.3 Analysis of Perturbations

One of the reasons for the remarkable usefulness of compressive sensing is its robustness to error, both additive (*i.e.* in $\mathbf{y}$) and multiplicative (*i.e.* in Φ). More precisely, suppose that a signal $\hat{\mathbf{y}} = \hat{\Phi}\mathbf{x}^*$ is acquired, but that we do not know the sensing matrix $\hat{\Phi}$ precisely. Instead, we have access only to $\Phi = \hat{\Phi} + M$, for some small perturbation M. This models the scenario where a sensing matrix Φ is designed, and then implemented in hardware (for example as an MRI coil) where a certain amount of error becomes unavoidable. Suppose further that there is a small amount of noise in the measurement process, so that the signal we actually receive is $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{e}$. Can one hope to approximate a sparse vector $\mathbf{x}^*$ from $\mathbf{y}$ well, given only Φ? This question is answered in the affirmative by several authors, starting with the work of [47]. For `SubspacePursuit`, we have the following result of Li [57]:

Theorem 1.4.6. *Let* $\mathbf{x}^*$, $\mathbf{y}$ $\hat{\mathbf{y}}$, Φ *and* $\hat{\Phi}$ *be as above and suppose that* $\|\mathbf{x}^*\|_0 \leq s$. *For any* $t \in [n]$, *let* $\delta_t := \delta_t(\Phi)$. *Define the following constants:*

$$\epsilon_{\mathbf{y}} := \|\mathbf{e}\|_2/\|\hat{\mathbf{y}}\|_2 \text{ and } \epsilon_\Phi^s = \|M\|_2^{(s)}/\|\hat{\Phi}\|_2^{(s)}$$

where for any matrix B, $\|B\|_2^{(s)} := \max\{\|B_S\|_2 : \ S \subset [n] \text{ with } |S| = s\}$. *Define further:*

$$\rho = \frac{\sqrt{2\delta_{3s}^2(1+\delta_{3s}^2)}}{1-\delta_{3s}^2} \quad \text{and} \quad \tau = \frac{(\sqrt{2}+2)\delta_{3s}}{\sqrt{1-\delta_{3s}^2}}(1-\delta_{3s})(1-\rho) + \frac{2\sqrt{2}+1}{(1-\delta_{3s})(1-\rho)}$$

Assume $\delta_{3s} \leq 0.4859$ *and let* $\mathbf{x}^{(m)}$ *be the output of* `SubspacePursuit` *applied to problem* (1.3) *after* m *iterations. Then:*

$$\frac{\|\mathbf{x}^* - \mathbf{x}^{(m)}\|_2}{\|\mathbf{x}^*\|_2} \leq \rho^m + \tau\frac{\sqrt{1+\delta_s}}{1-\epsilon_\Phi^s}(\epsilon_\Phi^s + \epsilon_{\mathbf{y}}).$$

Proof. This is Corollary 1 in [57]. Note that our convention on hats is different to theirs. More precisely our Φ is their $\hat{\Phi}$, hence our ρ is their $\hat{\rho}$ and so on. □

In fact, it is easy to obtain bounds on the quantity $\|B\|_2^{(s)}$.

Lemma 1.4.7. *For any symmetric matrix* B *and any* $2 \leq s \leq n$ *we have that* $\lambda_{s-1}(B) \leq \|B\|_2^{(s)} \leq \lambda_s(B)$, *where* $\lambda_j(B)$ *denotes the* j*-th smallest eigenvalue of* B.

Proof. Observe that, for any matrix B,

$$\|B\|_2^{(s)} := \max_{\substack{S\subset[n]\\|S|=s}} \|B_S\|_2 = \max_{\substack{S\subset[n]\\|S|=s}} \sigma_{\max}(B_S),$$

where $\sigma_{\max}(B_S)$ denotes the maximum singular value of B_S. Because $\sigma_{\max}(B_S) = \sigma_s(B_S)$, by the interlacing theorem for singular values (cf. [83]) $\lambda_{s-1}(B) \leq \sigma_{\max}(B_S) \leq \lambda_s(B)$. □

Also of interest is the following theorem of Herman and Strohmer which quantifies the effect of perturbation on the RIC:

Theorem 1.4.8 ([47])**.** *Suppose that* $\Phi = \hat{\Phi} + M$. *Let* $\hat{\delta}_s$ *and* δ_s *denote the* s *restricted isometry constants of* $\hat{\Phi}$ *and* Φ *respectively. Then:*

$$\delta_s \leq (1+\hat{\delta}_s)\,(1+\epsilon_\Phi^s)^2 - 1.$$

Chapter 2

Semi-Supervised Cluster Pursuit

In this chapter we describe the first major contribution of this dissertation, namely an efficient and provably accurate approach to cluster extraction which we call `SingleClusterPursuit`. This work is contained in the paper [55], joint with Ming-Jun Lai. This chapter is laid out as follows. In §2.1 we discuss the motivation for and derivation of `SingleClusterPursuit`. In §2.2 we gather some results on concentration inequalities for degrees and eigenvalues of graphs drawn from the stochastic block model. In §2.3 we present the first step of our algorithm and analyze its probability of success on graphs drawn from the stochastic block model, using results from §2.2. In §2.4 we present the second step of our algorithm and analyze its performance and computational complexity. In particular, we prove that our algorithm solves the almost exact cluster extraction problem (see Definition 1.3.2). Also in §2.4 are several technical results on the RIP of Laplacians of graphs drawn from the stochastic block model which may be of independent interest. The computational complexity of `SingleClusterPursuit` is shown to be $O(d_{\max} \log(n)n)$, where $d_{\max}$ is the maximum vertex degree of G, in §2.5. In §2.6 we show via extensive numerical experiments that our algorithm yields state-of-the-art performance for cluster extraction in unweighted and weighted graphs by comparing it to other recent cluster extraction algorithms. In §2.6.6 we describe an iterated version of `SingleClusterPursuit` which may be used for the semi-supervised classification problem.

2.1 Cluster Extraction as Compressive Sensing

Recall the following result from §1.2.2:

Theorem 2.1.1. *Let $C_1, \dots, C_k$ denote the connected components of a graph G. Then the cluster indicator vectors $\mathbf{1}_{C_1}, \dots, \mathbf{1}_{C_k}$ form a basis for the kernel of L.*

Now suppose that G has clusters $C_1, \dots, C_k$. By definition, clusters have few edges between them, and so it is useful to write G as the union of two edge-disjoint subgraphs, defined as follows: let $G^{\text{in}} = (V, E^{\text{in}})$ have only edges between vertices in the same cluster, while $G^{\text{out}} = (V, E^{\text{out}})$ consist only of edges between vertices in different clusters. We emphasize that this is a theoretical construction, as in practice we of course cannot ascertain whether two vertices are in the same cluster without first solving the clustering problem, which is precisely what we are trying to do. Denote by A^{in} and L^{in} (resp. A^{out} and L^{out}) the adjacency matrix and Laplacian of G^{in} (resp. G^{out}). Similarly, d_i^{in} (resp. d_i^{out}) shall denote the degree of the vertex i in the graph G^{in} (resp. G^{out}). Note that $C_1, \dots, C_k$ are now the connected components of G^{in}, and so $L^{\text{in}}\mathbf{1}_{C_a} = 0$ for all $a \in [k]$. Note further that $\|\mathbf{1}_{C_a}\|_0 = |C_a|$ which is typically much smaller than n. Hence we can think of $\mathbf{1}_{C_a}$ as a *sparse solution to the linear system* $L^{\text{in}}\mathbf{x} = 0$ and hence one may consider using sparse recovery algorithms (See §1.4.1) to solve for $\mathbf{1}_{C_a}$. This naïve approach does not work, as $\mathbf{x} = 0$ is trivially a solution. However suppose one can find a superset of the cluster of interest: $C_a \subset \Omega \subset V$. Then:

Theorem 2.1.2. *Suppose that $C_a \subset \Omega \subset V$ and that G_{C_a} is connected. Suppose further that C_a is the only cluster contained in Ω. Define $\mathbf{y}^{in} = L^{in}\mathbf{1}_\Omega = \sum_{i\in\Omega} \ell_i^{in}$ and let $n_a := |C_a|$. Then $\mathbf{1}_{\Omega\setminus C_a}$ is the unique solution to:*

$$\operatorname*{argmin}_{\mathbf{v}\in\mathbb{R}^n} \left\{ \|L^{in}\mathbf{v} - \mathbf{y}^{in}\|_2 : \quad \|\mathbf{v}\|_0 \le |\Omega| - n_a \text{ and } supp(\mathbf{v}) \subset \Omega \right\} \tag{2.1}$$

Proof. First observe that $\mathbf{1}_{\Omega\setminus C_a}$ is feasible for the problem (2.1). Moreover:

$$L^{\text{in}}\mathbf{1}_{\Omega\setminus C_a} - \mathbf{y}^{\text{in}} = L^{\text{in}}\mathbf{1}_{\Omega\setminus C_a} - L^{\text{in}}\mathbf{1}_\Omega = L^{\text{in}}\left(\mathbf{1}_{\Omega\setminus C_a} - \mathbf{1}_\Omega\right) = -L^{\text{in}}\mathbf{1}_{C_a} = 0$$

Now, suppose that $\mathbf{z} \in \mathbb{R}^n$ is any other vector also satisfying $L^{\text{in}}\mathbf{z} - \mathbf{y}^{\text{in}} = 0$. We shall show that such $\mathbf{z}$ cannot be feasible for (2.1). Observe that:

$$L^{\text{in}}\mathbf{z} - \mathbf{y}^{\text{in}} = 0 \Rightarrow L^{\text{in}}(\mathbf{z} - \mathbf{1}_\Omega) = 0$$

$$\Rightarrow \mathbf{z} - \mathbf{1}_\Omega \in \ker(L^{\text{in}})$$

Because $\mathbf{z} - \mathbf{1}_\Omega \in \ker(L^{\text{in}})$, by Theorem 3.3.1, we have $\mathbf{z} - \mathbf{1}_\Omega = \sum_{b=1}^{k} \alpha_b \mathbf{1}_{C_b}$. Suppose that $\text{supp}(\mathbf{z}) \subset \Omega$ whence $\text{supp}(\mathbf{z} - \mathbf{1}_\Omega) \subset \Omega$. By assumption, C_a is the only cluster contained in Ω, hence $\mathbf{z} - \mathbf{1}_\Omega = \alpha_a \mathbf{1}_{C_a}$ thus $\mathbf{z} = \mathbf{1}_\Omega + \alpha_a \mathbf{1}_{C_a}$. Because $\mathbf{z} \neq \mathbf{1}_{\Omega \setminus C_a}$ we must have that $\alpha_a \neq -1$. But then $\text{supp}(\mathbf{z}) = \Omega$ and so $\|\mathbf{z}\|_0 = |\Omega| > |\Omega| - n_a$. □

It follows that given Ω, one can indeed find $\mathbf{1}_{\Omega \setminus C_a}$ by solving problem (2.1). One can then recover $\Omega \setminus C_a$ as $\text{supp}(\mathbf{1}_{\Omega \setminus C_a})$ and hence C_a as $\Omega \setminus \text{supp}(\mathbf{1}_{\Omega \setminus C_a})$. This leads to a novel approach for cluster extraction, assuming one can address the following two questions:

1. Can one find a good superset Ω of C_a? Clearly the smaller Ω is, the sparser $\mathbf{1}_{\Omega \setminus C_a}$ is, and hence the better the performance of sparse recovery algorithms on (2.1) will be.

2. If one replaces L^{in} and $\mathbf{y}^{\text{in}}$ with L and $\mathbf{y}$ to obtain the perturbed problem:

$$\underset{\mathbf{v} \in \mathbb{R}^n}{\text{argmin}} \{\|L\mathbf{v} - \mathbf{y}\|_2 : \ \|\mathbf{v}\|_0 \leq |\Omega| - n_a \text{ and } \text{supp}(\mathbf{v}) \subset \Omega\} \tag{2.2}$$

   will the solution $\mathbf{v}^{\#}$ obtained by solving (2.2) using a sparse recovery problem be a good enough approximation to $\mathbf{1}_{\Omega \setminus C_a}$? Specifically, is $\text{supp}(\mathbf{v}^{\#}) \approx \text{supp}(\mathbf{1}_{\Omega \setminus C_a}) = \Omega \setminus C_a$?

We are able to solve both of these problems, and the resulting algorithm, `SingleClusterPursuit` is presented as Algorithm 6, with an important sub-routine `RandomWalkThresholding` presented as Algorithm 5. Before continuing, let us establish the convention that we are always trying to find C_1, and that $n_1 := |C_1|$. In §2.3 we discuss `RandomWalkThresholding` further, and show that if $G \sim \text{SSBM}(n, k, p, q)$ for suitable values of p and q, then it returns an Ω such that the fraction of "missed vertices", *i.e.* $|C_1 \setminus C_1 \cap \Omega|/n_1$ goes to zero as n_1 goes to infinity.

Note that in practice one would not solve problem (2.2) directly. Instead, if L has columns $\ell_1, \ldots, \ell_n$ define the submatrix $L_\Omega := [\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_{|\Omega|}}]$ where $\Omega = \{i_1, \ldots, i_\Omega\}$. One can then solve the reduced problem:

$$\underset{\mathbf{x}\in\mathbb{R}^{|\Omega|}}{\operatorname{argmin}} \{\|L_\Omega \mathbf{x} - \mathbf{y}\|_2 : \ \|\mathbf{x}\|_0 \leq |\Omega| - n_a\} \tag{2.3}$$

and, assuming one has kept track of the indices in Ω, easily infer $\operatorname{supp}(\mathbf{v}^\#)$ from $\operatorname{supp}(\mathbf{x}^\#)$ where $\mathbf{x}^\#$ is the solution to (2.3). Henceforth we focus our attention on Problem (2.3), and we shall abuse notation slightly, writing $\operatorname{supp}(\mathbf{x}^\#)$ when we really mean $\operatorname{supp}(\mathbf{v}^\#)$.

In §2.4 we discuss the trickier issue of extracting C_1 from Ω. This involves a careful analysis of sparse recovery under total perturbation, as discussed in §1.4.3. We prove that if L_Ω has sufficiently small restricted isometry constant then `Single Cluster Pursuit` does indeed find a good approximation to $\mathbf{1}_{\Omega\setminus C_1}$ when solving (2.4). Note that this problem is slightly different to Problem 2.3. We discuss the discrepency in §2.4. The operator $\tilde{\mathcal{L}}$ referred to in step (3) of Algorithm 5 is a thresholding operator, similar to $\mathcal{L}$ defined in §1.4.1, but defined as:

$$\tilde{\mathcal{L}}_s(\mathbf{v}) := \{i \in [n] : \ v_i \text{ among } s \text{ largest entries in } \mathbf{v}\}$$

Algorithm 5 `RandomWalkThresholding`

Input: Adjacency matrix A, a thresholding parameter $\epsilon \in (0,1)$, seed vertices $\Gamma \subset C_1$,$\hat{n}_1 \approx n_1$ and depth of random walk t.
(1) Compute $P = AD^{-1}$ and let $\mathbf{v}^{(0)} = (1/\text{vol}(\Gamma))D\mathbf{1}_\Gamma$.
(2) Compute $\mathbf{v}^{(t)} = P^t\mathbf{v}^{(0)}$
(3) Define $\Omega = \tilde{\mathcal{L}}_{(1+\epsilon)\hat{n}_1}(\mathbf{v}^{(t)})$
Output: $\Omega = \Omega \cup \Gamma$

2.2 Concentration in Random Graphs

The proof of Theorems 2.3.2, 2.4.8 and 2.4.11 rely on two *concentration phenomena* in Erdős - Rènyi graphs where by concentration we mean that a random variable is within a small deviation of its expected value, with high probability. The first concerns the minimum and

Algorithm 6 `SingleClusterPursuit`

Input: Adjacency matrix A, a thresholding parameter $\epsilon \in (0,1)$, seed vertices $\Gamma \subset C_1$,$\hat{n}_1 \approx n_1$, depth of random walk t and rejection parameter $R \in [0,1)$.
(1) Let $\Omega =$ `RandomWalkThresholding`$(A, \epsilon, \Gamma, \hat{n}_1, t)$
(2) Compute $L = I - D^{-1}A$ and $\mathbf{y} = \sum_{i\in\Omega} \ell_i$.
(3) Let $\mathbf{x}^{(m)}$ be the solution to

$$\text{argmin}\{\|L_\Omega \mathbf{x} - \mathbf{y}\|_2 : \ \|\mathbf{x}\|_0 \leq 1.1\epsilon\hat{n}_1\} \tag{2.4}$$

obtained after $m = O(\log(n))$ iterations of `SubspacePursuit`
(4) Let $W^\# = \{i : \ x_i^{(m)} > R\}$
Output: $C_1^\# = \Omega \setminus W^\#$.

maximum degrees of an Erdős - Rènyi graph while the second concerns the eigenvalues of A and L for such a graph.

Theorem 2.2.1 (see [15, 16])**.** *Let $G \sim ER(n,q)$ with $q = (b + o(1))\log(n)/n$. There exist a function $\eta_\Delta(b)$ satisfying $0 < \eta_\Delta(b) < 1$ and $\lim_{b\to\infty} \eta_\Delta(b) = 0$ such that*

$$d_{max}(G) = (1 + \eta_\Delta(b))b\log n + o(1) \leq 2b\log(n) + o(1) \ \text{a.s.}$$

Theorem 2.2.2 (see [42], Theorem 3.4 (ii))**.** *If $G \sim ER(n_0,p)$ with $p = \omega\log(n_0)/n_0$ where $\omega \to \infty$, then $d_{\min}(G) = (1 - o(1))\omega\log(n_0)$ and $d_{\max}(G) = (1 + o(1))\omega\log(n_0)$ a.s.*

Theorem 2.2.3. *Suppose that $G \sim ER(n_0,p)$ with $p = \omega\log(n_0)$ where $\omega \to \infty$. Then we have almost surely*

1. $\lambda_{\max}(A) \leq (1 + o(1))\omega\log(n_0)$
2. $\lambda_i(A) \leq o(\omega\log(n_0))$ *for* $\lambda_i < \lambda_{\max}$
3. $|\lambda_i(L) - 1| \leq \sqrt{\frac{12}{\omega}} = o(1)$ *for all* $i > 1$

Proof. Parts 1 and 2 are Theorem 3 in [28]. Relating their notation to ours, we have that $m = w_{\min} = pn_0 = \omega \log n_0$. Theorem 4 in [28] shows that

$$|\lambda_i(L^{\text{sym}}) - 1| \leq \sqrt{\frac{6\log(2n_0)}{\omega \log(n_0)}}$$

By Lemma 1.2.2 L^{sym} and L have the same spectrum. Because $\log(2n_0) \leq 2\log(n_0)$ for $n_0 \geq 2$ we obtain part 3. □

2.2.1 Reducing from the SBM to the ER model

Let G^{in} and G^{out} be as in §2.1. If $G \sim \text{SSBM}(n, k, p, q)$ then G^{in} consists of k disjoint i.i.d graphs, $G_{C_a} \sim \text{ER}(n_0, p)$ where $n_0 := n/k$. The graph G^{out} is not an Erdős - Rènyi graph, as there is 0 probability of it containing an edge between two vertices in the same cluster (because we have removed them). However, we can profitably think of G^{out} as a subgraph of some $\widetilde{G^{\text{out}}} \sim \text{ER}(n, q)$. In particular, any upper bounds on the degrees of vertices in $\widetilde{G^{\text{out}}}$ are automatically bounds on the degrees in G^{out}. Thus, we have the following corollaries of Theorems 2.2.2 and 2.2.1:

Corollary 2.2.4. *If $G \sim SSBM(n, k, p, q)$ with $q = b\log(n)/n$ then $d^{out}_{\max}(G) \leq 2b\log n + o(1)$ a.s.*

Proof. Consider G^{out} as a subgraph of $\widetilde{G^{\text{out}}} \sim \text{ER}(n, q)$ and apply Theorem 2.2.1 □

Corollary 2.2.5. *If $G \sim SSBM(n, k, p, q)$ with $k = O(1)$ and $p = \omega\log(n_0)/n_0$ where $\omega \to \infty$, then $d^{in}_{\min}(G) = (1 - o(1))\omega\log(n_0)$ and $d^{in}_{\max}(G) = (1 + o(1))\omega\log(n_0)$ a.s.*

Proof. If $i \in C_a$ then $d_i^{\text{in}} = d_i(G_{C_a})$, where $G_{C_a} \sim \text{ER}(n_0, p)$. Clearly:

$$d^{\text{in}}_{\max}(G) = \max_i d_i^{\text{in}} = \max_a d_{\max}(G_{C_a})$$

By Theorem 2.2.2, $d_{\max}(G_a) = (1 + o_{n_0}(1))\omega\log(n_0)$ a.s. Note that the $d_{\max}(G_{C_a})$ are i.i.d random variables, and since we are taking a maximum over $k = \mathcal{O}(1)$ of them, it follows that $\max_a d_{\max}(G_{C_a}) \leq (1 + o_{n_0}(1))\omega\log(n_0)$ a.s. Moreover, as $n_0 = n/k$, $o_{n_0}(1) = o_n(1)$. The proof for $d^{\text{in}}_{\min}(G)$ is similar. □

Corollary 2.2.6. *Suppose that $G \sim SSBM(n, k, p, q)$ with $p = \omega \log(n_0)/n_0$ where $\omega \to \infty$, $q = b \log(n)/n$ and $k = O(1)$ and define $r_i := d_i^{out}/d_i^{in}$ for $i \in [n]$. Then $r_{\max} \leq d_{\max}^{out}/d_{\min}^{in} = o(1)$ a.s.*

Proof. First of all, it is clear that for any i, $d_i^{\text{out}}/d_i^{\text{in}} \leq d_{\max}^{\text{out}}/d_{\min}^{\text{in}}$. From Corollaries 2.2.4 and 2.2.5 we have:

$$\begin{aligned}\frac{d_{\max}^{\text{out}}}{d_{\min}^{\text{in}}} &\leq \frac{2b \log n + o(1)}{(1 - o(1))\omega \log(n_0)} = \frac{2b \log n + o(1)}{(1 - o(1))\omega(\log(n) - \log(k))} \quad \text{as } n = kn_0 \\ &= \frac{2b + o(1)}{(1 - o(1))\omega(1 - o(1))} = o(1) \text{ since } k = O(1) \text{ and } \omega \to \infty\end{aligned}$$

□

2.3 Finding good supersets

In Algorithm 5 we presented an algorithm for finding an $\Omega \subset V$ such that Ω is relatively small ($|\Omega| = (1 + \epsilon)n_1$ for a user specified parameter ϵ) but that captures almost all of C_1. It will be useful to introduce a name for such sets:

Definition 2.3.1. Let $C_1 \subset V$ be a cluster of G of size $|C_1| = n_1$ and let $\epsilon \in (0, 1)$. If $\Omega \subset V$ is such that $|\Omega| = (1 + \epsilon)n_1$, $|C_1 \setminus C_1 \cap \Omega| = o(n_1)$ and $C_a \not\subset \Omega$ for $a = 2, \ldots, k$ then we say that Ω is an ϵ-good superset for C_1.

Intuitively, Algorithm 5 works as follows. We start a random walk on the seed vertices, Γ, and run it for t steps where t is a user specified parameter. We then rank the vertices by the probability of being visited by this random walk, and return the $(1 + \epsilon)\hat{n}_1$ vertices with largest such probabilities. Here $\hat{n}_1$ is a third user specified parameter, and represents an estimate of the size of the cluster of interest.

`RandomWalkThresholding` has many similarities with diffusion based local clustering methods such as `Nibble` [80, 81], `PageRank-Nibble` [8], the approaches suggested by Chung and collaborators [25, 26, 29] and the Heat Kernel method of Kloster and Gleich [54]. There

is, however, a key difference. All of the aforementioned algorithms attempt to directly find a good cluster. Hence they all proceed by running a diffusive process (either a random walk or heat flow) starting at Γ and ranking the vertices by the probability of being visited by this process. They then perform a sweep cut (see [25] for a definition) and return the set Ω which optimizes some measure of goodness-of-cluster, typically conductance. As a result, they have a tendency to return small, tight clusters whose size does not scale with the size of the graph.

Our approach however is intended to be used as step one in a two-step clustering algorithm. As such, the focus of Algorithm 5 is not returning a good cluster, but returning an Ω *which contains a cluster of interest*. The cluster of interest, C_1, is then extracted from Ω by the next step of the algorithm (see 6). We note that the two-step approaches of Bindel, He, Hopcroft, Kloster, Li and Shi such as `LEMON`, `LOSP` and `LBSA` [46, 58, 59, 79] bear some resemblance to ours. However, the method we propose to extract C_1 from Ω is distinct from their methods. In addition, we analyze the likelihood of Ω containing the cluster of interest given that G is drawn from a particular probabilistic model (we consider the stochastic block model), something which is absent in the aforementioned papers. Specifically, we prove the following theorem:

Theorem 2.3.2. *Suppose that $G \sim SSBM(n, k, p, q)$ with parameters $p = \omega \log(n_0)/n_0$ and $q = b \log(n)/n$ where $\omega \to \infty$ and b is a fixed constant. Fix the depth of the random walk $t \geq 1$ and suppose that $\Gamma \subset C_1$ with $|\Gamma| = gn_1/\omega^{t-1}$ for any constant $g \in (0, 1)$. Let Ω be the output of Algorithm 5 given inputs any fixed $\epsilon \in (0, 1)$, seed vertices Γ, depth of random walk t and $\hat{n}_1 = n_1$. Then Ω is an ϵ-good superset for C_1 almost surely.*

Proof. Recall the definition of ϵ-good superset (Definition 2.3.1). By construction, $|\Omega| = (1 + \epsilon)n_1$ thus we need to verify that $|C_1 \setminus \Omega \cap C_1| = o(n_1)$. Observe that because $n_1 = |C_1| = |C_2| = \ldots = |C_k|$ for $G \sim \text{SSBM}(n, k, p, q)$, from this it will also follow that $C_a \not\subset \Omega$ for $a = 2, \ldots, k$.

Let $U = C_1 \setminus (C_1 \cap \Omega)$ denote the "missed" indices, and $W = \Omega \setminus (C_1 \cap \Omega)$ denote the "bad" indices (i.e. vertices in Ω that are not in C_1). Let $|U| = un_1$, in which case $|W| = (\epsilon + u)n_1$. We prove that $u = o(1)$.

By definition, Ω is the set of the $(1+\epsilon)n_1$ largest entries in $\mathbf{v}^{(t)} := (1/\mathrm{vol}(\Gamma))P^t D\mathbf{1}_\Gamma$. Because U is not in Ω, but W is, we must have $v_i^{(t)} \leq v_j^{(t)}$ for every $i \in U$ and $j \in W$. We sum first over $j \in W$ and then sum over $i \in U$ to obtain:

$$v_i^{(t)} \leq v_j^{(t)} \Rightarrow (\epsilon+u)n_1 v_i^{(t)} \leq \sum_{j\in W} v_j^{(t)} \Rightarrow (\epsilon+u)n_1 \sum_{i\in U} v_i^{(t)} \leq un_1 \sum_{j\in W} v_j^{(t)}.$$

It follows that:

$$\sum_{i\in U} v_i^{(t)} \leq \frac{u}{\epsilon+u} \sum_{j\in W} v_j^{(t)} \leq \sum_{j\in W} v_j^{(t)}. \tag{2.5}$$

Looking ahead, we shall show that if inequality (2.5) holds then $u = o(1)$.

We first show that the term on the left-hand side of inequality 2.5, *i.e.* the sum over the vertices in C_1 that were missed by Ω, is necessarily quite large. We do this by relating P to P^{in}, the random walk transition matrix for the graph G^{in}. Note that G^{in} is a disjoint union of graphs $G_{C_a} \sim \mathrm{ER}(n_0, p)$ hence this reduces the problem to a question in Erdős - Rènyi graphs. For every $i \in [n]$, define $q_i := d_i^{\mathrm{in}}/d_i$. Observe that $1/d_i = q_i/d_i^{\mathrm{in}}$ and thus $D^{-1} = D_{\mathrm{in}}^{-1}Q$ where Q is the diagonal matrix with (i,i)-th entry q_i. Now:

$$P = AD^{-1} = \left(A^{\mathrm{in}} + A^{\mathrm{out}}\right) D^{-1} = A^{\mathrm{in}} \left(D_{\mathrm{in}}^{-1}Q\right) + A^{\mathrm{out}}D^{-1} = P^{\mathrm{in}}Q + A^{\mathrm{out}}D^{-1}$$

Observe that P, $P^{\mathrm{in}}Q$ and $A^{\mathrm{out}}D^{-1}$ all have non-negative entries. It follows that for any non-negative vector $\mathbf{x}$:

1. $P\mathbf{x}$ and $P^{\mathrm{in}}Q\mathbf{x}$ are also non-negative.

2. $P\mathbf{x} \geq P^{\mathrm{in}}Q\mathbf{x}$, where the inequality should be interpreted componentwise.

One can extend the second point by iterated multiplication:

$$P^t\mathbf{x} \geq \left(P^{\mathrm{in}}Q\right)^t \mathbf{x} \geq q_{\min}^t \left(P^{\mathrm{in}}\right)^t \mathbf{x}$$

and again the inequality should be interpreted componentwise. Now:

$$\sum_{i\in U} v_i^{(t)} = \langle \mathbf{1}_U, \mathbf{v}^{(t)} \rangle = \frac{1}{\mathrm{vol}(\Gamma)} \langle \mathbf{1}_U, P^t D \mathbf{1}_\Gamma \rangle \geq \frac{1}{\mathrm{vol}(\Gamma)} \langle \mathbf{1}_U, q_{\min}^t \left(P^{\mathrm{in}}\right)^t D \mathbf{1}_\Gamma \rangle$$
$$= \frac{q_{\min}^t}{\mathrm{vol}(\Gamma)} \langle \mathbf{1}_U, \left(P^{\mathrm{in}}\right)^t D_{\mathrm{in}} \mathbf{1}_\Gamma \rangle \tag{2.6}$$

Observe that, because $U, \Gamma \subset C_1$, we may replace P^{in} and D_{in} in the above inner product with $P_{G_{C_1}}$ and $D_{G_{C_1}}$ without changing its value.

Our goal now is to bound the quantity $\langle \mathbf{1}_U, \left(P_{G_{C_1}}\right)^t D_{G_{C_1}} \mathbf{1}_\Gamma \rangle$. One can rearrange the iterated matrix product slightly:

$$\left(P_{G_{C_1}}\right)^t = \left(A_{G_{C_1}} D_{G_{C_1}}^{-1}\right)^t = A_{G_{C_1}} D_{G_{C_1}}^{-1} A_{G_{C_1}} D_{G_{C_1}}^{-1} \dots A_{G_{C_1}} D_{G_{C_1}}^{-1}$$
$$= D_{G_{C_1}}^{1/2} \left(D_{G_{C_1}}^{-1/2} A_{G_{C_1}} D_{G_{C_1}}^{-1/2}\right) \left(D_{G_{C_1}}^{-1/2} A_{G_{C_1}} D_{G_{C_1}}^{-1/2}\right) \dots \left(D_{G_{C_1}}^{-1/2} A_{G_{C_1}} D_{G_{C_1}}^{-1/2}\right) D_{G_{C_1}}^{-1/2}$$
$$= D_{G_{C_1}}^{1/2} \left(D_{G_{C_1}}^{-1/2} A_{G_{C_1}} D_{G_{C_1}}^{-1/2}\right)^t D_{G_{C_1}}^{-1/2}$$

Chung and Graham refer to the matrix $D_{G_{C_1}}^{-1/2} A_{G_{C_1}} D_{G_{C_1}}^{-1/2}$ as $M_{G_{C_1}}$ in [27]. Rearranging slightly:

$$\langle \mathbf{1}_U, \left(P^{\mathrm{in}}\right)^t D \mathbf{1}_\Gamma \rangle = \langle \mathbf{1}_U, \left(D_{G_{C_1}}^{1/2} M_{G_{C_1}}^t D_{G_{C_1}}^{-1/2}\right) D \mathbf{1}_\Gamma \rangle = \langle D_{G_{C_1}}^{1/2} \mathbf{1}_U, M_{G_{C_1}}^t D_{G_{C_1}}^{1/2} \mathbf{1}_\Gamma \rangle$$

In [27], Lemma 2, bounds on this inner product are provided [1] in terms of the second largest eigenvalue of $M_{G_{C_1}}$. Because $M_{G_{C_1}} = I - L_{G_{C_1}}^{\mathrm{sym}}$ it follows that the second largest eigenvalue of $M_{G_{C_1}}$ is $1-\lambda_2$, where λ_2 is the second *smallest* eigenvalue of $L_{G_{C_1}}^{\mathrm{sym}}$, equivalently $L_{G_{C_1}}$ (see Lemma 1.2.2). Using the aforementioned result of [27]:

$$\left| \langle D_{G_{C_1}}^{1/2} \mathbf{1}_U, M_{G_{C_1}}^t D_{G_{C_1}}^{1/2} \mathbf{1}_\Gamma \rangle - \frac{\mathrm{vol}(U)\mathrm{vol}(\Gamma)}{\mathrm{vol}(G_{C_1})} \right| \leq (1-\lambda_2)^t \sqrt{\mathrm{vol}(U)\mathrm{vol}(\Gamma)}$$

Returning to (2.6):

$$\sum_{i\in U} v_i^{(t)} \geq q_{\min}^t \left(\frac{\mathrm{vol}(U)}{\mathrm{vol}(G_{C_1})} - (1-\lambda_2)^t \sqrt{\frac{\mathrm{vol}(U)}{\mathrm{vol}(\Gamma)}} \right) \tag{2.7}$$

[1] In the aforementioned paper they consider a slightly different model of graph, namely graphs with a given degree sequence. However the proof of Lemma 2 goes through in our case without issue

We now consider the right hand side of (2.5), *i.e.* the sum over W. It is convenient to decompose $\mathbf{v}^{(t)}$ as $\mathbf{v}^{(t)} = \mathbf{v}^{(t)}_{C_1} + \mathbf{v}^{(t)}_{V\setminus C_1}$ where $\mathbf{v}^{(t)}_{C_1}$ is supported on C_1 and $\mathbf{v}^{(t)}_{V\setminus C_1}$ is supported on the complement of C_1. Because $W \subset V \setminus C_1$ we have that:

$$\sum_{j\in W} v_j^{(t)} \leq \sum_{j\in V\setminus C_1} |v_j^{(t)}| = \|\mathbf{v}^{(t)}_{V\setminus C_1}\|_1$$

Thus it remains to bound $\|\mathbf{v}^{(t)}_{V\setminus C_1}\|_1$, which represents the probability of a random walk starting at $\Gamma \subset C_1$ being outside of C_1 at the t-th step. Observe that:

$$\mathbf{v}^{(t)}_{V\setminus C_1} = A^{\text{in}}D^{-1}\mathbf{v}^{(t-1)}_{V\setminus C_1} + \left(A^{\text{out}}D^{-1}\mathbf{v}^{(t-1)}\right)_{V\setminus C_1}$$

where $\left(A^{\text{out}}D^{-1}\mathbf{v}^{(t-1)}\right)_{V\setminus C_1}$ denotes the part of vector $A^{\text{out}}D^{-1}\mathbf{v}^{(t-1)}$ supported on $V \setminus C_1$. Clearly:

$$\left\|\left(A^{\text{out}}D^{-1}\mathbf{v}^{(t-1)}\right)_{V\setminus C_1}\right\|_1 \leq \left\|A^{\text{out}}D^{-1}\mathbf{v}^{(t-1)}\right\|_1$$

and so:

$$\|\mathbf{v}^{(t)}_{V\setminus C_1}\|_1 \leq \|A^{\text{in}}D^{-1}\mathbf{v}^{(t-1)}_{V\setminus C_1}\|_1 + \|A^{\text{out}}D^{-1}\mathbf{v}^{(t-1)}\|_1 \leq \|A^{\text{in}}D^{-1}\|_1\|\mathbf{v}^{(t-1)}_{V\setminus C_1}\|_1 + \|A^{\text{out}}D^{-1}\|_1\|\mathbf{v}^{(t-1)}\|_1$$

Moreover: $\|A^{\text{in}}D^{-1}\|_1 = \max_j \sum_i \frac{A^{\text{in}}_{ij}}{d_j} = \max_j \frac{d^{\text{in}}_j}{d_j} \leq 1$ and similarly $\|A^{\text{out}}D^{-1}\|_1 = \max_j \frac{d^{\text{out}}_i}{d_i} \leq \max_j \frac{d^{\text{out}}_i}{d^{\text{in}}_i} \leq r_{\max}$. Thus:

$$\|\mathbf{v}^{(t)}_{V\setminus C_1}\|_1 \leq 1\|\mathbf{v}^{(t-1)}_{V\setminus C_1}\|_1 + r_{\max}\|\mathbf{v}^{(t-1)}\|_1.$$

Because $\|\mathbf{v}^{(0)}_{V\setminus C_1}\|_1 = 0$, $\|\mathbf{v}^{(0)}\|_1 = 1$ and $\|\mathbf{v}^{(t)}_{C_1}\|_1 \leq \|\mathbf{v}^{(0)}_{C_1}\|_1$ for any $t \geq 0$ we obtain:

$$\sum_{j\in W} v_j^{(t)} \leq \|\mathbf{v}^{(t)}_{V\setminus C_1}\|_1 \leq t r_{\max} \tag{2.8}$$

Now let us put this all together. Returning to (2.5) with (2.7) and (2.8) in hand:

$$q^t_{\min}\left(\frac{\text{vol}(U)}{\text{vol}(G_{C_1})} - (1-\lambda_2)^t\sqrt{\frac{\text{vol}(U)}{\text{vol}(\Gamma)}}\right) \leq t r_{\max} \tag{2.9}$$

We now use the fact that $G \sim \text{SSBM}(n,k,p,q)$ as well as the assumptions on the parameters p, q and k and on the seed set Γ. Note that the following claims all hold almost surely. In particular $1-\lambda_2 \leq \sqrt{12/\omega}$ from Theorem 2.2.3 and $r_{\max} = o(1)$ from Corollary 2.2.6 while:

$$q_{\min} = \min_i \frac{d^{\text{in}}_i}{d^{\text{in}}_i + d^{\text{out}}_i} = \min_i \frac{1}{1 + d^{\text{out}}_i/d^{\text{in}}_i} \geq \frac{1}{1+r_{\max}} = 1 - o(1)$$

where the final equality also follows from Corollary 2.2.6. Additionally, as $n_0 = \frac{n}{k}$,

$$\text{vol}(U) \geq d_{\min}|U| \geq d_{\min}^{\text{in}}|U| = (1-o(1))\omega\left(\log(n) - \log(k)\right)|U| = (1-o(1))\omega\log(n)un_1 \tag{2.10}$$

where the second equality follows from Corollary 2.2.5 and the final equality comes from our definition that $|U| = un_1$. Similarly $\text{vol}(U) \leq (1+o(1))\omega\log(n)un_1$ and $\text{vol}(G_{C_1}) \leq (1+o(1))\omega\log(n)n_1$ while:

$$\text{vol}(\Gamma) \geq (1-o(1))\omega\log(n)|\Gamma| = (1-o(1))\omega\log(n)\left(\frac{gn_1}{\omega^{t-1}}\right) = \frac{(1-o(1))\log(n)n_1}{\omega^{t-2}}$$

Substituting this all into (2.9):

$$(1-o(t))\left(\frac{(1-o(1))\omega\log(n)un_1}{(1+o(1))\omega\log(n)n_1} - \sqrt{\frac{12^t}{\omega^t}}\sqrt{\frac{(1+o(1))\omega\log(n)un_1}{(1-o(1))\log(n)n_1/\omega^{t-2}}}\right) \leq o(t)$$

whence:

$$u \leq o(t) + o(1) + \frac{\sqrt{12^t}}{\omega^{1/2}}\sqrt{u}$$

as long as t is constant with respect to n_1, $o(t) = o(1)$ and $\sqrt{12^t}/\omega^{1/2} = o(1)$ too, thus this inequality can hold only if $u = o(1)$, which proves the theorem. □

Remark 2.3.3. The proof of Theorem 2.3.2 illustrates the tension between the size of the seed set, Γ, and the depth of the random walk, t. In practice we have found that taking t between 3 and 10 works well, even when $g \approx 0.01$ (see §2.6).

2.4 Extracting C_1 from Ω

As outlined in §2.1, our initial goal was to analyze the solution to:

$$\underset{\mathbf{x}\in\mathbb{R}^{|\Omega|}}{\text{argmin}}\left\{\|L_\Omega\mathbf{x} - \mathbf{y}\|_2 : \quad \|\mathbf{x}\|_0 \leq |\Omega| - n_1\right\} \tag{2.11}$$

by relating it to the solution of

$$\underset{\mathbf{x}\in\mathbb{R}^{|\Omega|}}{\text{argmin}}\left\{\|L_\Omega^{\text{in}}\mathbf{x} - \mathbf{y}^{\text{in}}\|_2 : \quad \|\mathbf{x}\|_0 \leq |\Omega| - n_1\right\} \tag{2.12}$$

under the assumption that $C_1 \subset \Omega$. However Theorem 2.3.2 only guarantees that Ω contains a fraction $1 - o(1)$ of C_1, thus we are forced to change our approach slightly. In section 2.4.1 we show that if the call to `RandomWalkThresholding` in step 1 of `SingleClusterPursuit` (Algorithm 6) returns an ϵ-good superset for C_1, then Steps 2–4 find a $C_1^{\#}$ satisfying $|C_1 \triangle C_1^{\#}|/|C_1|/n_1 = o(1)$. Our analysis applies to any probabilistic model of graphs with clusters, $\mathcal{G}_n$, such that the Laplacians of $G \sim \mathcal{G}_n$ satisfy an RIP condition, which we describe shortly. In §2.4.2 we verify that the symmetric stochastic block model satisfies these assumptions, almost surely. From here it is a short step to showing that `SingleClusterPursuit` solves the almost exact recovery problem, which we present as Theorem 2.4.12.

2.4.1 Extraction guarantees given the RIP for L

Let L be the Laplacian of a graph[2] with clusters, G, and let L^{in} be as in §2.1. Let Ω be an ϵ-good subset for C_1, such as those found by `RandomWalkThresholding`. Suppose further that L and L^{in} satisfy the following three properties:

1. For any $\gamma \in (0, 1)$ we have that $\delta_{\gamma n_1}(L^{\text{in}}_\Omega) \leq \gamma + o(1)$.

2. Letting $M := L - L^{\text{in}}$, we have that $\|M_\Omega\|_2 = o(1)$

3. There exist constants κ and C such that $\sigma_\kappa(L^{\text{in}}_\Omega) \geq C > 0$

Recall that by $\sigma_\kappa(L^{\text{in}}_\Omega)$ we mean the κ smallest singular value of L^{in}_Ω.

Remark 2.4.1. We are abusing notation slightly by writing $\delta_{\gamma n_1}$. Technically we mean $\delta_{\lfloor \gamma n_1 \rfloor}$ as the quantity γn_1 might not be an integer. We maintain the convention that whenever a non-integral quantity is used where an integer is required, the floor of that quantity is intended.

[2]In order to use the asymptotic notation $o(\cdot)$ and $O(\cdot)$, we should assume rather that L_n are the Laplacians of some family of graphs $\{G_n\}_{n=1}^{\infty}$. In order to reduce the notational burden, we shall carry this assumption implicitly.

We note that $\|L\|_2 \leq 2 = O(1)$ for the Laplacian of any graph, thus property 2 is not unreasonable. As mentioned above, we are no longer assuming that $C_1 \subset \Omega$, hence the best that we can hope for is that if $\mathbf{x}^\#$ is the solution to (2.11) then $\text{supp}(\mathbf{x}^\#) \approx \Omega \setminus C_1 \cap \Omega$. To prove this, we shall regard problem (2.11) as a perturbation of a sparse recovery problem for which the solution is $\mathbf{1}_{\Omega \setminus C_1 \cap \Omega}$ and then appeal to results in totally perturbed compressive sensing outlined in §1.4.3. This means revising our original goal, as $\mathbf{1}_{\Omega \setminus C_1 \cap \Omega}$ is not the solution to Problem (2.12).

Lemma 2.4.2. *Let Ω be an ϵ-good superset for C_1. Then $\mathbf{1}_{\Omega \setminus C_1 \cap \Omega}$ is the unique solution to:*

$$\underset{\mathbf{x} \in \mathbb{R}^{|\Omega|}}{\text{argmin}} \left\{ \| L^{in}_{\Omega} \mathbf{x} - \left(\mathbf{y}^{in} + \mathbf{z}^{in} \right) \|_2 \text{ subject to: } \|\mathbf{x}\|_0 \leq 1.1 \epsilon n_1 \right\} \tag{2.13}$$

for n_1 large enough, where $\mathbf{z}^{in} := L^{in} \mathbf{1}_{C_1 \setminus \Omega \cap C_1}$.

Proof. The proof is essentially the same as the proof of Theorem 2.1.2. As in that proof, we observe that $\mathbf{1}_{\Omega \setminus C_1 \cap \Omega}$ is feasible for problem (2.13) because $\|\mathbf{1}_{\Omega \setminus C_1 \cap \Omega}\|_0 = |\Omega| - |C_1 \cap \Omega| = (1+\epsilon)n_1 - (1-o(1))n_1 = (\epsilon + o(1))n_1$ and by taking n_1 to be large enough we may guarantee that the $o(1)$ term is smaller than 0.1ϵ (the choice of 1.1ϵ is essentially arbitrary. We could pick any multiple of ϵ greater than 1 here). Next we observe that:

$$L^{\text{in}}_{\Omega} \mathbf{1}_{\Omega \setminus C_1 \cap \Omega} - \left(\mathbf{y}^{\text{in}} + \mathbf{z}^{\text{in}} \right) = L^{\text{in}} \mathbf{1}_{\Omega \setminus C_1 \cap \Omega} - L^{\text{in}} \mathbf{1}_{\Omega} - L^{\text{in}} \mathbf{1}_{C_1 \setminus \Omega \cap C_1} = -L^{\text{in}} \left(\mathbf{1}_{C_1} \right) = \mathbf{0}$$

Finally, suppose another $\mathbf{u}$ also satisfies $L^{\text{in}}_{\Omega} \mathbf{u} - \left(\mathbf{y}^{\text{in}} + \mathbf{z}^{\text{in}} \right) = \mathbf{0}$. Then:

$$\mathbf{0} = L^{\text{in}}_{\Omega} \mathbf{u} - L^{\text{in}} \mathbf{1}_{\Omega} - L^{\text{in}} \mathbf{1}_{C_1 \setminus \Omega \cap C_1} = L^{\text{in}} \left(\mathbf{u} - \mathbf{1}_{\Omega} - \mathbf{1}_{C_1 \setminus \Omega \cap C_1} \right) = L^{\text{in}} \left(\mathbf{u} - \mathbf{1}_{\Omega \cup C_1} \right)$$

Thus $\mathbf{u} - \mathbf{1}_{\Omega \cup C_1} \in \ker(L^{\text{in}})$ and $\text{supp}(\mathbf{u} - \mathbf{1}_{\Omega \cup C_1}) \subset \Omega \cup C_1$. Again by Theorem 3.3.1, we have that $\mathbf{u} - \mathbf{1}_{\Omega \cup C_1} = \sum_{a=1}^{k} \alpha_a \mathbf{1}_{C_a}$ and so $\mathbf{u} = \mathbf{1}_{\Omega \cup C_1} + \sum_{a=1}^{k} \alpha_a \mathbf{1}_{C_a}$. By assumption C_1 is the only cluster contained in $\Omega \cup C_1$ so $\alpha_a = 0$ for $a \geq 2$ thus $\mathbf{u} = \mathbf{1}_{\Omega \cup C_1} + \alpha_1 \mathbf{1}_{C_1}$ As for Theorem 2.1.2, because $\mathbf{u} \neq \mathbf{1}_{\Omega \setminus C_1 \cap \Omega}$ we cannot have $\alpha_1 = -1$. But then $\text{supp}(\mathbf{u}) = \Omega \cup C_1$ and so $\mathbf{u}$ is not feasible as $\|\mathbf{u}\|_0 \geq |\Omega| = (1+\epsilon)n_1 > 1.1\epsilon n_1$ □

We can now profitably regard problem (2.11) as a perturbation of problem (2.13). Let us quantify this. Define $\mathbf{e} = \mathbf{y} - \left(\mathbf{y}^{\text{in}} + \mathbf{z}^{\text{in}}\right) = \left(\mathbf{y} - \mathbf{y}^{\text{in}}\right) - \mathbf{z}^{\text{in}}$ and recall that $M = L - L^{\text{in}}$. Recall from §1.4.3, and in particular Theorem 1.4.6, that the two key parameters in perturbed compressed sensing are

$$\epsilon_{\mathbf{y}} = \frac{\|\mathbf{e}\|_2}{\|\mathbf{y}^{\text{in}} + \mathbf{z}^{\text{in}}\|_2} \quad \text{and} \quad \epsilon_L^s = \frac{\|M_\Omega\|_2^{(s)}}{\|L_\Omega^{\text{in}}\|_2^{(s)}}$$

Theorem 2.4.3. *Assume that L satisfies the three properties at the beginning of this section for an ϵ-good superset Ω. Then $\epsilon_{\mathbf{y}} = o(1)$ and $\epsilon_L^{\gamma n_1} = o(1)$ for any $\gamma \in (0,1)$.*

Proof. From the proof of Lemma 2.4.2 we have $\mathbf{y}^{\text{in}} + \mathbf{z}^{\text{in}} = L^{\text{in}}\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)}$ and $\|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)}\|_0 = |\Omega \setminus (C_1 \cap \Omega)| = (\epsilon + o(1))n_1 \leq 1.1\epsilon n_1$ for large enough n_1. Using the assumption on the restricted isometry constant of L^{in} we have:

$$\begin{aligned}\|\mathbf{y}^{\text{in}} + \mathbf{z}^{\text{in}}\|_2^2 = \|L^{\text{in}}\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)}\|_2^2 &\geq \left(1 - \delta_{1.1\epsilon n_1}(L_\Omega^{\text{in}})\right)\|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)}\|_2^2 \\ &\geq (1.1\epsilon - o(1))\,|\Omega \setminus (C_1 \cap \Omega)| \geq (1.1\epsilon - o(1))\,\epsilon n_1\end{aligned}$$

Thus $\|\mathbf{y}^{\text{in}} + \mathbf{z}^{\text{in}}\|_2 \geq \sqrt{(\epsilon(1.1\epsilon - o(1))}\sqrt{n_1}$. On the other hand, $\|\mathbf{e}\|_2 \leq \|\mathbf{y} - \mathbf{y}^{\text{in}}\|_2 + \|\mathbf{z}^{\text{in}}\|_2$ and:

$$\|\mathbf{y} - \mathbf{y}^{\text{in}}\|_2 = \|L_\Omega\mathbf{1}_\Omega - L_\Omega^{\text{in}}\mathbf{1}_\Omega\|_2 = \|M_\Omega\mathbf{1}_\Omega\|_2 \leq \|M_\Omega\|_2\|\mathbf{1}_\Omega\|_2 \leq o(1)\sqrt{(1+\epsilon)n_1}$$

while:

$$\|\mathbf{z}^{\text{in}}\|_2 = \|L^{\text{in}}\mathbf{1}_{C_1\setminus\Omega\cap C_1}\|_2 \leq \|L^{\text{in}}\|_2\|\mathbf{1}_{C_1\setminus\Omega\cap C_1}\|_2 \leq 2\sqrt{|C_1 \setminus \Omega \cap C_1|} = 2o(1)\sqrt{n_1}$$

Thus:

$$\epsilon_{\mathbf{y}} = \frac{\|\mathbf{e}\|_2}{\|\mathbf{y}^{\text{in}} + \mathbf{z}^{\text{in}}\|_2} \leq \frac{o(1)\left(\sqrt{(1+\epsilon)} + 2\right)\sqrt{n_1}}{\sqrt{(\epsilon(1.1\epsilon - o(1))}\sqrt{n_1}} = o(1)$$

as ϵ is a constant, *i.e.* independent of n_1. The bound on $\epsilon_L^{\gamma n_1}$ is easier. By Lemma 1.4.7 and Property 3:

$$\|L_\Omega^{\text{in}}\|_2^{(\gamma n_1)} \geq \sigma_{\gamma n_1 - 1}\left(L_\Omega^{\text{in}}\right) \geq \sigma_\kappa\left(L_\Omega^{\text{in}}\right) \geq C > 0$$

for $n_1 \geq (\kappa+1)/\gamma$. By Property 2 and Lemma 1.4.7 $\|M_\Omega\|_2^{(\gamma n_1)} \leq \|M_\Omega\|_2 = o(1)$. It follows that:

$$\epsilon_L^{\gamma n_1} = \frac{\|M_\Omega\|_2^{(\gamma n_1)}}{\|L_\Omega^{\text{in}}\|_2^{(\gamma n_1)}} = \frac{o(1)}{C} = o(1)$$

□

We also bound the restricted isometry constant of L:

Lemma 2.4.4. *Assuming Properties 1–3 from the beginning of this section, $\delta_{\gamma n_1}(L_\Omega) \leq \gamma + o(1)$ for any $\gamma \in (0,1)$.*

Proof. From Theorem 1.4.8 we have that:

$$\begin{aligned}\delta_{\gamma n_1}(L_\Omega) &\leq (1+\delta_{\gamma n_1}(L^{\text{in}}))(1+\epsilon_\Phi^{\gamma n_1})^2 - 1 \\ &\leq (1+\gamma+o(1))\,(1+o(1))^2 - 1 \\ &= 1+\gamma+o(1)-1 = \gamma+o(1)\end{aligned}$$

where the value of $\delta_{\gamma n_1}(L^{\text{in}})$ comes from Property 2 while the value of $\epsilon_\Phi^{\gamma n_1}$ comes from Theorem 2.4.3. □

Finally, there are two auxiliary quantities, namely ρ and τ, which appear in the statement of Theorem 1.4.6. For any s, given a bound on $\delta_{3s}(L)$, computing bounds for ρ and τ is purely computational, and hence we omit the details

Lemma 2.4.5. *Suppose that $\delta_{3s}(L) \leq 0.45$, and let ρ and τ be as defined in Theorem 1.4.6. Then $\rho \leq 0.8751$ and $\tau \leq 55.8490$.*

Proof. Follows from direct computation. □

We are now able to prove the theorem advertised at the beginning of this section.

Theorem 2.4.6. *Let $C_1^\#$ denotes the output of* `SingleClusterPursuit`*, given parameters $\epsilon \in (0, 0.13]$, $\hat{n}_1 = n_1$, $R = 0.5$ and Γ such that the superset Ω found in Step 1 of* `SingleClusterPursuit` *is ϵ-good. Suppose that L and L^{in} satisfy the three Properties listed at the beginning of this section. Then $|C_1 \triangle C_1^\#|/n_1 = o(1)$*

Proof. For notational convenience, let $s = \lfloor 1.1\epsilon n_1 \rfloor$. Recall that $\mathbf{x}^{(m)}$ is the solution obtained by $m = O(\log(n))$ iterations of `SubspacePursuit` on Problem 2.4, which we are regarding as a perturbation of Problem 2.13. Because $\epsilon \leq 0.13$ we have that $s \leq 0.145 n_1$ and so by Lemma 2.4.4 $\delta_s(L_\Omega) \leq 0.145 n_1 + o(1) < 0.15$ for n_1 large enough. Similarly $\delta_{3s}(L_\Omega) \leq 0.45$, again for n_1 large enough. It follows from Lemma 2.4.5 that $\rho \leq 0.8751$ and $\tau \leq 55.8490$. We now appeal to Theorem 1.4.6 to obtain:

$$\frac{\|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)} - \mathbf{x}^{(m)}\|_2}{\|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)}\|_2} \leq \rho^m + \tau\frac{\sqrt{1+\delta_s}}{1-\epsilon_\Phi^s}(\epsilon_\Phi^s + \epsilon_{\mathbf{y}})$$

The second term on the right-hand side is $o(1)$. As long as $m \geq \log_\rho(1/n) = O(\log(n))$, we obtain that $\rho^m = 1/n = o(1)$ too. Thus:

$$\frac{\|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)} - \mathbf{x}^{(m)}\|_2}{\|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)}\|_2} \leq o(1) \Rightarrow \|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)} - \mathbf{x}^{(m)}\|_2 \leq o\left(\|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)}\|_2\right)$$

as established in the proof of Lemma 2.4.2, $|\Omega \setminus (C_1 \cap \Omega)| = (\epsilon + o(1))n_1$ and so $\|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)}\|_2 = \sqrt{(\epsilon + o(1))n_1}$. It follows that $\|\mathbf{1}_{\Omega\setminus(C_1\cap\Omega)} - \mathbf{x}^{(m)}\|_2 = o(\sqrt{n_1})$.

By definition (see step (4) of Algorithm 6) $W^\# = \{i : \ x_i^{(m)} > 0.5\}$. It will follow from Lemma 2.4.7 that $|W^\# \triangle (\Omega \setminus (\Omega \cap C_1))| = o(n_1)$. As $C_1^\# = \Omega \setminus W^\#$, it will follow that $|C_1^\# \triangle \Omega \cap C_1| = o(n_1)$. Accounting for $U := C_1 \setminus \Omega \cap C_1$, we have that

$$|C^\# \triangle C_1| = |C^\# \triangle (\Omega \cap C_1)| + |U| = o(n_1) + o(n_1) = o(n_1)$$

whence it follows that $|C^\# \triangle C_1|/n_1 = o(1)$ as desired. □

Lemma 2.4.7. *Let $T \subset [n]$ and $\mathbf{v} \in \mathbb{R}^n$. Define $W_1 = \{i : \ v_i > 0.5\} \subset [n]$. If $\|\mathbf{1}_T - \mathbf{v}\|_2 \leq D$ then $|T \triangle W_1| \leq 4D^2$.*

Proof. Define $W_2 = [n] \setminus W_1$ and write $\mathbf{v} = \mathbf{v}_{W_1} + \mathbf{v}_{W_2}$ where $\mathbf{v}_{W_i}$ denotes the part of $\mathbf{v}$ supported on W_i. Write:

$$\|\mathbf{1}_T - \mathbf{v}\|^2 = \|\mathbf{1}_{T\cap W_1} - (\mathbf{v}_{W_1})_{T\cap W_1}\|^2 + \|(\mathbf{v}_{W_1})_{W_1\setminus T}\|^2 + \|\mathbf{1}_{T\setminus W_1} - \mathbf{v}_{W_2}\|^2$$

Now $\|(\mathbf{v}_{W_1})_{W_1 \setminus T}\|^2 \geq 0.25|W_1 \setminus T|$ and $\|\mathbf{1}_{T \setminus W_1} - \mathbf{v}_{W_2}\|^2 \geq 0.25|T \setminus W_1|$. It follows that:

$$0.25|T \triangle W_1| = 0.25\left(|T \setminus W_1| + |W_1 \setminus T|\right) \leq \|(\mathbf{v}_{W_1})_{W_1 \setminus T}\|^2 + \|\mathbf{1}_{T \setminus W_1} - \mathbf{v}_{W_2}\|^2 \leq \|\mathbf{1}_T - \mathbf{v}\|^2$$

Hence $|T \triangle W_1| \leq 4D^2$. □

2.4.2 RIP for the stochastic block model

In this section we establish that if L is the Laplacian of some $G \sim \mathrm{SSBM}(n, k, p, q)$ with our standing assumptions on p, q, and k, then L satisfies the three assumptions at the beginning of §2.4.1 almost surely.

Theorem 2.4.8. *Suppose that $G \sim SSBM(n, k, p, q)$ with $p = \omega \log(n_0)/n_0$ with $\omega \to \infty$, $q = b \log(n)/n$ and $k = \mathcal{O}(1)$. Then $\|M\|_2 \leq o(1)$.*

Proof. Letting δ_{ij} denote the Kronecker delta symbol, observe that

$$L_{ij} := \delta_{ij} - \frac{1}{d_i} A_{ij} = \delta_{ij} - \frac{1}{d_i^{\text{in}} + d_i^{\text{out}}} \left(A_{ij}^{\text{in}} + A_{ij}^{\text{out}}\right).$$

Earlier we defined $r_i = d_i^{\text{out}}/d_i^{\text{in}}$. We now use the following easily verifiable one dimensional version of the Woodbury formula:

$$\frac{1}{d_i^{\text{in}} + d_i^{\text{out}}} = \frac{1}{d_i^{\text{in}}} - \frac{1}{d_i^{\text{in}}} \left(\frac{r_i}{r_i + 1}\right)$$

Thus:

$$\begin{aligned}
L_{ij} &= \delta_{ij} - \left(\frac{1}{d_i^{\text{in}}} - \frac{1}{d_i^{\text{in}}} \left(\frac{r_i}{r_i + 1}\right)\right) \left(A_{ij}^{\text{in}} + A_{ij}^{\text{out}}\right) \\
&= \left(\delta_{ij} - \frac{1}{d_i^{\text{in}}} A_{ij}^{\text{in}}\right) - \frac{1}{d_i^{\text{in}}} A_{ij}^{\text{out}} + \frac{1}{d_i^{\text{in}}} \left(\frac{r_i}{r_i + 1}\right) \left(A_{ij}^{\text{in}} + A_{ij}^{\text{out}}\right) \\
&= L_{ij}^{\text{in}} - \frac{1}{d_i^{\text{in}}} \left(1 - \frac{r_i}{r_i + 1}\right) A_{ij}^{\text{out}} + \frac{1}{d_i^{\text{in}}} \left(\frac{r_i}{r_i + 1}\right) A_{ij}^{\text{in}} \\
&= L_{ij}^{\text{in}} - \frac{1}{d_i^{\text{in}}} \left(\frac{1}{r_i + 1}\right) A_{ij}^{\text{out}} + \frac{1}{d_i^{\text{in}}} \left(\frac{r_i}{r_i + 1}\right) A_{ij}^{\text{in}}.
\end{aligned}$$

That is, $M_{ij} = -\frac{1}{d_i^{\text{in}}}\left(\frac{1}{r_i+1}\right)A_{ij}^{\text{out}} + \frac{1}{d_i^{\text{in}}}\left(\frac{r_i}{r_i+1}\right)A_{ij}^{\text{in}}$. To bound the spectral norm we use Gershgorin's disks, noting that $M_{ii} = 0$ for all i:

$$\begin{aligned}
\|M\|_2 &= \max_i\{|\mu_i| : \ \mu_i \text{ eigenvalue of } M\} \leq \max_i \sum_j |M_{ij}| \\
&= \max_i \frac{1}{d_i^{\text{in}}}\left(\frac{1}{r_i+1}\right)\sum_j A_{ij}^{\text{out}} + \frac{1}{d_i^{\text{in}}}\left(\frac{r_i}{r_i+1}\right)\sum_j A_{ij}^{\text{in}} \\
&= \max_i \left\{\frac{1}{d_i^{\text{in}}}\left(\frac{1}{r_i+1}\right)(d_i^{\text{out}}) + \frac{1}{d_i^{\text{in}}}\left(\frac{r_i}{r_i+1}\right)(d_i^{\text{in}})\right\} \\
&= \max_i \left\{\left(\frac{r_i}{r_i+1}\right) + \left(\frac{r_i}{r_i+1}\right)\right\} \leq 2r_{\max} = o(1) \text{ a.s. by Corollary 2.2.6}
\end{aligned}$$

□

Lemma 2.4.9. *Let G be any connected graph on n_0 vertices, and let $s < n_0$. Let $\lambda_i := \lambda_i(L)$ denote the i-th smallest eigenvalue of L. Then:*

$$\delta_s(L) \leq \max\{1 - \lambda_2^2\left(\frac{d_{\min}}{d_{\max}} - \frac{d_{\max}}{d_{\min}}\frac{s}{n_0}\right), 1 - \lambda_{\max}^2\}.$$

Proof. Recall that the s-Restricted Isometry Constant $\delta_s(L)$ is the smallest δ such that, for any $\mathbf{v}$ with $\|\mathbf{v}\|_0 \leq s$ and $\|\mathbf{v}\|_2 = 1$:

$$(1-\delta) \leq \|L\mathbf{v}\|_2^2 \leq (1+\delta).$$

We shall prove the theorem by showing that, for any such $\mathbf{v}$, $\|L\mathbf{v}\|_2 \leq \lambda_{\max}$ and $\|L\mathbf{v}\|_2 \geq \lambda_2^2\left(\frac{d_{\min}}{d_{\max}} - \frac{d_{\max}}{d_{\min}}\frac{s}{n_0}\right)$. The first bound is straightforward:

$$\|L\mathbf{v}\|_2 \leq \|L\|_2\|\mathbf{v}\|_2 = \lambda_{\max}(1) = \lambda_{\max}$$

The second bound requires some work. Recall that $L = I - D^{-1}A$. This matrix is not symmetric, but $L^{\text{sym}} = I - D^{-1/2}AD^{-1/2}$ is. By Lemma 1.2.2 L and L^{sym} have the same eigenvalues. Let $\mathbf{w}_1, \ldots, \mathbf{w}_{n_0}$ be an orthonormal eigenbasis for L^{sym}. These eigenvectors are well studied (see, for example, [24]) and in particular $\mathbf{w}_1 = \frac{1}{\sqrt{\text{vol}(G)}}D^{1/2}\mathbf{1}$ where $\mathbf{1}$ is the all-ones vector. Observe that:

$$L\mathbf{v} = D^{-1/2}\left(D^{1/2}LD^{-1/2}\right)D^{1/2}\mathbf{v} = D^{-1/2}L^{\text{sym}}D^{1/2}\mathbf{v} = D^{-1/2}L^{\text{sym}}\mathbf{z},$$

where $\mathbf{z} := D^{1/2}\mathbf{v}$. It follows that:

$$\|L\mathbf{v}\|_2 = \|D^{-1/2}L^{\text{sym}}\mathbf{z}\|_2 \geq \frac{1}{\sqrt{d_{\max}}}\|L^{\text{sym}}\mathbf{z}\|_2. \tag{2.14}$$

Express $\mathbf{z}$ in terms of the orthonormal basis $\{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$, namely $\mathbf{z} = \sum_{i=1}^{n_0} \alpha_i \mathbf{w}_i$. Then:

$$\|L^{\text{sym}}\mathbf{z}\|_2^2 = \|\sum_{i=1}^{n_0} \alpha_i\lambda_i\mathbf{w}_i\|_2^2 = \|\sum_{i=2}^{n_0} \alpha_i\lambda_i\mathbf{w}_i\|_2^2 \geq \lambda_2^2\left(\sum_{i=2}^{n_0}\alpha_i^2\right)$$

and $\sum_{i=2}^{n_0}\alpha_i^2 = \|\mathbf{z}\|_2^2 - \alpha_1^2$. We now bound $\|\mathbf{z}\|_2$ and α_1.

$$\|\mathbf{z}\|_2^2 = \|D^{1/2}\mathbf{v}\|_2^2 \geq \left(\sqrt{d_{\min}}\right)^2\|\mathbf{v}\|_2^2 = d_{\min}$$

while:

$$\alpha_1 = \langle\mathbf{z}, \mathbf{w}_1\rangle = \langle D^{1/2}\mathbf{v}, \frac{1}{\sqrt{\text{vol}(G)}}D^{1/2}\mathbf{1}\rangle = \frac{1}{\sqrt{\text{vol}(G)}}\langle\mathbf{v}, D\mathbf{1}\rangle \leq \frac{d_{\max}}{\sqrt{\text{vol}(G)}}\langle\mathbf{v}, \mathbf{1}\rangle.$$

We now use the assumptions on $\mathbf{v}$. Specifically $\langle\mathbf{v}, \mathbf{1}\rangle \leq \|\mathbf{v}\|_1 \leq \sqrt{s}\|\mathbf{v}\|_2 = \sqrt{s}$ and so

$$\alpha_1 \leq d_{\max}\frac{\sqrt{s}}{\sqrt{\text{vol}(G)}} \leq d_{\max}\frac{\sqrt{s}}{\sqrt{d_{\min}n_0}} = \frac{d_{\max}}{\sqrt{d_{\min}}}\frac{\sqrt{s}}{\sqrt{n_0}}.$$

Returning to equation (2.14):

$$\|L\mathbf{v}\|_2^2 \geq \frac{1}{d_{\max}}\|L^{\text{sym}}\mathbf{z}\|_2^2 \geq \frac{1}{d_{\max}}\lambda_2^2\left(d_{\min} - \frac{d_{\max}^2}{d_{\min}}\frac{s}{n_0}\right) = \lambda_2^2\left(\frac{d_{\min}}{d_{\max}} - \frac{d_{\max}}{d_{\min}}\frac{s}{n_0}\right).$$

□

Lemma 2.4.10. *Suppose that $G \sim ER(n_0, p)$ with $p = \omega\ln(n_0)/n_0$ for some $\omega \to \infty$. Then $\delta_s(L) \leq s/n_0 + o(1)$ a.s.*

Proof. This is a simple consequence of Lemma 2.4.9. If G is as in the hypothesis then $d_{\min} = (1 - o(1))\, n_0p$ and $d_{\max} = (1 + o(1))\, n_0p$ a.s. by Theorem 2.2.2. Moreover $\lambda_2 \geq 1 - o(1)$ and $\lambda_{n_0} \leq 1 + o(1)$ a.s. by Theorem 2.2.3. Hence:

$$\begin{aligned}
\lambda_2^2\left(\frac{d_{\min}}{d_{\max}} - \frac{d_{\max}}{d_{\min}}\frac{s}{n_0}\right) &\geq (1 - o(1))^2\left(\frac{(1 - o(1))\, n_0p}{(1 + o(1))\, n_0p} - \frac{(1 + o(1))\, n_0p}{(1 - o(1))\, n_0p}\frac{s}{n_0}\right) \\
&\geq (1 - o(1))\left(\frac{1 - o(1)}{1 + o(1)} - \frac{1 + o(1)}{1 - o(1)}\frac{s}{n_0}\right) \\
&= (1 - o(1))\left(1 - o(1) - (1 + o(1))\,\frac{s}{n_0}\right) \\
&= 1\left(1 - \frac{s}{n_0} - o(1)\right) - o(1) = 1 - \frac{s}{n_0} - o(1) \quad \text{a.s.}
\end{aligned}$$

Hence by Lemma 2.4.9, we have that

$$\delta_t(L) \leq \max\left\{1 - \left(1 - \frac{s}{n_0} - o(1)\right), o(1)\right\} = \frac{s}{n_0} + o(1) \quad \text{a.s.}$$

□

Theorem 2.4.11. *Suppose that $G \sim SSBM(n, k, p, q)$ with $k = O(1)$ and $p = \omega \log(n_0)/n_0$. Then for any $\gamma \in (0, 1)$, we have that $\delta_{\gamma n_0}(L^{in}) \leq \gamma + o(1)$ a.s.*

Proof. L^{in} is the Laplacian of G^{in}, which is a disjoint union of k Erdős - Rènyi graphs. It follows that L^{in} is block diagonal, with blocks $L_{G_{C_1}}, \ldots, L_{G_{C_k}}$, where each $L_{G_{C_a}}$ is the Laplacian of an i.i.d graph $G_a \sim \text{ER}(n_0, p)$ with $p = \omega \log(n_0)/n_0$.

By Lemma 2.4.10 $\delta_{\gamma n_0}(L_{G_{C_a}}) \leq \gamma + o(1)$ a.s. That is, $\mathbb{P}\left[\delta_{\gamma n_0}(L_{G_{C_a}}) \leq \gamma + o(1)\right] = 1 - o(1)$. For a block diagonal matrix such as L^{in}, one can easily check that for any $s \in [n_0]$ we have $\delta_s(L^{\text{in}}) = \max_a \delta_s(L_{G_{C_a}})$. Finally, observe that:

$$\mathbb{P}\left[\max_a \delta_{\gamma n_0}(L_{G_{C_a}}) \leq \gamma + o(1)\right] = \prod_{a=1}^{k} \mathbb{P}\left[\delta_s(L_{G_{C_a}}) \leq \gamma + o(1)\right] = (1 - o(1))^k$$

as the G_{C_a} are i.i.d. Because $k = O(1)$, it follows that $(1 - o(1))^k = 1 - o(k) = 1 - o(1)$, and the lemma follows.

□

2.4.3 Almost Exact Recovery

Finally, we may conclude that `SingleClusterPursuit` solves the almost exact cluster extraction problem, presented as Definition 1.3.2.

Theorem 2.4.12. `SingleClusterPursuit` *with parameters $\epsilon \in (0, 0.13]$, $\hat{n}_1 = n_1$, $R = 0.5$ and Γ with $|\Gamma| = gn_1/\omega^{t-1}$ solves the almost exact cluster extraction problem for $SSBM(n, k, p, q)$ with $p = \omega \log(n_0)/n_0$, $q = b \log(n)/n$ and $k = O(1)$ where $\omega \to \infty$.*

Proof. By Theorem 2.3.2 step 1 of `SingleClusterPursuit` returns an ϵ-good subset given these parameters, hence we may achieve the theorem using the machinery of §2.4.1, in particular Theorem 2.4.6, by showing that the three properties listed at the beginning of §2.4.1 are satisfied, almost surely.

By Theorem 2.4.11 L_Ω^{in} satisfy Property 1 of §2.4.1 almost surely. By Theorem 2.4.8 , $\|M_\Omega\|_2 \leq \|M\|_2 = o(1)$ hence Property 2 is satisfied almost surely. We now consider Property 3. As in Theorem 2.4.11 L^{in} is block diagonal with blocks $L_{G_{C_1}}, \dots L_{G_{C_k}}$ where each $L_{G_{C_a}}$ is the Laplacian of an i.i.d graph $G_{C_a} \sim \text{ER}(n_0, p)$ with $p = \omega \log(n_0)/n_0$. The set of eigenvalues of L^{in} is the union of the sets of eigenvalues of the $L_{G_{C_a}}$. Because $\lambda_1(L_a) = 0$ for $a = 1, \dots, k$, we know that L^{in} has the eigenvalue 0 with multiplicity k. Hence:

$$\lambda_{k+1}(L^{\text{in}}) = \min_{a=1}^{k} \lambda_2(L_a)$$

From Theorem 2.2.3 we have that $\mathbb{P}\left[\lambda_2(L_{G_{C_a}}) \geq 1 - o(1)\right] = 1 - o(1)$ for all $a \in [k]$. Because the G_{C_a}, are i.i.d:

$$\mathbb{P}\left[\lambda_{k+1}(L^{\text{in}}) \geq 1 - o(1)\right] = \mathbb{P}\left[\min_{a=1}^{k} \lambda_2(L_a) \geq 1 - o(1)\right] = \prod_{a=1}^{k} \mathbb{P}\left[\lambda_2(L_a) \geq 1 - o(1)\right] = (1 - o(1))^k$$

As in the proof of Theorem 2.4.11, we have that $(1 - o(1))^k = 1 - o(1)$. Finally, by the interlacing theorem for singular values (cf. [83]) we get that $\lambda_{k+1}(L^{\text{in}}) \leq \sigma_{k+2}\left(L_\Omega^{\text{in}}\right)$, thus taking $\kappa = k + 2$ and $C = 1/2$, for example, we get that Property 3 holds almost surely as:

$$\mathbb{P}\left[\sigma_{k+2}(L_\Omega^{\text{in}}) \geq 1 - o(1)\right] \geq \mathbb{P}\left[\lambda_{k+1}(L^{\text{in}}) \geq 1 - o(1)\right] = 1 - o(1)$$

and for large enough n the $o(1)$ term inside the brackets is less than $1/2$. □

Remark 2.4.13. 1. We have chosen to write $p = \omega \log(n_0)/n_0$ as we feel this makes the presentation slightly clearer, but of course as $n_0 = n/k$ by redefining ω this is equivalent to $p = \omega \log(n)/n$.

2. We emphasize that we place no restrictions on how slowly $\omega \to \infty$.

3. The restriction to the symmetric block model is not essential. By wading through a slightly larger sea of notation one can show that a theorem analogous to Theorem 2.4.12 holds for SBM($\mathbf{n}, P$) as long as the number of clusters, $k = O(1)$ and $p := \min_a P_{aa} = \omega \log(n)/n$ while $q := \max_{a \neq b} P_{ab} = b \log(n)/n$.

2.5 Computational Complexity

In this section we bound the number of operations that `SingleClusterPursuit` requires. We assume that Properties 1–3 at the beginning of §2.4 and we assume that A is stored as a sparse matrix with $d_{\max} := \max_i \sum_i A_{ij} \ll n$. An observation that we will use repeatedly is that multiplying a vector by such a sparse matrix (and observe that L and P are similarly sparse) takes $O(d_{\max} n)$ operations.

1. Running `RandomWalkThresholding` (step (1) of `SingleClusterPursuit`) takes $O\left((t d_{\max} + \log(n))n\right)$ operations, because:

   - Computing D and P require $O(d_{\max} n)$ operations each. Computing $\mathbf{v}^{(0)}$ requires $|\Gamma|$ operations.
   - Computing $\mathbf{v}^{(t)}$ requires t matrix-vector multiplies, each requiring $O(d_{\max} n)$ operations. So this step requires $O(t d_{\max} n)$ operations.
   - The most expensive part of the thresholding step (step (3)) is sorting the entries of $\mathbf{v}^{(t)} \in \mathbb{R}^n$, which requires $O(n \log(n))$ operations.

2. In step (2) of `SingleClusterPursuit`, we get L essentially for free, as $L = I - P^\top$ and we have already computed P. Observe that $\mathbf{y} = \sum_{i \in \Omega} \ell_i = L\mathbf{1}_\Omega$ hence this is another sparse matrix-vector multiply, costing $O(d_{\max} n)$.

3. The call to `SubspacePursuit` (see Algorithm 4) in step (3) costs m times the cost of each iteration, which we now bound. The cost of the j-iteration is dominated by the cost of solving the least squares problem:

$$\operatorname*{argmin}_{\mathbf{z}\in\mathbb{R}^n} \left\{ \|L_\Omega \mathbf{z} - \mathbf{y}\|_2 : \ \operatorname{supp}(\mathbf{x}) \subset \hat{S}^j \right\}.$$

Because of the support condition, and because $|\hat{S}^j| = 2s = 2\epsilon\hat{n}_1$, this is equivalent to the least squares problem:

$$\operatorname*{argmin}_{\mathbf{z}\in\mathbb{R}^{2s}} \left\{ \|L_{\hat{S}^j} \mathbf{z} - \mathbf{y}\|_2 \right\} \tag{2.15}$$

We recommend using an iterative method, such as conjugate gradient (in our implementation we use MATLAB's `lsqr` operation). Fortunately, as pointed out in [72], the matrix in question, $L_{\hat{S}^j}$ is extremely well conditioned. This is because $\delta_{2s}(L) \leq \delta_{3s}(L) \leq 0.45$ by assumption 1, and see the proof of Theorem 2.4.6. By [72], specifically Proposition 3.1 and the discussion of §5, this implies that the condition number is small:

$$\kappa(L_{\hat{S}^j}^\top L_{\hat{S}_j}) := \frac{\lambda_{\max}(L_{\hat{S}^j}^\top L_{\hat{S}_j})}{\lambda_{\min}(L_{\hat{S}^j}^\top L_{\hat{S}_j})} \leq \frac{1+\delta_{2s}}{1-\delta_{2s}} \leq 2.64$$

The upshot of this is that it only requires a constant number of iterations of conjugate gradient to approximate the solution to the least-squares problem 2.15 to within an acceptable tolerance. Indeed, Corollary 5.3 of [72] argues that three iterations suffices. We play it safe by performing ten iterations. The cost of each iteration of conjugate gradient is equal to (a constant times) the cost of a sparse matrix vector multiply by $L_{\hat{S}_j}$ or $L_{\hat{S}_j}^\top$, which is $O(d_{\max} n)$. Hence the total cost of step (3) of `SingleClusterPursuit` is $O(m d_{\max} n) = O(d_{\max} \log(n) n)$ because we are taking $m = O(\log(n))$.

Adding this all up, we get that the number of operations required is $O\left((t + \log(n)) d_{\max} n\right)$. Because we take t, the depth of the random walk, to be constant with respect to n, the asymptotic complexity of `SingleClusterPursuit` is $O(d_{\max} \log(n) n)$. To make this more concrete, observe that for the stochastic block model $\mathrm{SSBM}(n, k, p, q)$ with parameters as in Theorem 2.4.12 we have that $d_{\max} \leq d_{\max}^{\text{in}} + d_{\max}^{\text{out}} \leq (1 + o(1))\omega \log(n) + b \log(n) + o(1)$

by Corollaries 2.2.4 and 2.2.5. Because $\omega \to \infty$, for large enough n this certainly gives $d_{\max} \leq 2\omega \log(n)$. In the numerical experiments (§2.6.3) we take $\omega = \log(n)$, thus we get an asymptotic computational complexity of $O(\log^3(n)n)$. In particular, we note that this is faster than the `SphereComparison` algorithm of Abbe and Sandon (see [5] or §1.3.3), which has complexity $O(n^{1+o(1)})$.

2.6 Numerical Results

2.6.1 Implementation of algorithms

All algorithms considered were run in MATLAB.

`SingleClusterPursuit` The implementation of `SingleClusterPursuit` used is available as the function `SCPMainRW`. We set the parameters $\epsilon = 0.13$, $R = 0.5$ and the depth of the random walk $t = 3$. Unless otherwise indicated, $\hat{n}_1$ was set to be the true size of the cluster of interest.

`ESSC` The algorithm we refer to as `ESSC` is technically the sub-routine referred to as `Community-Search` on pg. 1863 of [89] and as `Main.Search` in the R package for `ESSC` (available at `http://jdwilson-statistics.com/publications/`). We use a MATLAB implementation of this algorithm written by the author. We compared the accuracy and run time of our MATLAB version to that of the R version, and found them to be nearly identical. We set the maximum number of iterations to 50 and the parameter $\alpha = 0.05$. As we found `ESSC` to run slowly on large data sets, we did not use it in all experiments.

`LBSA` We use the MATLAB implementation provided by the authors of [79], available at `https://github.com/PanShi2016/LBSA`. The `LBSA` algorithm actually includes six distinct methods; we use the heat kernel sampling with Lanczos method, denoted in [79] as `hkLISA`, as experimental evidence presented in the aforementioned paper suggests that this variant performs best. We also tried other methods (specifically Heat Kernel sampling with Power

method, and random walk sampling with power and Lanczos methods), but did not observe any significant difference in performance on our data sets. We set the parameter which governs the number of Lanczos iterations to take, namely k_2, to be equal to 4 as suggested in [79].

`HKGrow` We use the MATLAB implementation of this algorithm available at `https://www.cs.purdue.edu/homes/dgleich/codes/hkgrow/`. This implementation requires no input parameters.

The size of the seed set Γ given to `SSCP`, `LOSP++` and `HKGrow` is gn_1, where $g \in (0, 0.1)$ and n_1 is the true size of the cluster of interest. `ESSC` is seeded with the neighborhood of the highest degree vertex in the cluster of interest, as done in [89].

2.6.2 Measures of cluster quality

When there exists a known, ground truth cluster C, we measure the accuracy of cluster extraction using the *Jaccard Index*: $\mathrm{Jac}(C, C^{\#}) := \left|C \cap C^{\#}\right| / \left|C \cup C^{\#}\right|$. The maximum value of $\mathrm{Jac}(C, C^{\#})$ is 1, and this occurs when $C = C^{\#}$. The Jaccard index has a minimum value of 0, which is achieved when C and $C^{\#}$ are disjoint. Note that the Jaccard index also penalizes "trivial" cluster extraction algorithms, that is algorithms that return $C^{\#} \approx V$, with a low score.

2.6.3 Synthetic Data Sets 1: The Stochastic Block Model

We consider graphs drawn from three different stochastic block models. Typical adjacency matrices for graphs drawn from these three models are shown in Figure 2.1. In all cases we focus on extracting the first cluster, C_1, and in all cases we vary n_1 from 200 to 600 (n scales with n_1) and take $|\Gamma| = 0.02n_1$. We perform ten independent trials for each value of n_1.

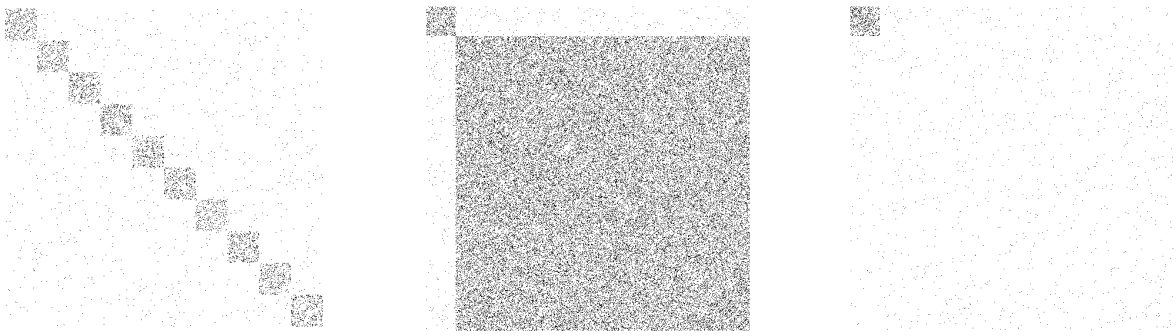

Figure 2.1: The adjacency matrices of typical graphs for each of the three benchmarks, permuted to reveal the ground truth clusters. From left to right: Experiments 1–3

Experiment 1. We consider the symmetric stochastic block model with parameters as in Theorem 2.4.12. That is, we draw $G \sim \mathrm{SSBM}(n, k, p, q)$ with $k = 10$, $n = 10n_1$, $p = 2\log^2(n)/n$ and $q = \log(n)/n$.

Experiment 2. Here, we consider graphs $G \sim \mathrm{SBM}(\mathbf{n}, P_2)$, where $\mathbf{n} = (n_1, 10n_1)$ and

$$P_1 = \begin{bmatrix} 2\log^2(n)/n & \log(n)/n \\ \log(n)/n & 2\log^2(n)/n \end{bmatrix}$$

Experiment 3. Finally, we draw graphs from $\mathrm{SBM}(\mathbf{n}, P_2)$ where again $\mathbf{n} = (n_1, 10n_1)$ bu the connection probability matrix is given by:

$$P_2 = \begin{bmatrix} 2\log^2(n)/n & \log(n)/n \\ \log(n)/n & \log(n)/n \end{bmatrix}.$$

This models the problem of extracting a cluster from a weak background.

The results of these three experiments are presented in Figures 2.2, 2.3 and 2.4. As there was a wide disparity in run times, we plot these using a logarithmic scale. To motivate further the consistency of `SingleClusterPursuit`, we include a box plot for the Jaccard Indices.

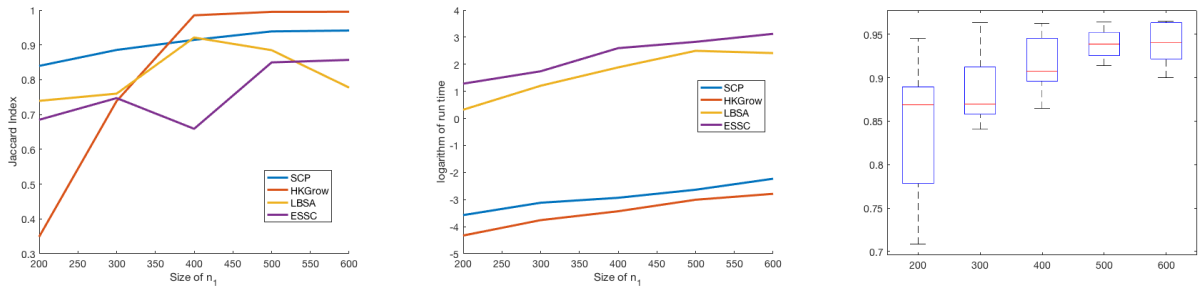

Figure 2.2: Results from Experiment 1. From left to right: The Jaccard index as a function of cluster size, the run time as a function of cluster size, and a box plot of the Jaccard indices for `SingleClusterPursuit`.

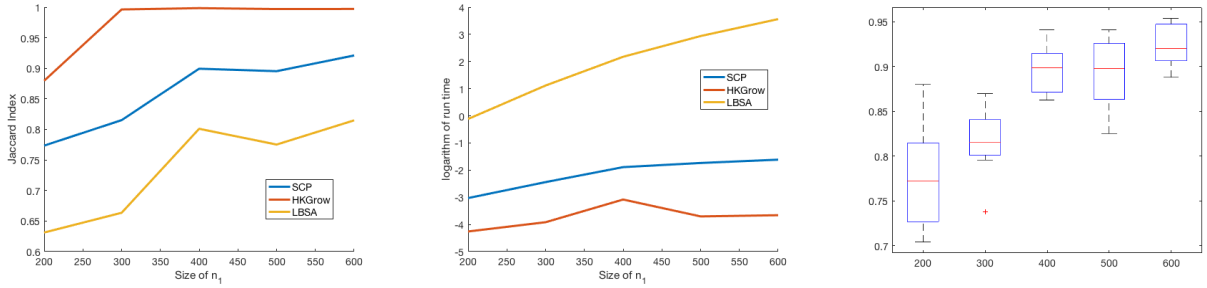

Figure 2.3: Results from Experiment 2. From left to right: The Jaccard index as a function of cluster size, the run time as a function of cluster size, and a box plot of the Jaccard indices for `SingleClusterPursuit`.

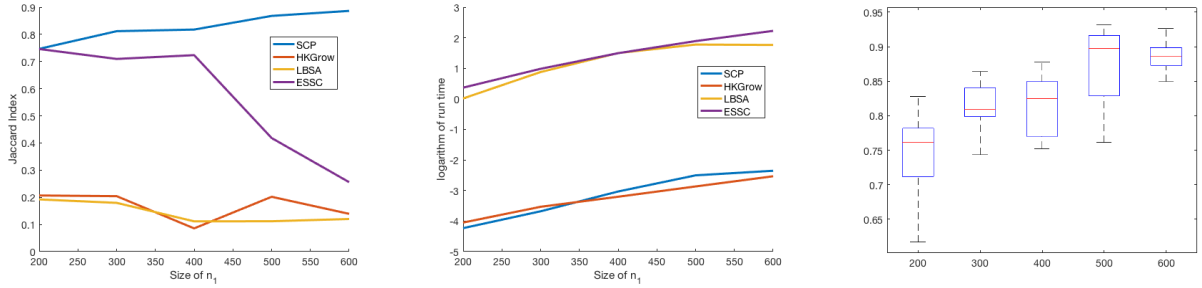

Figure 2.4: Results from Experiment 3. From left to right: The Jaccard index as a function of cluster size, the run time as a function of cluster size, and a box plot of the Jaccard indices for `SingleClusterPursuit`.

2.6.4 Real Data sets 1: Social Networks

The `facebook100` dataset consists of anonymized Facebook "friendship" networks at 100 American universities, and was first introduced and studied in [84]. It contains, for each college or university, a graph whose vertices correspond to undergraduates with a Facebook account at that institution. Edges connect students who were friends on Facebook the day (in September 2005) the data was collected. Certain demographic markers (year of entry, gender, residence, high school etc.) were also collected in an anonymized format. We focus on four schools, California Institute of Technology (Caltech), Rice, University of California, Santa Cruz (UCSC) and Smith College, identified by Traud *et. al.* ([84]) as being most strongly clustered by residence. We treat the residence assignments as the ground truth clusters. We note that there are always some students whose residential affiliation is unknown; we treat these as background vertices. For each cluster, we run each algorithm ten times, each time with a different set of uniformly randomly selected seed vertices. For `SingleClusterPursuit`, `HKGrow` and `LBSA` the seed set consists of 5 randomly selected vertices. For `ESSC`, the seed set is the neighborhood of a certain vertex in the ground truth cluster. We tried taking this vertex to be the highest degree vertex in the cluster (as in [89]) as well as selecting this vertex uniformly at random. Experimentally, we observed better results for the latter, so we report these. We note that for the larger networks (*i.e.* Smith, Rice and UCSC) `ESSC` did not converge within a reasonable amount of time. Important properties of the networks are reported in 2.1, while results are presented in Figures 2.5, 2.7, 2.6 and 2.8.

	Vertices	Clusters	Max cluster size	Min cluster size	Mean cluster size
Caltech	769	8	99	44	74.63
Smith	2970	36	113	12	70.17
Rice	4087	9	414	382	396
UCSC	8991	10	925	622	773.7

Table 2.1: Basic properties of the four social networks studied.

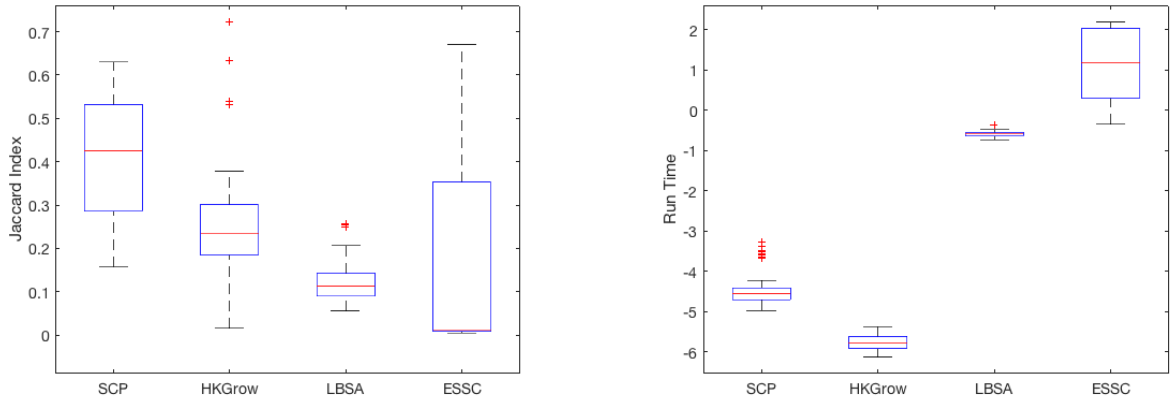

Figure 2.5: Results for Caltech Social Network: Jaccard index on left and run time on right.Note that the run times are presented in a logarithmic scale.

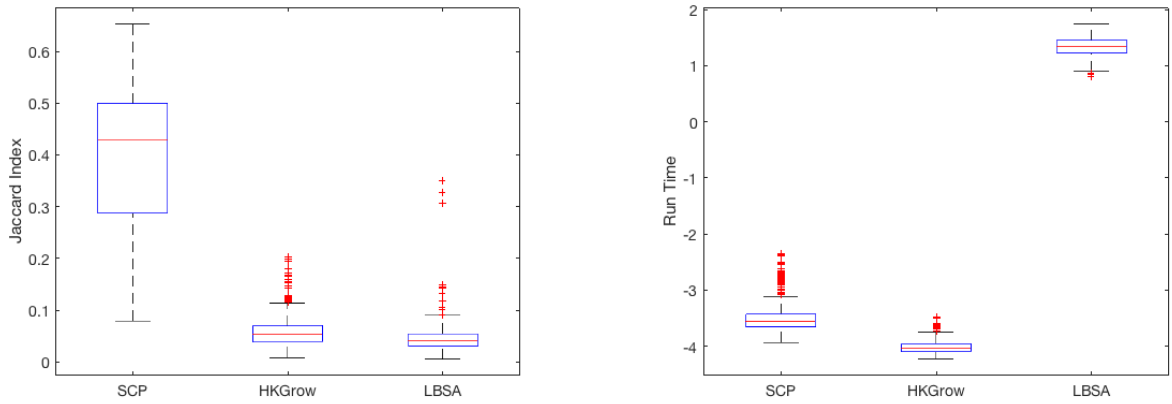

Figure 2.6: Results for Smith Social Network: Jaccard index on left and run time on right. Note that the run times are presented in a logarithmic scale.

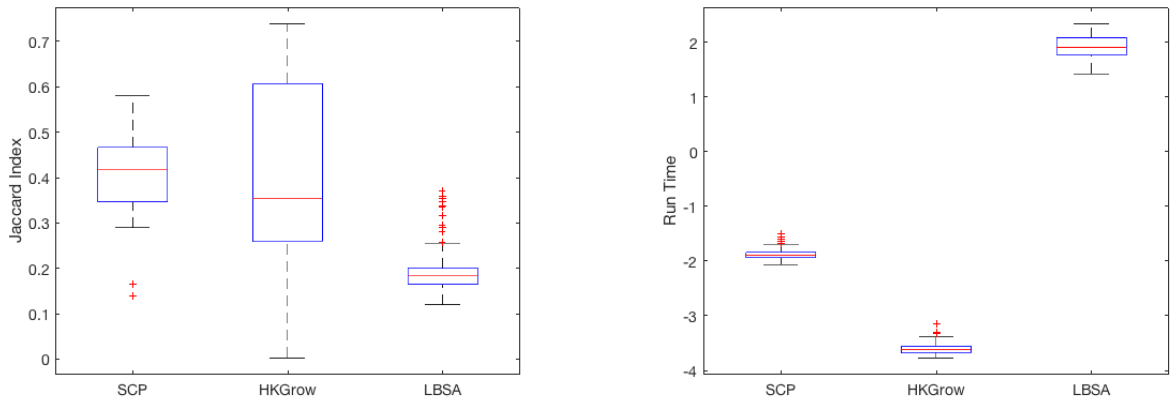

Figure 2.7: Results for Rice Social Network: Jaccard index on left and run time on right. Note that the run times are presented in a logarithmic scale.

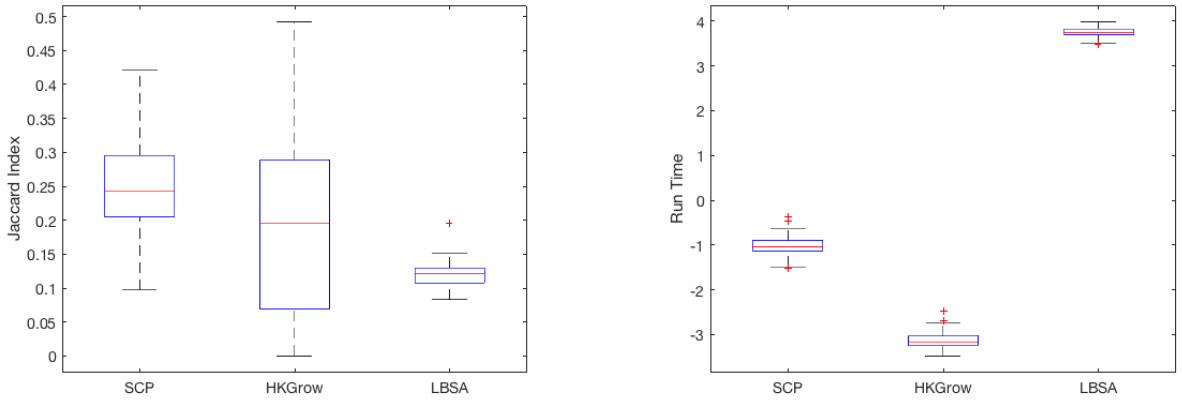

Figure 2.8: Results for UCSB Social Network: Jaccard index on left and run time on right. Note that the run times are presented in a logarithmic scale.

2.6.5 Real Data sets 2: Machine Learning Benchmarks

Of particular interest to us is extraction of clusters of linear size (*i.e.* $n_1 \sim n/k$) from the K-Nearest Neigbor (K-NN) graphs that arise in Machine Learning. Typically, the data points in such data sets are presented as real-valued vectors, for example, vectorized grayscale images. So, let us first discuss how to turn such a data set $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^D$ into a graph G so that algorithms such as `SingleClusterPursuit` may be used.

Preprocessing Euclidean Data Given such an $\mathcal{X}$, we construct a weighted graph on n vertices with weighted adjacency matrix defined as: $A_{ij} = \varphi(\|\mathbf{x}_i - \mathbf{x}_j\|)$. Here φ is referred to as a *kernel function* and should, at least, be non-negative, continuous at 0 with $\varphi(0) = 1$, non-increasing on $[0, \infty)$ and sufficiently fast decaying (requiring $s^q\varphi(s) = o(1)$ for all $q > 0$ is typical, see [9]). The kernel function usually depends on some user-specified parameters, for example $\varphi_\sigma(s) := \exp(-s^2/\sigma^2)$ is a common choice [73]. Such an approach results in a complete graph, but in dealing with large data sets a sparse graph is desired, due to memory constraints. Thus, many researchers consider variants of K-NN graphs, defined loosely as

inserting an edge between i and j if and only if $\mathbf{x}_i$ is amongst the K closest data points in $\mathcal{X}$ to $\mathbf{x}_j$, or *vice versa*. Here we use the following variant of a K-NN graph that incorporates the local scaling of Zelnik-Manor and Perona [95], which we first learned of from [51].

- Fix parameters r and K.
- For all $i \in [n]$, define $\sigma_i := \|\mathbf{x}_i - \mathbf{x}_{[r,i]}\|$, where $\mathbf{x}_{[r,i]}$ denotes the r-th closest point in $\mathcal{X}$ to $\mathbf{x}_i$. (If there is a tie, break it arbitrarily). Let $\mathrm{NN}(\mathbf{x}_i, K) \subset \mathcal{X}$ denote the set of the K closest points in $\mathcal{X}$ to $\mathbf{x}_i$. Again, one may break ties arbitrarily if they occur.
- Define $\tilde{A}$ as: $\tilde{A}_{ij} = \begin{cases} \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma_i\sigma_j\right) & \text{if } \mathbf{x}_j \in NN(\mathbf{x}_i, K) \\ 0 & \text{otherwise} \end{cases}$
- Observe that $\tilde{A}$ is not necessarily symmetric, as it may occur that $\mathbf{x}_j \in \mathrm{NN}(\mathbf{x}_i, K)$ while $\mathbf{x}_i \notin \mathrm{NN}(\mathbf{x}_j, K)$. So, symmetrize $\tilde{A}$ to obtain A, the adjacency matrix of G. In this dissertation we consider two symmetrizations:

$$A_{\text{mult}} := \tilde{A}^\top \tilde{A} \quad \text{and} \quad A_{\max} \text{ where } (A_{\max})_{ij} = \max\left\{\tilde{A}_{ij}, \tilde{A}_{ji}\right\}$$

Clearly $A_{\max}$ is sparser than A_{mult}, although we observe that `SingleClusterPursuit` performs slightly better when using A_{mult}.

We consider two data sets, `MNIST` and `OptDigits`. For both data sets we consider both $A_{\max}$ and A_{mult}.

`MNIST`. This data set consists of $n = 70\ 000$ grayscale images of the handwritten digits 0–9, all of size 28×28. We vectorize these images to obtain data points $\mathbf{x}_i \in \mathbb{R}^{784}$. There are approximately 7 000 images of each digit.

`OptDigits`. Similar to `MNIST`, this data set also consists of images of handwritten digits 0–9, although now they are of size 8×8 which vectorize to give data points $\mathbf{x}_i \in \mathbb{R}^{64}$, and there are only $n = 5620$ of them. As before, the clusters are balanced with approximately

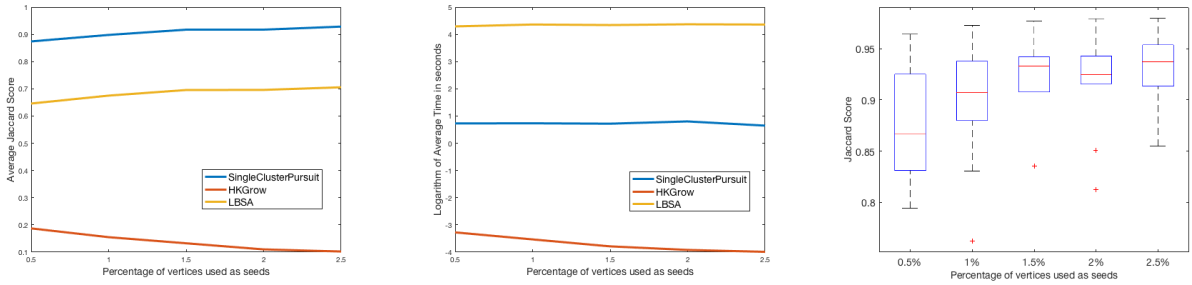

Figure 2.9: Cluster extraction for `MNIST` with A_{mult} as the adjacency matrix. Results averaged over all 10 digits. From left to right: The Jaccard index as a function of cluster size, the run time as a function of cluster size, and a box plot of the Jaccard indices for `SingleClusterPursuit`.

560 images of each digit.

We vary the fraction of data points to be labeled from 0.5% to 2.5% in increments of 0.5%. For each sample fraction, $g = 0.005, \ldots, 0.025$, we perform one experiment for each of the 10 clusters in the respective data set. We choose $g(n/10)$ data points from the cluster uniformly at random and use this as Γ. The parameters for `LBSA` are as described in §2.6.1. For `SingleClusterPursuit` we take $\epsilon = 0.13$ and $R = 0.7$ in all experiments. For the larger data set `MNIST` we take $t = 3$ while for `OptDigits` we take $t = 10$. Note that in all experiments we do not use any *a priori* information about the sizes of the clusters. Instead, we set the parameter $\hat{n}_1 = n/10$, which only encodes the assumption that we expect the clusters to be more or less balanced.

The results for `MNIST` are presented in Figures 2.9 and 2.10 while the results for `OptDigits` are presented in Figures 2.11 and 2.12. Note that for both data sets there is a significant gap in Jaccard index between the best and worst results for any given sampling fraction. This is due to the significant gap in difficulty between finding the clusters for "easy" digits, such as 0, and "hard" digits, such as 7.

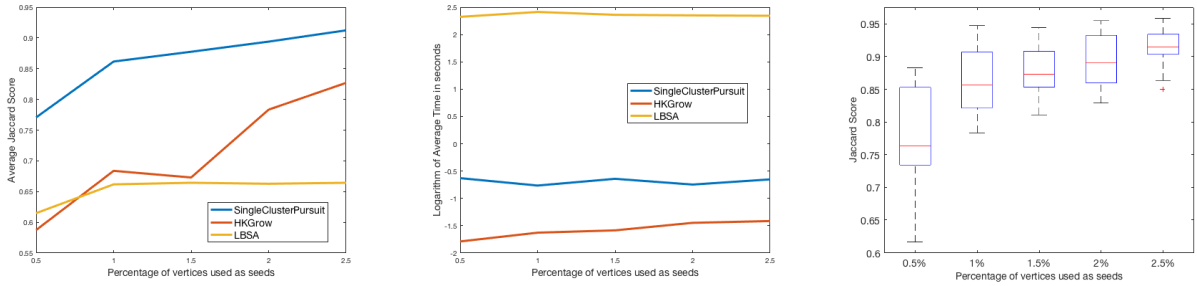

Figure 2.10: Cluster extraction for `MNIST` with $A_{\max}$ as the adjacency matrix. Results averaged over all 10 digits. From left to right: The Jaccard index as a function of cluster size, the run time as a function of cluster size, and a box plot of the Jaccard indices for `SingleClusterPursuit`.

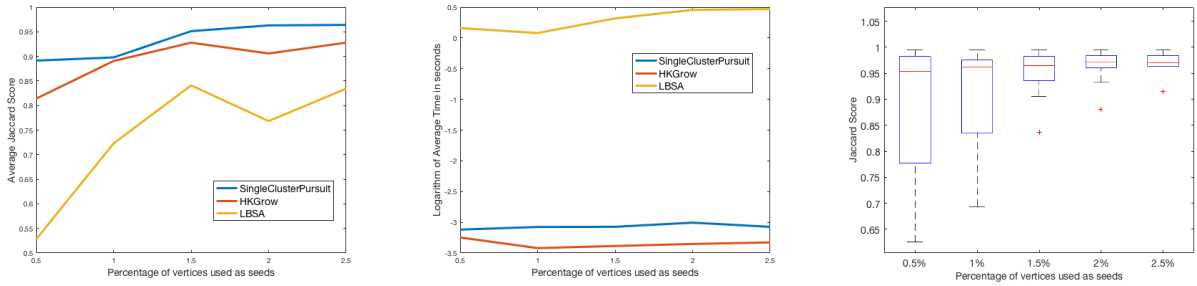

Figure 2.11: Cluster extraction for `OptDigits` with $A_{\max}$ as the adjacency matrix. Results averaged over all 10 digits. From left to right: The Jaccard index as a function of cluster size, the run time as a function of cluster size, plotted in logarithmic scale, and a box plot of the Jaccard indices for `SingleClusterPursuit`.

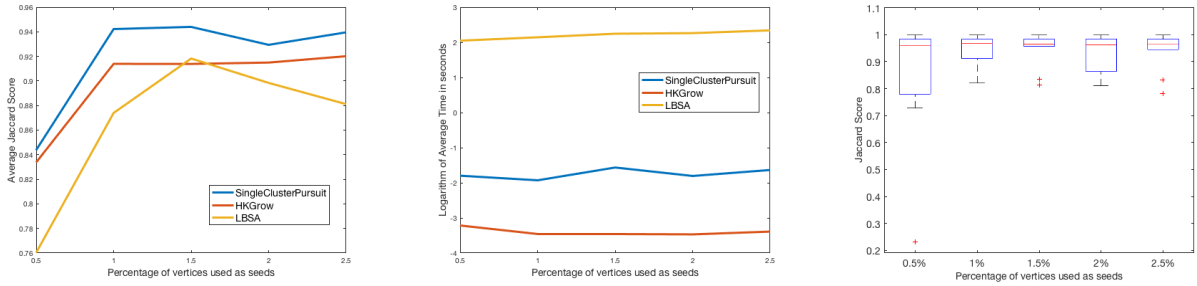

Figure 2.12: Cluster extraction for `OptDigits` with A_{mult} as the adjacency matrix. Results averaged over all 10 digits. From left to right: The Jaccard index as a function of cluster size, the run time as a function of cluster size, and a box plot of the Jaccard indices for `SingleClusterPursuit`.

2.6.6 Extension to Semi-Supervised Learning

Here we explore the use of an iterated `SingleClusterPursuit`, capable of finding all clusters in a data set, on the `MNIST` data set. We call the precise algorithm we use `IteratedSingleClusterPursuit`, and present it as Algorithm 7. Note that as input `IteratedSingleClusterPursuit` requires a set of seed vertices $\Gamma_a \subset C_a$ for each cluster. We collate the sets into an array $\mathbf{\Gamma} = \{\Gamma_1, \ldots, \Gamma_k\}$ and, following standard terminology in the machine learning literature, refer to $\mathbf{\Gamma}$ as the *labeled data*. We collect the values of $\hat{n}_1$ into a vector $\hat{\mathbf{n}}$. Note that there is an interesting flexibility in this approach which manifests in the "Cleanup" steps of 7. Because `SingleClusterPursuit` is extractive, it can happen that once one has extracted clusters $C_1^{\text{temp}}, \ldots C_k^{\text{temp}}$ there are still vertices which have not been assigned a cluster, denoted `LeftOver` in Algorithm 7. One can now treat the extracted clusters $C_1^{\text{temp}}, \ldots C_k^{\text{temp}}$ as labeled data, train a supervised classifier on them, and use it to classify the remaining data points represented by `LeftOver`. In `IteratedSingleClusterPursuit` as presented in 7 we use a simple nearest neighbors classifier, although certainly more sophisticated approaches are possible.

In our numerical experiment with `MNIST` we draw the same number of labeled examples, randomly and uniformly, from each cluster: $|\Gamma_a| = g|C_a|$. As before we vary the sampling fraction g in increments of 0.005 from 0.005 to 0.025. We set the cluster sizes to their exact values, that is $\hat{\mathbf{n}} = (n_1, n_2, \ldots, n_k)$. We use the same A_{mult} as described above as the adjacency matrix. The results displayed in Table 2.2 are averaged over 20 independent trials. In Table 2.3 we compare our results with other state-of-the-art results for Semi-Supervised clustering on `MNIST`.

2.6.7 Verifying the Asymptotic Bound on Run Time

In §2.5 we claim that the run time of `SingleClusterPursuit` is $O(d_{\max} \log(n)n)$ and argued further that when $G \sim \text{SSBM}(n, k, p, q)$, with parameters $p = \log^2(n)/n$ and $q = b\log(n)/n$

Amount of Labeled Data	Accuracy
0.5%	95.29%
1%	97.03%
1.5%	97.37%
2%	97.49%
2.5%	97.57%

Table 2.2: Accuracy of classification (defined as $\#\{$ images correctly classified$\}/n$) using `IteratedSingleClusterPursuit`

.

Method	Labelled	Accuracy
TSVM [31]	1000	95.62%
TVRF [94]	600	96.7%
Deep Generative Model [53]	1000	97.13%
`IteratedSingleClusterPursuit`	1050	97.37%
Multi-Class MBO with Auction Dynamics [51]	700	97.43%
`IteratedSingleClusterPursuit`	1400	97.49%
Ladder Networks [76]	1000	99.16%

Table 2.3: Comparing `IteratedClusterPursuit` to other, state-of-the-art, semi-supervised methods on MNIST. We remark that the Ladder Network approach requires a full two hours of training on a GPU. In constrast, `IteratedSingleClusterPursuit` runs in under 20 seconds, although it does require $\sim$ 20 minutes to create the adjacency matrix A from the raw image data.

Algorithm 7 `IteratedSingleClusterPursuit`

Input: Adjacency matrix A, parameters $\epsilon, R \in (0,1)$ and depth of random walk t. Labeled data $\mathbf{\Gamma}$. Vector of approximate cluster sizes, $\hat{\mathbf{n}}$.
Initialization: $N = n = \texttt{size}(A, 1)$
for $a = 1 : k-1$ **do**
  $C_a^{\text{temp}} = \texttt{SingleClusterPursuit}(A, \epsilon, \Gamma(a), \hat{\mathbf{n}}(a), t, R)$
  $\texttt{NewInds} = [n] \setminus C_a^{\text{temp}}$
  $A = A(\texttt{NewInds}, \texttt{NewInds})$
  $n = |\texttt{NewInds}|$
end for
$\Omega^{\text{final}} = [N] \setminus (C_1 \cup C_2 \cup \ldots \cup C_{k-1})$
Find C_k^{temp} using Ω^{final} and steps (2)–(4) of `SingleClusterPursuit`.
Cleanup: Let $\texttt{LeftOver} = [N] \setminus (C_1 \cup \ldots \cup C_k)$
For all $a \in [k]$ and $i \in \texttt{LeftOver}$ define $\texttt{Score}(a, i) = \sum_{j \in C_a \cup \mathcal{N}(i)} A_{ij}$
Let $C_b^{\text{new}} = \{i \in \texttt{LeftOver} : \text{argmax}_{a \in [k]} \texttt{Score}(a, i) = b\}$
Output: for $a \in [k]$ let $C_a^{\#} = C_a^{\text{temp}} \cup C_a^{\text{new}}$

for some constant b, that this reduces to $O(\log^3(n)n)$. Here we verify that this is the case. We draw graphs from SSBM(n, k, p, q), with p and q as indicated above, for n varying from 5 000 to 25 000, in increments of 500. For each value of n, we compute the average run time of `SingleClusterPursuit` over ten independent trials. We then plot the *logarithm* of the average run time, as a function of n in Figure 2.13. Section §2.5 predicts that the logarithm of the run time should be equal to $\log(n) + 3\log(\log(n)) + \log(C)$ for some constant C. In Figure 2.13 we thus plot the predicted run time, $T = \log(n) + 3\log(\log(n)) - 15.99$, where the constant was determined by fitting to the data. As can be seen, there is excellent agreement between the theoretical prediction and experimental result.

2.6.8 Analysis of Experimental Results

`SingleClusterPursuit` performs better than the other algorithms considered for every real data set considered, and is also the best algorithm for the "cluster extraction from a weak background" problem of Synthetic Data Experiment 3. Although `SingleClusterPursuit` is

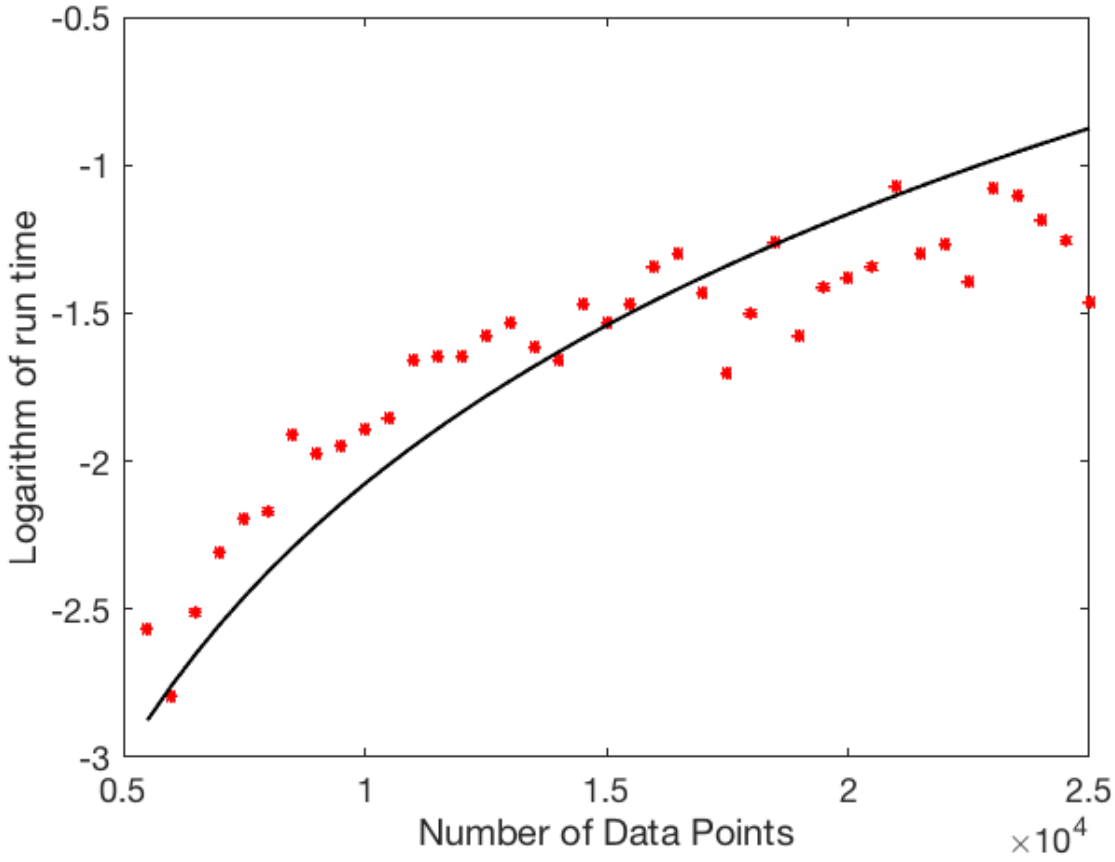

Figure 2.13: Comparing theoretical versus experimental run times

outperformed by `HKGrow` in most synthetic cases where there are well-defined, well-separated clusters, *i.e.* Experiments 1 and 2, we argue that such data sets are not very realistic. It is interesting to note that `SingleClusterPursuit` performs equally well on very different kinds of data—for example social networks and K-NN graphs. Finally we note that the run time of `SingleClusterPursuit` is low, and grows slowly with n as predicted by §2.5. Based on all of the above, it appears that `SingleClusterPursuit` is an excellent general purpose cluster extraction algorithm.

Chapter 3

Dynamic Cluster Pursuit

The discussion thus far has tacitly assumed that our networks of interest are static, *i.e.* they do not change with time. This is not always a realistic assumption, and indeed many real world networks do evolve as time progresses. Consider the example of an email network, where users are represented by vertices which are connected if the users concerned exchanged emails within a given time period (say a week). Hence every week provides a new static graph whose edges may be markedly different from the previous week's. We shall represent such dynamic graphs as a sequence of ordinary, static, graphs: $\mathbf{G} = \left\{G^{(1)}, \ldots, G^{(T)}\right\}$ and refer to the $G^{(t)}$ as "snapshots." For conceptual clarity, we shall restrict to the case where all the $G^{(t)}$ all have the same vertex set, namely V, although this is not essential.

Consider again the example of email correspondence described above. It is reasonable to assume that each $G^{(t)}$ has clusters $C_1^{(t)} \cup \ldots \cup C_k^{(t)}$ corresponding to people who work together, are in the same friendship circle and so on. Certainly these clusters will change with time, for example as people change jobs or move city, but we expect few vertices to change clusters between adjacenct snapshots. We emphasize that there are two temporal processes going on here. Between snapshots there is an essentially random reconfiguration of the edges between existing vertices, while over the longer term there is the more meaningful process of vertices transitioning between clusters. The problem of tracking these evolving clusters is called the "dynamic clustering" problem.

We considered a more refined variant of this problem, where one is interesting in tracking the evolution of a single cluster, $C_a^{(t)}$, over time. In analogy with earlier work, we term this the "dynamic cluster extraction" problem. Clearly, any algorithm that solves this problem can be iterated to track all k clusters. However if one is only interested in a certain cluster, then using a dynamic cluster extraction algorithm is preferable is it does not use unnecessary computational resources. In §3.3 we will present a novel algorithm that solves this problem. Before that, we briefly review the literature on dynamic clustering (§3.1) and discuss various probabilistic models of dynamic graphs with clusters that generalize the stochastic block model (§3.2). Numerical experiments are presented in §3.4.

3.1 Overview of Dynamic Clustering

To the best of the author's knowledge, the first paper to consider the idea of dynamic clustering was [20]. Since then, interest in this problem has grown steadily, as documented in the survey articles [6] (2014), [45] (2016) and [77] (2018). In particular, over 50 approaches to dynamic clustering are documented in [77] (although we remark that no approaches to *dynamic cluster extraction* are presented). Note that dynamic clustering can mean different things to different authors. Some authors assume that one first has access to all snapshots $\{G^{(1)}, \ldots, G^{(T)}\}$ and then tries to determine a sequence of clusterings $\{\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(T)}\}$. We refer to such approaches as "offline" methods. On the other hand, one could try to find clusterings in an "online" manner, whereby in finding the t-th clustering, $\mathbf{C}^{(t)}$, one only has access to $\{G^{(1)}, \ldots, G^{(t)}\}$ and $\{\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(t-1)}\}$. Our focus in this chapter is on online methods.

In an attempt to categorize the disparate approaches to dynamic clustering, Rossetti and Cazabet introduced a taxonomy of dynamic clustering approaches in [77]. We follow their

lead in surveying the literature, and use their terminology to describe the three main families of dynamic clustering algorithms.

3.1.1 Instant Optimal methods

Arguably the simplest approach to dynamic clustering, instant optimal methods apply a static clustering algorithm to each snapshot, $G^{(t)}$, to find a clustering $\mathbf{C}^{(t)}$, and then use a set-matching procedure to identify these static communities across time steps. The primary drawback of such approaches is that the communities found in this manner tend to vary wildly. Indeed, even for static networks it is widely accepted that there is no unique "correct" decomposition of the vertices into communities (cf. [77]). Moreover, many common static clustering algorithms (for example Spectral Clustering, or GenLouvain [34]) incorporate some kind of randomization, and hence may find different clusterings on different runs, even on the same graph. It follows that even if there is little change from $G^{(t)}$ to $G^{(t+1)}$, the clusterings found by a static clustering algorithm may be significantly different, and hence there may be little interpretability in dynamic clusters formed by identifying such static clusters across snapshots.

3.1.2 Temporal Trade Off methods

These algorithms seek to "update" the clustering found in $G^{(t-1)}$ to a new clustering for $G^{(t)}$. It is usually assumed that a clustering for $G^{(1)}$ is given, or is found using a static clustering method. The updating can take a variety of forms, for example one approach is to "warm-start" a static clustering algorithm on $G^{(t)}$, such as GenLouvain [34], with $\mathbf{C}^{(t-1)}$, thus making it likely that the $\mathbf{C}^{(t)}$ found will be close to $\mathbf{C}^{(t-1)}$. Another set of approaches, referred to in [77] as "Informed Community Detection by Multi-Objective Optimization", seeks to find $\mathbf{C}^{(t)}$ by balancing the competing goals of finding a clustering which fits the current snapshot and finding a clustering which is not too dissimilar to that found for the previous snapshot. Loosely speaking, such methods use a measure of (static) cluster quality $F(\mathbf{C}, G^{(t)})$, such

as normalized cut (see Definition 1.2.5) and add a regularizing term $H(\mathbf{C}, G^{(t-1)})$ which penalizes potential clusterings of $G^{(t)}$ that deviate too much from recent history. One then finds an optimal clustering of $G^{(t)}$, namely $\mathbf{C}^{(t)}$, as:

$$\mathbf{C}^{(t)} = \text{argmin}\left(F(\mathbf{C}, G^{(t)}) + \alpha H(\mathbf{C}, G^{(t-1)})\right)$$

Two such regularizing terms are suggested in [22]. The first, which they call H_{PCQ} for *Preserving Cluster Quality* penalizes $\mathbf{C}$ if it is also not a decent clustering for $G^{(t-1)}$. That is, $H_{\text{PCQ}}(\mathbf{C}) = F(\mathbf{C}, G^{(t-1)})$ where F is the same static measure of cluster quality as before. The second penalizes $\mathbf{C}$ if the constituent clusters of $\mathbf{C}$, namely $C_1, \ldots, C_k$, differ from $C_1^{(t-1)}, \ldots, C_k^{(t-1)}$. They term this function H_{PCM} for *Preserving Cluster Membership* and define it using the chi-squared statistic:

$$H_{\text{PCM}}(\mathbf{C}, C^{(t-1)}) = -\sum_{a=1}^{k}\sum_{b=1}^{k} \frac{\left|C_a^{(t-1)} \cap C_b\right|}{\left|C_a^{(t-1)}\right| |C_b|} \tag{3.1}$$

In both cases the parameter α needs to be specified by the user.

Another important sub-category of Temporal Trade Off methods are the so-called "Informed Community Detection by Network Smoothing" approaches. Emblematic of such approaches is the AFFECT paradigm, introduced in [92]. In this paper, the authors suppose that at each time step the adjacency matrix can be decomposed as $A^{(t)} = \Psi^{(t)} + N^{(t)}$ where $\Psi^{(t)}$ is an unknown, "true" weighted adjacency matrix while $N^{(t)}$ is a matrix of zero-mean Gaussian noise. If $\Psi^{(t)}$ were known, the authors argue that performing any static clustering on $\Psi^{(t)}$ instead of $A^{(t)}$ would result in optimal results. Of course $\Psi^{(t)}$ is not known, hence the authors suggest using an estimator $\hat{\Psi}^{(t)}$ instead. If one only had access to a single snapshot $G^{(t)}$ then $\hat{\Psi}^{(t)} = A^{(t)}$ would be the best estimate of $\Psi^{(t)}$ we could hope for, but as we have access to prior snapshots the authors propose an estimator of the form $\hat{\Psi}^{(t)} = \alpha^{(t)}\hat{\Psi}^{(t-1)} + (1 - \alpha^{(t)})A^{(t)}$ where the parameter $\alpha^{(t)}$ is called the *forgetting factor* and the authors describe a way to determine it from the data $\{A^{(1)}, \ldots, A^{(t)}\}$. Finally, the

authors of [22] present two efficient algorithms, based on the normalized spectral clustering of Shi and Malik (cf. [78] or Algorithm 2), for approximately solving Problem (3.1) for the cases $H = H_{\text{PCQ}}$ and $H = H_{\text{PCM}}$, which we deem `EvolSpec-PCQ` and `EvolSpec-PCM` respectively.

Our proposed algorithm for dynamic cluster extraction, `DynamicClusterPursuit`, is most similar to to Temporal Trade Off approaches to dynamic clustering.

3.1.3 Cross-Time Community Discovery methods

Methods in this category differ from the previous two in that they are all offline approaches. That is, they require as input all T snapshots, and then perform the clustering in a single process. Thus, a key distinguishing feature of such approaches is how they aggregate the snapshots $G^{(1)}, \ldots, G^{(T)}$. We mention the *multislice* approach of Mucha, Porter and collaborators which stitches the snapshots together into a single network by connecting each node in one snapshot to itself in every other snapshot. One can vary the weight of these "interslice" edges, which allows one to place more or less importance on historical trends. A multislice version of modularity is then defined (see [69] for the definition) and a clustering which maximizes this modularity is then sought. We refer the reader to the report [69] for further details. As the focus in this chapter is on online dynamic clustering, we shall have little more to say about this category of approaches in the sequel.

3.2 Probabilistic Models of Dynamic Random Graphs

We remind the reader of the definition of the Stochastic Block Model, as introduced in §1.2.1. We alter the notation slightly so as to emphasize the role of the clustering $\mathbf{C}$.

Definition 3.2.1. Let $\mathbf{C} = \{C_1, \ldots, C_k\}$ denote a partition of the vertex set V and let $P \in \mathbb{R}^{k \times k}$ be a non-negative matrix. We say that G is drawn from the stochastic block

model SBM($\mathbf{C}, P$) (and write $G \sim \text{SBM}(\mathbf{C}, P)$) if, for every pair of vertices i, j with $i \in C_a$ and $j \in C_b$ the edge $\{i, j\}$ is inserted independently with probability P_{ab}.

There exist several extensions of the Stochastic Block Model to the setting of dynamic graphs, all under the name 'Dynamic Stochastic Block Model' (see [93, 75, 91, 48]). Here we shall use the model introduced by Yang *et al* in [93].

Definition 3.2.2. Fix an initial clustering $\mathbf{C}^{(1)}$, the number of time steps T, a connection probability matrix P and a cluster-switching probability matrix Q, which is required to be doubly stochastic. We say that the sequence of snapshots $\mathbf{G} = \{G^{(1)}, \ldots, G^{(T)}\}$ is drawn from the Dynamic Stochastic Block Model DSBM($\mathbf{C}^{(1)}, P, Q, T$) if:

1. $G^{(1)} \sim \text{SBM}(\mathbf{C}^{(1)}, P)$.
2. For each $2 \leq t \leq T$ and every $1 \leq a \leq b \leq k$ vertex $i \in C_a^{(t-1)}$ is placed in $C_b^{(t)}$ with probability Q_{ab}. Then $G^{(t)} \sim \text{SBM}(\mathbf{C}^{(t)}, P)$.

In [75, 91] models are proposed where the connection probability matrix P is allowed to change in time, while in [90, 48] a dynamic model is explored that allows for mixed cluster membership at each time step. Models which allow for the vertex set V to change are also possible. In analogy with the symmetric stochastic block model introduced in §1.2.1, we define:

Definition 3.2.3. If $\mathbf{G} \sim \text{DSBM}(\mathbf{C}^{(1)}, P, Q, T)$ with $P_{aa} = p$ for all $a \in [k]$ and $P_{ab} = q$ for all $a \neq b$, and $Q_{aa} = 1 - \epsilon$ for all $a \in [k]$ while $Q_{ab} = \epsilon/(k-1)$ for $a \neq b$ and $|C_1^{(1)}| = |C_2^{(1)}| = \ldots = |C_k^{(1)}| = n/k$ we say that $\mathbf{G}$ is drawn from the *Symmetric Dynamic Stochastic Block Model*, and write $\mathbf{G} \sim \text{SDSBM}(n, k, p, q, \epsilon, T)$

This is essentially the model considered in [43]. Note that the cluster switching probabilities in Definition 3.2.3 are chosen so that the sizes of the clusters remain relatively stable. In fact:

Theorem 3.2.4. *Let* $\mathbf{G} \sim SDSBM(n, k, p, q, \epsilon, T)$. *Then for each a* $n_a^{(t)} := |C_a^{(t)}|$ *is a random variable with* $\mathbb{E}\left[n_a^{(t)}\right] = n/k$.

Proof. By the law of iterated expectation:

$$\begin{aligned}\mathbb{E}\left[n_a^{(t)}\middle|n_a^{(1)}\right] &= \mathbb{E}\left[\mathbb{E}\left[n_a^{(t)}\middle|n_a^{(2)}, n_a^{(1)}\right]\middle|n_a^{(1)}\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[\mathbb{E}\left[n_a^{(t)}\middle|n_a^{(3)}, n_a^{(2)}, n_a^{(1)}\right]\middle|n_a^{(2)}, n_a^{(1)}\right]\middle|n_a^{(1)}\right] \\ &= \ldots = \mathbb{E}\left[\mathbb{E}\left[\ldots\mathbb{E}\left[n_a^{(t)}\middle|n_a^{(t-1)}, \ldots, n_a^{(1)}\right]\middle|\ldots\right]\middle|n_a^{(1)}\right]\end{aligned}$$

To unpack this, observe that for any s:

$$\mathbb{E}\left[n_a^{(s)}\middle|n_a^{(s-1)}, \ldots, n_a^{(1)}\right] = \mathbb{E}\left[n_a^{(s)}\middle|n_a^{(s-1)}\right] = (1-\epsilon)n_a^{(s-1)} + \frac{\epsilon}{k-1}(n - n_a^{(s-1)}) = \frac{\epsilon n}{k-1} + (1 - \frac{k\epsilon}{k-1})n_a^{(s-1)} \tag{3.2}$$

It follows that:

$$\begin{aligned}\mathbb{E}\left[n_a^{(t)}\middle|n_a^{(1)}\right] &= \mathbb{E}\left[\ldots\mathbb{E}\left[\frac{\epsilon n}{k-1} + (1 - \frac{k\epsilon}{k-1})n_a^{(t-1)}\middle|n_a^{(t-1)}\right]\ldots\middle|n_a^{(1)}\right] \\ &= \mathbb{E}\left[\ldots\mathbb{E}\left[\frac{\epsilon n}{k-1} + (1 - \frac{k\epsilon}{k-1})\left(\frac{\epsilon n}{k-1} + (1 - \frac{k\epsilon}{k-1})n_a^{(t-2)}\right)\middle|n_a^{(t-2)}\right]\ldots\middle|n_a^{(1)}\right] \\ &= \mathbb{E}\left[\ldots\mathbb{E}\left[\left(1 + (1 - \frac{k\epsilon}{k-1}) + (1 - \frac{k\epsilon}{k-1})^2\right)\frac{\epsilon n}{k-1} + (1 - \frac{k\epsilon}{k-1})^2 n_a^{(t-2)}\middle|n_a^{(t-2)}\right]\ldots\middle|n_a^{(1)}\right]\end{aligned}$$

Continuing in this fashion we observe a geometric sequence developing. So:

$$\begin{aligned}\mathbb{E}\left[n_a^{(t)}\middle|n_a^{(1)}\right] &= \frac{\epsilon n}{k-1}\sum_{j=0}^{t-1}\left(1 - \frac{k\epsilon}{k-1}\right)^j + \left(1 - \frac{k\epsilon}{k-1}\right)^{t-1} n_a^{(1)} \\ &= \frac{\epsilon n}{k-1}\frac{1 - \left(1 - \frac{k\epsilon}{k-1}\right)^{t-1}}{\frac{k\epsilon}{k-1}} + \left(1 - \frac{k\epsilon}{k-1}\right)^{t-1}\frac{n}{k} \\ &= \frac{n}{k}\left(1 - \left(1 - \frac{k\epsilon}{k-1}\right)^{t-1}\right) + \frac{n}{k}\left(1 - \frac{k\epsilon}{k-1}\right)^{t-1} = \frac{n}{k}\end{aligned}$$

□

However tail bounds on the cluster sizes $|C_a^{(t)}|$ seem surprisingly hard to prove. In order to analyze the probability of success of the `DCP-Thresh` algorithm in §3.3.2, we introduce a further simplification that guarantees that the cluster sizes are constant (*i.e.* not just constant in expectation):

Definition 3.2.5. We say that $\mathbf{G}$ is drawn from the *Constrained Dynamic Stochastic Block Model* $\mathrm{CDSBM}(n, k, p, q, \epsilon, T)$ if

1. $\mathbf{C}^{(1)}$ is such that $|C_a^{(1)}| = n/k$ for all $a \in [k]$.
2. $G^{(1)} \sim \mathrm{SSBM}(\mathbf{C}^{(1)}, p, q)$.
3. For each $2 \leq t \leq T$ and all $a, b \in [k]$ with $a \neq b$ precisely $\epsilon|C_a^{(t-1)}|/(k-1)$ vertices are selected uniformly at random from $C_a^{(t-1)}$ to be moved to $C_b^{(t)}$. The vertices in $C_a^{(t-1)}$ which are not selected in this manner are also placed in $C_a^{(t)}$

Remark 3.2.6. The upshot of this rather convoluted construction is that the cluster sizes are constant: $|C_a^{(t)}| = n/k$ for all t. Moreover, for every t, and every a, $|C_a^+| = |C_a^-| = \epsilon n/k$.

It is also of interest to have a model of *weighted* dynamic graph. Several approaches use Dynamic Gaussian Mixture Models, for example [19]. We mention also the model of Xu, Kliger and Hero [91] alluded to in §3.1.2, where the adjacency matrix A is modeled by a block-constant matrix $\Psi^{(t)}$ plus a zero mean Gaussian "noise" matrix $N^{(t)}$.

3.3 Dynamic cluster pursuit

In this section we present three algorithms for the dynamic cluster extraction problem. Without loss of generality, we shall assume that the cluster of interest is $C_1^{(t)}$. At a high level, our algorithms work by approximating the *cluster change indicator vector*: $\mathbf{1}_{\delta C_1} = \mathbf{1}_{C_1^{(t)}} - \mathbf{1}_{C_1^{(t-1)}}$. If we denote $C_1^+ = C_1^{(t)} \setminus C_1^{(t-1)}$ and $C_1^- = C_1^{(t-1)} \setminus C_1^{(t)}$ then equivalently $\mathbf{1}_{\delta C_1} = \mathbf{1}_{C_1^+} - \mathbf{1}_{C_1^-}$. We recall the following theorem from §1.2.2:

Theorem 3.3.1. *Let $C_1, \ldots, C_k$ denote the connected components of a graph G. Then the cluster indicator vectors $\mathbf{1}_{C_1}, \ldots, \mathbf{1}_{C_k}$ form a basis for the kernel of L.*

As in the analysis of `SingleClusterPursuit`, we decompose each $G^{(t)}$ into an edge-disjoint union of subgraphs $G^{\mathrm{in},(t)}$ and $G^{\mathrm{out},(t)}$ where $V^{\mathrm{in},(t)} = V^{\mathrm{out},(t)} = V^{(t)}$ but $E^{\mathrm{in},(t)}$

contains only the edges in $E^{(t)}$ between vertices in the same cluster while $E^{\text{out},(t)}$ contains only edges between vertices in different clusters. We may further write $L^{(t)} = L^{\text{in},(t)} + M^{(t)}$. If G has well-defined clusters we may assume that $\|M^{(t)}\|_2$ is small relative to $\|L^{\text{in},(t)}\|_2$. As in the development of `SingleClusterPursuit`, we observe that:

Remark 3.3.2. If $C_1^{(t)}, \ldots, C_k^{(t)}$ are the clusters of $G^{(t)}$, then they are the connected components of $G^{\text{in},(t)}$. In particular $L^{\text{in},(t)}\mathbf{1}_{C_1^{(t)}} = 0$ for all t.

Now write $\mathbf{1}_{C_1^{(t)}} = \mathbf{1}_{C_1^{(t-1)}} + \mathbf{1}_{\delta C_1}$. From Remark 3.3.2 it follows that:

$$L^{\text{in},(t)}\left(\mathbf{1}_{C_1^{(t-1)}} + \mathbf{1}_{\delta C_1}\right) = 0 \Rightarrow L^{\text{in},(t)}\mathbf{1}_{\delta C_1} = -L^{\text{in},(t)}\mathbf{1}_{C_1^{(t-1)}} \tag{3.3}$$

For future use, define $\mathbf{y}^{\text{in},(t)} = -L^{\text{in},(t)}\mathbf{1}_{C_1^{(t-1)}}$ and and $\mathbf{y}^{(t)} := -L^{(t)}\mathbf{1}_{C_1^{(t-1)}} = \mathbf{y}^{\text{in},(t)} + \mathbf{e}^{(t)}$ where $\mathbf{e}^{(t)} := -M^{(t)}\mathbf{1}_{C_1^{(t-1)}}$. From (3.3) it follows that, in principle, one might recover $\mathbf{1}_{\delta C_1}$ by solving the linear system $L^{\text{in},(t)}\mathbf{x} = \mathbf{y}^{\text{in},(t)}$ but of course knowing $L^{\text{in},(t)}$ and $\mathbf{y}^{\text{in},(t)}$ presupposes that one knows $C_1^{(t)}, \ldots, C_k^{(t)}$, which is what we are trying to find. However, as was the case for `SingleClusterPursuit`, we shall show that $L^{(t)}\mathbf{x} = -\mathbf{y}^{(t)}$ is a sufficiently good proxy and all three variants of Dynamic Cluster Pursuit that we will present work by solving this linear system. The simplest of the three, `DCP-Thresh` approximates the signed support of $\mathbf{1}_{\delta C_1}$ by considering only the values of $\mathbf{y}^{(t)}$. `DCP-Thresh` is also the only variant for which we are currently able to provide theoretical guarantees (see Theorem 3.3.3) but numerical experiment reveals it to be the least robust to noise. On the other hand `DCP-OMP` is experimentally the most robust to noise, but requires an estimate of the number of vertices switching to and from C_1 at the t-th time step. The final variant, `DSCP-BP` does not require any user specified parameters, but is slower and less accurate than `DCP-OMP` (although still more accurate than `DSCP-Thresh`).

3.3.1 DCP-Thresh

Let us first analyze a deceptively simple approach to determining $\mathbf{1}_{\delta C_1}$. Observe that if $i \in C_1^{(t-1)}$ then:

$$y_i^{(t)} = -1 + \frac{1}{d_i^{(t)}} \sum_{j \in C_1^{(t-1)}} A_{ij}^{(t)} = -1 + \frac{1}{d_i^{(t)}} \left(\sum_{j \in C_1^{(t)}} A_{ij}^{(t)} - \sum_{j \in C_1^{+}} A_{ij}^{(t)} + \sum_{j \in C_1^{-}} A_{ij}^{(t)} \right)$$

if in fact $i \in C_1^{(t-1)} \cap C_1^{(t)}$ then $\sum_{j \in C_1^{(t)}} A_{ij}^{(t)} = d_i^{\text{in},(t)}$ and of course $\sum_{j \in C_1^{+}} A_{ij}^{(t)} \leq |C_1^+|$. Recalling the definition $r_i^{(t)} = d_i^{\text{out},(t)} / d_i^{\text{in},(t)} \geq d_i^{\text{out},(t)} / d_i^{(t)}$ we get that:

$$-r_i^{(t)} - \frac{|C_1^+|}{d_i^{(t)}} \leq y_i^{(t)} \leq 0 \tag{3.4}$$

(the upper bound follows trivially as $\sum_{j \in C_1^{(t-1)}} A_{ij}^{(t)} \leq d_i^{(t)}$).

On the other hand, if $i \in C_1^{(t-1)} \setminus C_1^{(t)} = C_1^-$ then $\sum_{j \in C_1^{(t)}} A_{ij}^{(t)} \leq d_i^{\text{out},(t)}$ and $\sum_{j \in C_1^{-}} A_{ij}^{(t)} \leq |C_1^-|$ so:

$$-1 \leq y_i^{(t)} \leq -1 + r_i^{(t)} + \frac{|C_1^-|}{d_i^{(t)}} \tag{3.5}$$

The point is that if $i \in C_1^-$ then $y_i^{(t)}$ is close to -1, whereas $y_i^{(t)}$ is close to 0 if there is no change in the cluster membership of i, that is $i \in C_1^{(t-1)} \cap C_1^{(t)}$. This holds assuming that $r_i^{(t)}$, $|C_1^+|$ and $|C_1^-|$ are sufficiently small. On the other hand, if $i \notin C_1^{(t-1)}$ then:

$$y_i^{(t)} = \frac{1}{d_i^{(t)}} \sum_{j \in C_1^{(t-1)}} A_{ij}^{(t)} = \frac{1}{d_i^{(t)}} \left(\sum_{j \in C_1^{(t)}} A_{ij}^{(t)} - \sum_{j \in C_1^{+}} A_{ij}^{(t)} + \sum_{j \in C_1^{-}} A_{ij}^{(t)} \right)$$

So a similar calculation for $i \notin C_1^{(t-1)}$ gives that:

$$0 \leq y_i^{(t)} \leq r_i^{(t)} + \frac{|C_1^-|}{d_i^{(t)}} \quad \text{if } i \notin C_1^{(t-1)} \text{ and } i \notin C_1^{(t)} \text{ (that is, } i \notin C_1^{(t-1)} \cup C_1^{(t)}) \tag{3.6}$$

$$1 - r_i^{(t)} - \frac{|C_1^+|}{d_i^{(t)}} \leq y_i^{(t)} \leq 1 \quad \text{if } i \in C_1^{(t)} \setminus C_1^{(t-1)} = C_1^+ \tag{3.7}$$

which means that again $y_i^{(t)} \approx 0$ if there is no change in the cluster membership of i whereas $y_i^{(t)} \approx 1$ if $i \in C_1^+$. This leads us to Algorithm 8.

Algorithm 8 `DSP-Thresh`

Input: Adjacency matrix $A^{(t)}$ and cluster $C^{(t-1)}$.
(1) Compute $L^{(t)}$ and $\mathbf{y}^{(t)} = -L^{(t)}\mathbf{1}_{C_1^{(t-1)}}$.
(2) Let $C^{\#,+} = \{i \in [n] : \ y_i^{(t)} > 0.5\}$ and $C^{\#,-} = \{i \in [n] : \ y_i^{(t)} < -0.5\}$
Output: $C^{\#,(t)} := \left(C^{(t-1)} \setminus C^{\#,-}\right) \cup C^{\#,+}$

Using a threshold value of 0.5 in Algorithm 8 is not particularly vital—any other value in $(0, 1)$ could be used.

3.3.2 Analysis of `DCP-Thresh`

The Thresholding rule in Step 2 of 8 will succeed as long as $y_i^{(t)} > 0.5 > y_j^{(t)}$ for all $i \in C_1^+$ and $j \in C_1^{(t)} \cap C_1^{(t-1)}$ and $y_i^{(t)} < -0.5 < y_j^{(t)}$ for all $i \in C_1^-$ and $j \notin C_1^{(t)} \cup C_1^{(t-1)}$. In simple cases one can guarantee that this is the case.

Theorem 3.3.3. *Let* $\mathbf{G} \sim$ *CDSBM*$(n, k, p, q, \epsilon, T)$ *with parameters* $p = \omega \log(n_0)/n_0$, $q = b \log(n)/n$, $\epsilon \leq \omega \log(n_0)/2n_0$ *and* k *constant. Here* ω *is any function such that* $\omega \to \infty$ *while* b *is a constant. Assuming* $C_1^{(t-1)}$ *is known, Algorithm 8 returns* $C_1^{\#,(t)} = C_1^{(t)}$, *for sufficiently large* n, *with probability* $1 - o(1)$.

Proof. From equations (3.4), (3.5), (3.6) and (3.7) as well as the discussion above, we see that the conditions for success are met if $r_i^{(t)} + |C_1^-|/d_i^{(t)} < 0.5$ and $r_i^{(t)} + |C_1^+|/d_i^{(t)} < 0.5$ for all $i \in [n]$. By construction, $G^{(t)} \sim \mathrm{SSBM}(n, k, p, q)$, hence by Corollary 2.2.6 of Chapter 2 $r_{\max}^{(t)} = o(1)$. By Corollary 2.2.5 of Chapter 2, $d_{\min}^{(t)} \geq d_{\min}^{\mathrm{in},(t)} \geq (1 - o(1))\omega \log(n_0)$ also with probability $1 - o(1)$. By our definition of the constrained dynamic stochastic block model, $|C_1^+| = |C_1^-| = \epsilon n_0$, thus:

$$\max\left\{r_i^{(t)} + |C_1^-|/d_i^{(t)}, r_i^{(t)} + |C_1^+|/d_i^{(t)}\right\} = r_{\max}^{(t)} + \frac{\epsilon n_0}{d_{\min}^{(t)}} = o(1) + (1 + o(1))\frac{\epsilon n_0}{\omega \log(n_0)}$$

We may take the $o(1)$ terms to be arbitrarily small, hence this quantity will be less than 0.5, for large enough n, if $\epsilon < \dfrac{\omega \log(n_0)}{2n_0}$. □

The important thing to take away from Theorem 3.3.3 is that Algorithm 8 will work only when ϵ is of the same order as p.

3.3.3 DSCP-OMP AND DSCP-BP

We now present a more refined approach to approximating $\mathbf{1}_{\delta C_1}$. As we are assuming not too many vertices change cluster at each time step, $\mathbf{1}_{\delta C_1}$ is *sparse*. With $\mathbf{y}^{\text{in},(t)} := -L^{\text{in},(t)}\mathbf{1}_{C_1^{(t-1)}}$ as before, we may set up two sparse recovery problems:

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \|\mathbf{x}\|_1 \text{ s.t. } L^{\text{in},(t)}\mathbf{x} = \mathbf{y}^{\text{in},(t)} \tag{3.8}$$

or, if an upper bound $\|\mathbf{1}_{\delta C_1}\|_0 \leq s$ is known:

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \|L^{\text{in},(t)}\mathbf{x} - \mathbf{y}^{\text{in},(t)}\|_2 \text{ s.t. } \|\mathbf{x}\|_0 \leq s \tag{3.9}$$

Of course in practice we do not have access to $L^{\text{in},(t)}$. But as in the development of `SingleClusterPursuit` this suggests that we might be able to get away with using $L^{(t)}$ and $\mathbf{y}^{(t)}$ in place of $L^{\text{in},(t)}$ and $\mathbf{y}^{\text{in},(t)}$. That is, we consider the sparse approximation problems:

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \|\mathbf{x}\|_1 \text{ s.t. } \|L^{(t)}\mathbf{x} - \mathbf{y}^{(t)}\|_2 \leq \epsilon \tag{3.10}$$

and

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \|L^{(t)}\mathbf{x} - \mathbf{y}^{(t)}\|_2 \text{ s.t. } \|\mathbf{x}\|_0 \leq s \tag{3.11}$$

Thus we propose the following approach to dynamic cluster extraction:

1. Solve Problem (3.10) using Basis Pursuit or (3.11) using `OMP` to obtain $\mathbf{x}^{\#}$.
2. Let $C^{\#,+} = \{i \in [n] : \ x_i^{\#} > 0\}$ and $C^{\#,-} = \{i \in [n] : \ x_i^{\#} < 0\}$
3. Return $C^{\#,(t)} := \left(C^{(t-1)} \setminus C^{\#,-}\right) \cup C^{\#,+}$

Step 2 above is not particularly robust. In particular, if we do not know exactly the sparsity of $\mathbf{1}_{\delta C_1}$ (which is almost certainly the case in real life), then taking s to be an overestimate of $\|\mathbf{1}_{\delta C_1}\|_0$ we expect $\mathbf{x}^{\#}$ to have "noise" components that are close to zero, and should be ignored. This suggests that one should pick a threshold $\rho \in (0, 1)$ and replace Step 2 with the rule:

$$C^{\#,+} = \{i \in [n] : \ x_i^{\#} > \rho\} \text{ and } C^{\#,-} = \{i \in [n] : \ x_i^{\#} < -\rho\} \tag{3.12}$$

When OMP is used in Step 1 we have had more success in practice letting $S = \text{supp}(\mathbf{x}^{\#})$, then solving the following (box-constrained) integer least squares problem:

$$\mathbf{z}^{\#} = \underset{\mathbf{x}\in\{-1,0,1\}^s}{\text{argmin}} \|L_S^{(t)}\mathbf{x} - \mathbf{y}^{(t)}\|_2 \tag{3.13}$$

and then taking $C^{\#,+} = \{i \in [n] : \ z_i^{\#} = 1\}$ and $C^{\#,-} = \{i \in [n] : \ z_i^{\#} = -1\}$. We formalize this discussion into two algorithms: Dynamic Cluster Pursuit with OMP (DCP-OMP) and Dynamic Cluster Pursuit with Basis Pursuit (DCP-BP). Note that in the pseudocode for DCP-OMP, one should use either Step (3.a) or Step (3.b).

Algorithm 9 DCP-OMP

Input: Adjacency matrix $A^{(t)}$ and cluster $C^{(t-1)}$. Parameter s and possibly parameter ρ.
(1) Compute $L^{(t)}$ and $\mathbf{y}^{(t)}$.
(2) Solve (3.11) using OMP to obtain $\mathbf{x}^{\#}$.
(3.a) Solve (3.13) to obtain $\mathbf{z}^{\#}$ and let $C^{\#,+} = \{i \in [n] : \ z_i^{\#} = 1\}$ and $C^{\#,-} = \{i \in [n] : z_i^{\#} = -1\}$.
(3.b) Let $C^{\#,+} = \{i \in [n] : \ x_i^{\#} > \rho\}$ and $C^{\#,-} = \{i \in [n] : \ x_i^{\#} < -\rho\}$
Output: $C^{\#,(t)} := \left(C^{(t-1)} \setminus C^{\#,-}\right) \cup C^{\#,+}$

Algorithm 10 DCP-BP

Input: Adjacency matrix $A^{(t)}$ and cluster $C^{(t-1)}$. Noise parameter η and threshold parameter $\rho \in (0, 1)$.
Step 1 Compute $L^{(t)}$ and $\mathbf{y}^{(t)}$.
Step 2 Solve (3.10) using Basis Pursuit to obtain $\mathbf{x}^{\#}$.
Step 3 Let $C^{\#,+} = \{i \in [n] : \ x_i^{\#} > \rho\}$ and $C^{\#,-} = \{i \in [n] : \ x_i^{\#} < -\rho\}$
Output: $C^{\#,(t)} := \left(C^{(t-1)} \setminus C^{\#,-}\right) \cup C^{\#,+}$

The theoretical analysis of Algorithms 9 and 10 is significantly more complicated than the analysis of Algorithm 8. Essentially, the noise term $\mathbf{e}^{(t)}$ is too large in the ℓ_2 norm to use deterministic Compressive Sensing performance guarantees, and not Gaussian enough to use probabilistic ones. However, both algorithms work excellently in the experimental setting, as demonstrated in §3.4. We leave their analysis to future work.

3.4 Experimental Results

3.4.1 Implementation of Algorithms

All algorithms were implemented in MATLAB.

`DCP-OMP` was implemented as described in Algorithm 9. We use a standard implementation of OMP, available at: `https://www.mathworks.com/matlabcentral/fileexchange/32402-cosamp-and-omp-for-sparse-recovery`. For smaller problems (that is, problems with $\epsilon n_0 \leq 100$) we use using the MATLAB package MILES, available at `https://www.cs.mcgill.ca/~chang/MILES.php` and see also [21], to solve the integer least squares problem (3.13). For larger problems we used the thresholding approach, with $\rho = 0.5$. The value of the parameter s is case-specific.

`DCP-BP` was implemented as described in Algorithm 10. We solved Problem (3.10) using the ℓ_1 minimization algorithm of [56]. Again, we take $\rho = 0.5$.

`AFFECT` was implemented using the `AFFECT` MATLAB toolbox, available at: `https://herogroup.engin.umich.edu/affect-matlab-toolbox`. There are no parameters for this algorithm.

`EvolSpec-PCQ` and `EvolSpec-PCM` were implemented by the author, as described in [22], and see also §3.1.2. For both algorithms we took the parameter $\alpha = 0.5$.

3.4.2 Synthetic Data Sets

We first consider graphs drawn from the symmetric dynamic stochastic block model, SDSBM$(n, k, p, q, \epsilon, T)$. We fix $k = 10$ and $T = 10$ but vary the parameters n, p, q and ϵ to exhibit different phenomena.

	Experiment 1	Experiment 2	Experiment 3	Experiment 4
DCP-OMP	2.61	1.78	10.53	2.61
DCP-Thresh	0.47	0.13	0.45	0.45
AFFECT	10.10	3.22	10.16	11.47
Spec-PCQ	16.41	2.73	16.29	15.73
Spec-PCM	13.16	2.42	12.86	12.27

Table 3.1: Run time for the various dynamic clustering algorithms, in seconds. Note that this is total run time, *i.e.* time taken to process all ten snapshots.

Experiment 1. We set $\epsilon = 0.05$, $p = 2\log^2(n)/n$ and $q = \log(n)/n$. The values of p and q are as in Experiment 1 on synthetic data in Chapter 2 (see §2.6.3).

Experiment 2. Here we increase ϵ to 0.2 and increase p slightly to $4\log^2(n)/n$. We keep $q = \log(n)/n$ and all other parameters unchanged.

Experiment 3. We set $n = 2000$, keep p and q as in Experiment 1 but let $\epsilon = 0.25$.

Experiment 4. For the final experiment we keep $n = 2000$ $k = T = 10$ and set $\epsilon = 0.05$. However, we set $p = 20\log(n)/n$ and $q = 2\log(n)/n$ which is below the threshold given in Theorem 1.3.4, thus we could not expect any static clustering algorithm to perform well on the snapshots $G^{(t)}$.

The Jaccard Indices for the various clusters found are displayed in Figure 3.1. The run times are displayed in Table 3.1. Note that we did not use DCP-BP for this experiment as we found it to be too slow.

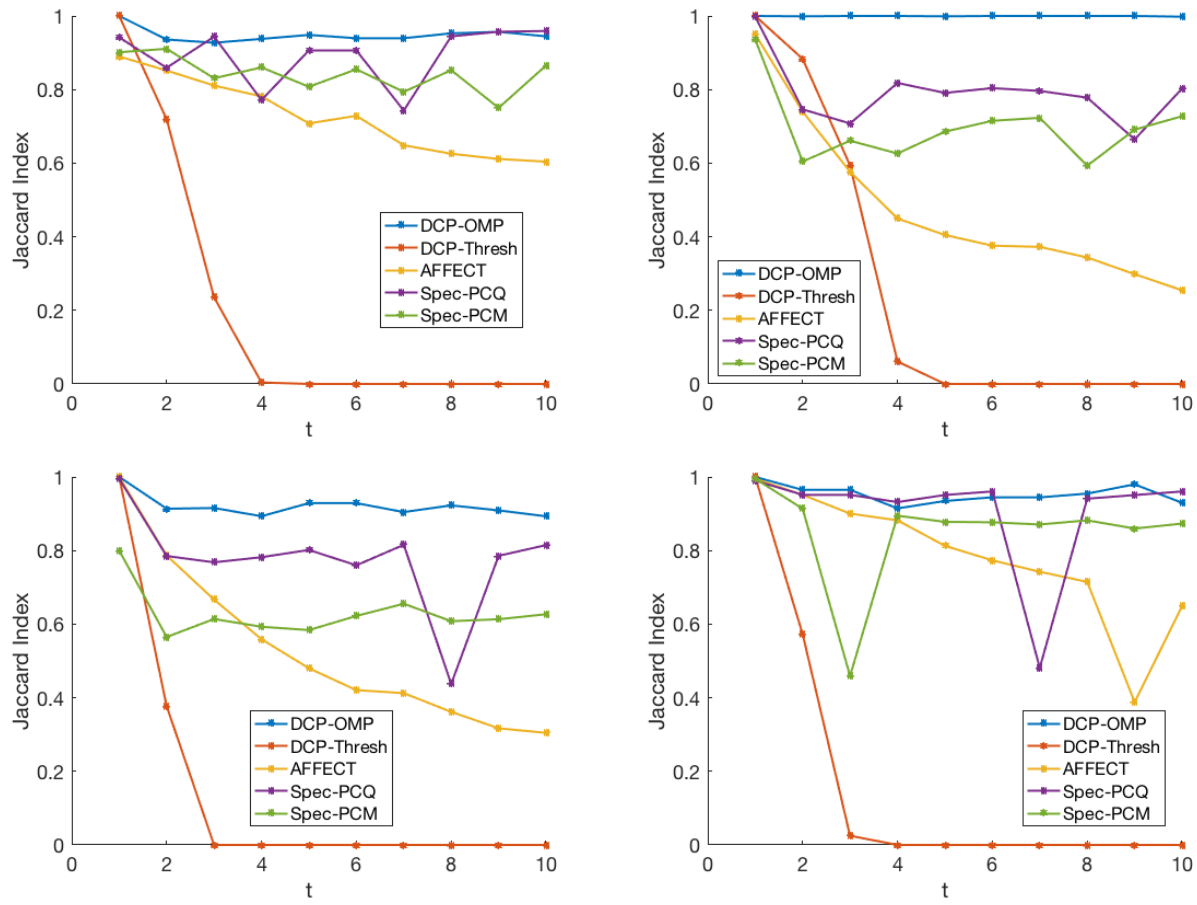

Figure 3.1: Jaccard indices for the various cluster extraction algorithms, for each snapshot. Averaged over ten independent trials.

3.4.3 Real Data Sets

We construct a new data set based on U.S senate voting records. We use data available at `https://voteview.com`, and our study is inspired by similar constructions using the same raw data in [69] and [88], although our approach is distinct from theirs.

We consider U.S. senate voting records from the 87th Congress (seated on January 3rd 1961) to the 115th Congress (which ran until January 2nd 2019). In the U.S. senate, each state is assigned two seats, which are filled by the winner of a state-wide election held once every six years. We assign one vertex to each seat, for a total of 100 vertices (note that the 87th Congress was the first full congress with 50 states, which explains our choice of time window). For each congress we construct a snapshot as follows. We first construct an auxiliary matrix $B^{(t)} \in \mathbb{R}^{100\times 100}$ which records the number of times two given senators voted the same way. More precisely:

$$B^{(t)}_{ij} = \text{number of votes on which Senators } i \text{ and } j \text{ both voted "yea" or both voted "nay"}$$

For any $i \in [100]$ define:

$$\mathrm{NN}(i, K) := \{j \in [n] : \ B_{ij} \text{ amongst } K\text{-largest entries in } i\text{-th row of } B\}$$

We then construct $A^{(t)}$ as the adjacency matrix of a symmetric unweighted K-nearest neighbors graph:

$$A^{(t)}_{ij} = \begin{cases} 1 & \text{if } j \in \mathrm{NN}(i, K) \text{ or } i \in \mathrm{NN}(j, K) \\ 0 & \end{cases}$$

We emphasize again that each vertex in $\mathbf{G}$ is associated with a senate seat, not a senator. Thus the individual associated to a vertex may change between snapshots. In particular, we are interested in tracking the *change in political affiliation* of senate seats over time. For example, in the 87th congress Southern senate seats (*i.e.* Georgia, Alabama, Mississippi and so on) were solidly Democratic but by the 115th congress were in Republican hands. Thus,

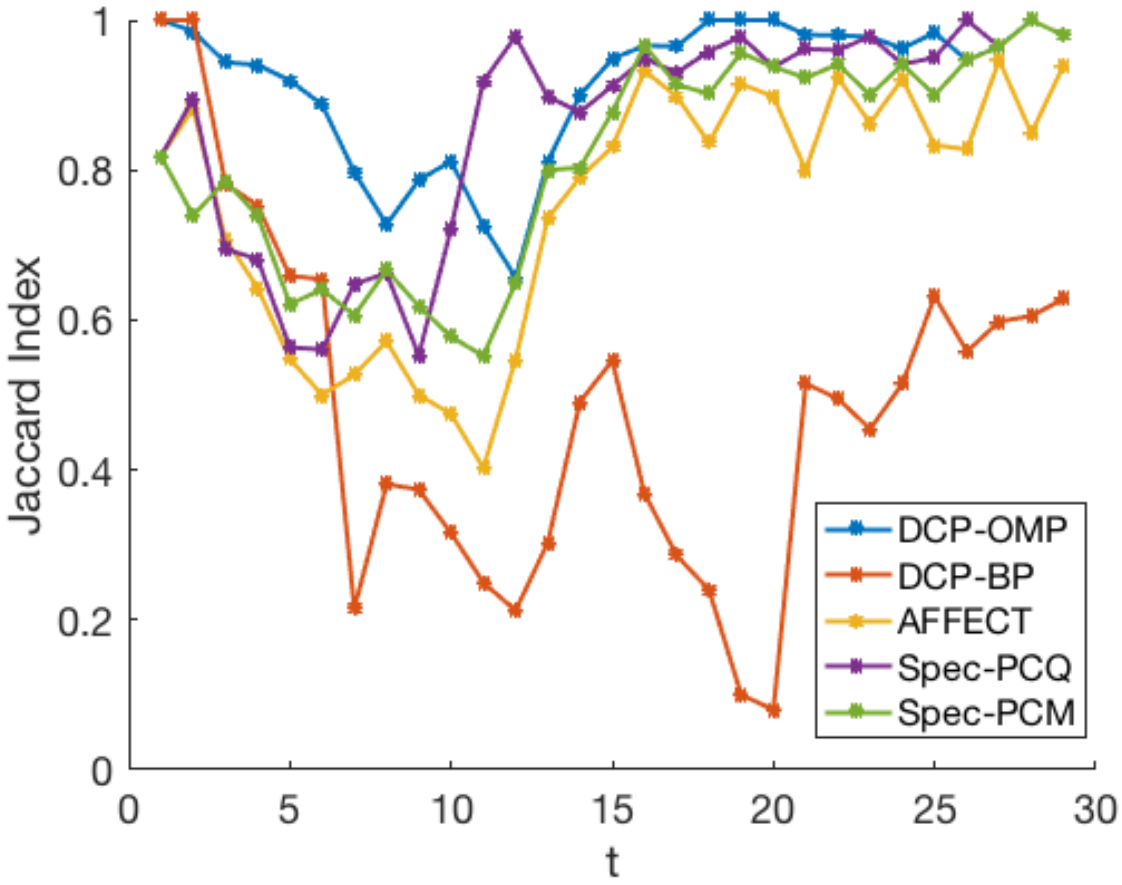

Figure 3.2: Results of Dynamic Clustering on the "Senate" Data set

we specify as initial cluster of interest, $C_1^{(1)}$, the set of all senate seats held by Democratic senators, and track the evolution of this cluster. The results are shown in Figure 3.2.

3.4.4 Analysis

As can be seen from Figures 3.1 and 3.2, `DSCP-OMP` outperforms the other baseline methods almost all of the time. We remark that it runs significantly faster too, particularly when the number of clusters is larger than 2 (for example in the synthetic experiments). In Figure 3.2 the decline in accuracy from the 87th congress to the 97th congress is largely due to

a realigning of ideologies between the two major American political parties thus we argue that this is more of a feature than a bug of `DCP-OMP`. In particular, the clusters returned by `DSCP-OMP` in the 87th-97th congress excludes Southern Democrats, who voted against their caucus on bills such as the Civil Rights Act of 1964 (88th Congress), and includes Republicans who voted with the Democratic majority for such bills. Thus one can argue that the cluster tracked by `DSCP-OMP` is more coherent ideologically than the cluster given by party affiliation, which we are taking as ground-truth.

Chapter 4

Shortest path distances for clustering Euclidean data

In this chapter, we return to the problem of turning a data set in Euclidean space into a graph that was first encountered in §2.6.5. Recall that there the goal was to turn $\mathcal{X} \subset \mathbb{R}^D$ into a graph G so that a graph-based clustering algorithm such as `SingleClusterPursuit` could be used to divide $\mathcal{X}$ into clusters $\mathcal{X}_1, \ldots, \mathcal{X}_k$. Intuitively, the data points in the same $\mathcal{X}_a$ should be more "similar" than data points in different clusters. Clearly, the notion of similarity is context dependent. The approach taken in §2.6.5, which is typical of such problems, was to construct the (weighted) adjacency matrix of G as $A_{ij} = \varphi(\|\mathbf{x}_i - \mathbf{x}_j\|)$ where φ is a kernel function and $\|\cdot\|$ represents the Euclidean metric. Here we consider replacing $\|\cdot\|$ with a more general distance function, $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}_{\geq 0}$. Ideally, one would choose d such that points in the same cluster are close with respect to d, while points in different clusters remain distant. Thus, the choice of distance function should reflect, in some way, our assumptions about the data $\mathcal{X}$ and the notion of similarity we would like the clusters to capture.

A common assumption, frequently referred to as the *manifold hypothesis* (see, for example, [36]) posits that each $\mathcal{X}_a$ is sampled according to some probability distribution μ_a supported on a latent data manifold $\mathcal{M}_a$. It is usually assumed that the dimension of each $\mathcal{M}_a$ is much lower than the ambient dimension D. Although it can be shown that taking d to be the Euclidean distance can be successful [9] for such data, *data-driven* distance functions have been increasingly favored [30, 13, 23, 60].

Here we consider taking d to be a *power weighted shortest path distance*. We shall show, theoretically and experimentally, that such distances provide superior performance for spectral clustering, assuming the manifold hypothesis. We also provide a novel fast algorithm (a variation of Dijkstra's shortest path algorithm) that finds the K nearest neighbors of any $\mathbf{x}_i \in \mathcal{X}$, with respect to the power weighted shortest path distance, in $O(K^2 \log(n))$ time. This represents a significant improvement over existing approaches, and makes the use of power weighted shortest path distances practical for large data sets. This chapter represents joint work with Steve Damelin.

The rest of the chapter is laid out as follows. In §4.1 we define p weighted shortest path distances (p-WSPD's), and show that they can be thought of as interpolants between the Euclidean distance and the Longest Leg Path Distance (LLPD). In §4.2 we survey the existing literature on p-WSPD's, and path-based distances in general. In §**??** In §4.3 we connect p-WSPD's with the manifold hypothesis, and show that if $\mathcal{X}_a \subset \mathcal{M}_a$ for $a = 1, \ldots, k$ where the $\mathcal{M}_a$ are low dimensional manifolds then the pairwise distance between any two points in the same cluster goes to zero, while the distance between points in different clusters remains bounded, as the number of data points tends to infinity. §4.4 discusses the problem of determining, for any $\mathbf{x}_i \in \mathcal{X}$, its K nearest neighbors in a p-WSPD and also presents our new algorithm, while in §4.5 we present some numerical results. As the focus is on demonstrating that using p-WSPD's provide higher classification accuracy than the Euclidean metric ($\|\cdot\|$), we stick mainly to one graph clustering algorithm, namely spectral clustering, and compare the performance of spectral clustering with $\|\cdot\|$ to that of spectral clustering with a p-WSPD for a variety of power weightings p although in §4.5.3 we experiment with combining p-WSPD's with `SingleClusterPursuit`.

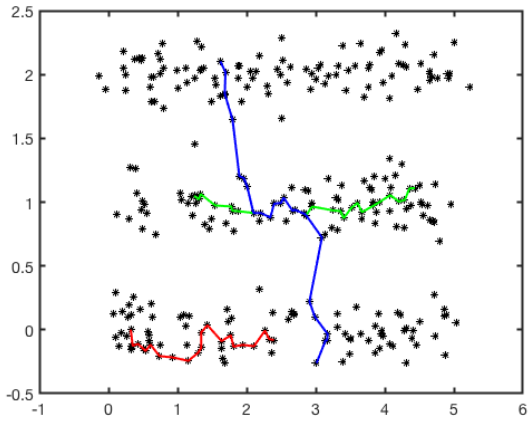

Figure 4.1: Three sample geodesics in the power weighted shortest path distance with $p = 2$, for the data set 'Three Lines' (see §4.5). Observe how the geodesics consist of many small hops, instead of several large hops. The total lengths of the red and green paths are significantly smaller than the length of the blue path.

4.1 Power Weighted Shortest Path Distances

For any distinct pair $\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}$ let $\gamma = \mathbf{x}_\alpha \to \mathbf{x}_{i_1} \to \ldots \to \mathbf{x}_{i_m} \to \mathbf{x}_\beta$ where $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_m} \in \mathcal{X}$ denote a path from $\mathbf{x}_\alpha$ to $\mathbf{x}_\beta$ through $\mathcal{X}$. We shall refer to the subpaths $\mathbf{x}_{i_j} \to \mathbf{x}_{i_{j+1}}$ as "legs" or "hops" of γ. Occasionally we shall use the shorthand $\gamma = \mathbf{x}_\alpha \rightsquigarrow \mathbf{x}_\beta$. By convention we shall declare $\mathbf{x}_{i_0} = \mathbf{x}_\alpha$ and $\mathbf{x}_{i_{m+1}} = \mathbf{x}_\beta$ For any $p \geq 1$ define the p-weighted length of γ to be:

$$L^{(p)}(\gamma) := \left(\sum_{j=0}^{m} \|\mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}\|^p \right)^{1/p} \tag{4.1}$$

We define the *p-weighted path distance* from $\mathbf{x}_\alpha$ to $\mathbf{x}_\beta$ through $\mathcal{X}$ to be the minimum length over all such paths:

$$d_{\mathcal{X}}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) := \min \left\{ L^{(p)}(\gamma) : \ \gamma \text{ a path from } \mathbf{x}_\alpha \text{ to } \mathbf{x}_\beta \text{ through } \mathcal{X} \right\} \tag{4.2}$$

If γ^* is the path achieving the minimum in Equation (4.2) we shall call it a *geodesic*. We do not place any upper bound on the number of vertices involved in paths under consideration—in principle the minimizing path could traverse all other $n - 2$ data points in $\mathcal{X}$. Note that

$d_{\mathcal{X}}^{(p)}$ will depend on the power weighting p and the data set $\mathcal{X}$. As several authors [50, 23, 7] have noted, the distance $d_{\mathcal{X}}^{(p)}$ is *density-dependent*, so that if $\mathbf{x}_\alpha$ and $\mathbf{x}_\beta$ are contained in a region of high density (*i.e.* a cluster) the path distance $d^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$ will likely be shorter than the Euclidean distance $\|\mathbf{x}_\alpha - \mathbf{x}_\beta\|$ (as long as $p > 1$).

It can be useful to think of γ as a path in the complete graph on n vertices with edge weights $A_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^p$. We shall refer to such a graph as a (power-weighted) *distance graph* of $\mathcal{X}$.

4.1.1 Longest-Leg Path Distance

Another common path-based distance, analyzed in connection with spectral clustering in [38, 21, 60], is the longest-leg path distance (LLPD), which we shall denote as $d_{\mathcal{X}}^{(\infty)}$ (the choice of this notation should become clear shortly). It is defined as the minimum, over all paths from $\mathbf{x}_i$ to $\mathbf{x}_j$ through $\mathcal{X}$, of the maximum distance between consecutive points in the path (*i.e.* legs). Before formally defining $d_{\mathcal{X}}^{(\infty)}$, define, for any path γ from $\mathbf{x}_\alpha$ to $\mathbf{x}_\beta$ through $\mathcal{X}$, *the longest-leg length of* γ as:

$$L^{(\infty)}(\gamma) = \max_{j=0,\ldots,m} \|\mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}\|$$

again we are using the convention that $\mathbf{x}_{i_0} = \mathbf{x}_\alpha$ and $\mathbf{x}_{i_{m+1}} = \mathbf{x}_\beta$. Now, in analogy with (4.2):

$$d_{\mathcal{X}}^{(\infty)}(\mathbf{x}_i, \mathbf{x}_j) = \min\left\{L^{(\infty)}(\gamma) : \gamma \text{ a path from } \mathbf{x}_\alpha \text{ to } \mathbf{x}_\beta \text{ through } \mathcal{X}\right\} \tag{4.3}$$

4.1.2 Relationship between these distances

Lemma 4.1.1. *For any fixed $\mathcal{X}$, we have that $\lim_{p\to\infty} d^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) = d^{(\infty)}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$ for all $\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}$.*

Proof. First observe that for any fixed path γ from $\mathbf{x}_\alpha$ to $\mathbf{x}_\beta$ we have that $\lim_{p\to\infty} L^{(p)}(\gamma) = L^{(\infty)}(\gamma)$. To see this, let us suppose that $\|\mathbf{x}_{i_{j^*}} - \mathbf{x}_{i_{j^*+1}}\| = \max_{j=0,\ldots,m} \|\mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}\|$. That is,

$\mathbf{x}_{i_{j^*}} \to \mathbf{x}_{i_{j^*+1}}$ is the longest leg in γ. Then for any p:

$$L^{(p)}(\gamma) := \left(\sum_{j=0}^{m} \|\mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}\|^p\right)^{1/p} = \|\mathbf{x}_{i_{j^*}} - \mathbf{x}_{i_{j^*+1}}\| \left(1 + \sum_{\substack{j=0 \\ j\neq j^*}}^{m} \left(\frac{\|\mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}\|}{\|\mathbf{x}_{i_{j^*}} - \mathbf{x}_{i_{j^*+1}}\|}\right)^p\right)^{1/p}$$
$$\leq \|\mathbf{x}_{i_{j^*}} - \mathbf{x}_{i_{j^*+1}}\| \left(m^{1/p}\right)$$

and as $m^{1/p} \to 1$, $L^{(p)}(\gamma) \to \|\mathbf{x}_{i_{j^*}} - \mathbf{x}_{i_{j^*+1}}\| = \max_{j=0,\ldots,m} \|\mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}\| = L^{(\infty)}(\gamma)$. Because the operation of taking a minimum is continuous, we get that:

$$\lim_{p\to\infty} d^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) = \lim_{p\to\infty} \min_\gamma \left\{L^{(p)}(\gamma)\right\} = \min_\gamma \left\{\lim_{p\to\infty} L^{(p)}(\gamma)\right\} = \min_\gamma \left\{L^{(\infty)}(\gamma)\right\} = d^{(\infty)}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$$

□

Lemma 4.1.2. *For all* $\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}$, $d^{(1)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) = \|\mathbf{x}_\alpha - \mathbf{x}_\beta\|$

Proof. $d^{(1)}$ is defined as a minimum over all paths from $\mathbf{x}_\alpha$ to $\mathbf{x}_\beta$ through $\mathcal{X}$, and in particular the one hop path $\gamma_{\alpha\to\beta} = \mathbf{x}_\alpha \to \mathbf{x}_\beta$ is such a path. We claim it is the shortest such path as for any other path $\gamma = \mathbf{x}_\alpha \to \mathbf{x}_{i_1} \to \ldots \to \mathbf{x}_{i_m} \to \mathbf{x}_\beta$ by repeated applications of the triangle inequality:

$$L^{(1)}(\gamma_{\alpha\to\beta}) = \|\mathbf{x}_\alpha - \mathbf{x}_\beta\| = \|\mathbf{x}_\alpha - \sum_{j=1}^{m}(\mathbf{x}_{i_j} - \mathbf{x}_{i_j}) - \mathbf{x}_\beta\|$$
$$\leq \sum_{j=0}^{m} \|\mathbf{x}_{i_j} - \mathbf{x}_{i_{j+1}}\| = L^{(1)}(\gamma)$$

□

Hence the $d_{\mathcal{X}}^{(p)}$ can be thought of as interpolating between Euclidean distance and longest-leg path distance.

4.1.3 Computational Issues

The use of shortest path type distances for clustering has typically been hindered by the high computational cost of computing shortest paths. Indeed, computing the shortest

path distance between all pairs $\mathbf{x}_\alpha, \mathbf{x}_\beta$ using Dijkstra's algorithm naively requires $O(n^3)$ operations—clearly infeasible for large data sets. But, when one is using shortest paths in conjunction with clustering this is inefficient, as we typically only use the distances between $\mathbf{x}_\alpha$ and its *K Nearest Neighbors*. In §4.4 we introduce a modified version of Dijkstra's algorithm (Algorithm 12) that computes the K-nearest-neighbors of a given vertex in any p-WSPD in $O(K^2 \log(n))$ operations, assuming that nearest neighbors queries, with respect to the Euclidean distance, can be done efficiently which is the case when the $\mathcal{X}_a$ are sampled from low dimensional manifolds. It follows that one can find the K nearest neighbors of all $\mathbf{x}_\alpha \in \mathcal{X}$ in $O(K^2 n \log(n))$ operations, a more reasonable time.

4.2 Prior Work

The idea of using p-WSPD's for clustering was proposed in [86], and further explored in [74]. More generally, shortest path distances are a core part of ISOMAP [82], although we emphasize that here not all paths through $\mathcal{X}$ are considered—first a K-Nearest Neighbors graph $\mathcal{G}^{(K)}$ is computed from $\mathcal{X}$ and only paths in this graph are admissible. We also mention [7] for some cautionary results on using the shortest path distance on the *unweighted* K-NN graph when the density with respect to which points are sampled is non-uniform. Finally, several recent papers consider the use of LLPD for clustering, for example [60].

The works most similar to ours are [13] and [23]. In [13] p-WSPD's are proposed for semi-supervised learning, and a fast Dijkstra-style algorithm is provided. However their set-up is fundamentally different to ours, as their focus is on finding, for every $\mathbf{x} \in \mathcal{X}$, its nearest neighbor, in a p-WSPD, in some set of labeled data points $\mathcal{L}$. Moreover, they only consider semi-supervised methods, specifically nearest-neighbor classification, and do not provide any quantitative results on the asymptotic behaviour of the lengths of power weighted shortest paths. In [23] the p-WSPD with $p = 2$ is studied, and some interesting

connections between this distance and the nearest-neighbor geodesic distance are discovered. However, the applications of this distance to clustering is not explicitly explored.

On the computational side, we are unaware of any prior mention of Algorithm 12 in the literature, although similar algorithms, which solve slightly different problems, are presented in [44], [66] and [13]. As mentioned above, the algorithm presented in [13] is concerned with finding, for all $\mathbf{x} \in \mathcal{X}$, the nearest neighbor of $\mathbf{x}$ with respect to a p-WSPD in a set of labeled data $\mathcal{L}$ and has run time $O(n\log(n))$. It does not seem possible to extend it to finding K path nearest neighbors. The algorithm of [44] is formulated for any graph $G = (V, E)$ (*i.e.* not just graphs obtained from data sets $\mathcal{X} \subset \mathbb{R}^D$) and as such is not well-adapted to the problem at hand. In particular, it has run time $O(K(n\log(n) + |E|))$. Because the distance graph obtained from $\mathcal{X}$ is implicitly complete, $|E| = O(n^2)$ and this results in a run time proportional to Kn^2, which is infeasible for large data sets. Finally, the algorithm presented in [66], although adapted to the situation of distance graphs of data sets, actually solves a slightly different problem. Specifically they consider finding the K_1 path nearest neighbors of each $\mathbf{x} \in \mathcal{X}$ *in a* K_2 *Euclidean nearest neighbors graph* of $\mathcal{X}$. As such, it is not clear whether the set of path nearest neighbors produced by their algorithm are truly the p-WSPD nearest neighbors in $\mathcal{X}$. Finally we mention that our approach is "one at a time", whereas the other three algorithms mentioned are "all at once". That is, our algorithm, given $\mathbf{x} \in \mathcal{X}$, finds the K p-WSPD nearest neighbors $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathcal{X}$. This can then be iterated to find the p-WSPD nearest neighbors of all points in $\mathcal{X}$. In contrast, "all at once" algorithms directly return the sets of K nearest neighbors for each $\mathbf{x} \in \mathcal{X}$. Thus it is possible our algorithm will have applications in other scenarios where the p-WSPD nearest neighbors of only some small subset of points of $\mathcal{X}$ are required.

4.3 Analysis of shortest path distances in the multi-manifold setting

One of the most useful aspects of p-WSPD's, when applied to clustering problems, is that they tend to "squeeze" points in the same cluster together, while (hopefully) keeping points in different clusters separated. Here we make this more precise. Specifically we show that for any $p > 1$ if the data comes from the model described in §4.3.1 then:

- $\min_{\mathbf{x}_\alpha \in \mathcal{X}_a, \mathbf{x}_\beta \in \mathcal{X}_b} d_{\mathcal{X}}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \geq \delta > 0$ (see Lemma 4.3.1)
- $\max_{a \in [k]} \max_{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a} d_{\mathcal{X}}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \to 0$ (see Theorem 4.3.5). In fact, we can specify the rate at which this quantity goes to zero.

Note that in this section it is sometimes necessary to enlarge our definition of p-WSPD to allow for paths between $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$ that are not necessarily in $\mathcal{X}$ (and points that are not in $\mathcal{X}$ shall be denoted without a subscript). Thus $d_{\mathcal{X}}^{(p)}(\mathbf{x}, \mathbf{y})$ is technically defined as, using the notation of §4.1, $d_{\mathcal{X} \cup \{\mathbf{x}, \mathbf{y}\}}^{(p)}(\mathbf{x}, \mathbf{y})$.

4.3.1 Data Model

Let us be more precise about the sorts of data sets we are considering. We shall assume that $\mathcal{X}$ can be partitioned into k clusters $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \ldots \cup \mathcal{X}_k \subset \mathbb{R}^D$ which are *a priori* unknown. We posit that for each $\mathcal{X}_a$ there is a smooth, compact, embedded manifold $\mathcal{M}_a \hookrightarrow \mathbb{R}^D$ and further that $\mathcal{X}_a$ is sampled from $\mathcal{M}_a$ according to a probability density function μ_a supported on $\mathcal{M}_a$. We assume that $\mu_a^{\min} := \min_{x \in \mathcal{M}_a} \mu_a(x) > 0$. We further assume that the data manifolds are fairly well separated, that is:

$$\operatorname{dist}(\mathcal{M}_a, \mathcal{M}_b) = \min_{\mathbf{x} \in \mathcal{M}_a, \mathbf{y} \in \mathcal{M}_b} \|\mathbf{x} - \mathbf{y}\| \geq \delta > 0 \text{ for all, } 1 \leq a < b \leq k \tag{4.4}$$

For future use let us gather some results and notation relating to our data model. Let $|\mathcal{X}_a| = n_a$ and let g_a denote the restriction of the Euclidean metric to $\mathcal{M}_a$. Note that $(\mathcal{M}_a, g_a)$ is now a compact Riemannian manifold. For any $\mathbf{x}, \mathbf{y} \in \mathcal{M}_a$ let

$$\operatorname{dist}_a(\mathbf{x}, \mathbf{y}) := \inf_{\eta} \int_0^1 \sqrt{g_a(\eta'(t), \eta'(t))} dt$$

denote the metric induced by g_a, where the infimum is over all piecewise smooth curves $\eta : [0,1] \to \mathcal{M}_a$ with $\eta(0) = \mathbf{x}$ and $\eta(1) = \mathbf{y}$. Define the *diameter* of $\mathcal{M}_a$ to be the supremum over all distances between points in $\mathcal{M}_a$:

$$\operatorname{diam}(\mathcal{M}_a) := \sup_{\mathbf{x},\mathbf{y} \in \mathcal{M}_a} \operatorname{dist}_a(\mathbf{x}, \mathbf{y})$$

Since each $\mathcal{M}_a$ is compact this supremum is in fact a maximum and $\operatorname{diam}(\mathcal{M}_a)$ is finite.

It is sometimes of interest to extend this model such that the density functions μ_a are supported "near", but not necessarily on, the $\mathcal{M}_a$. In [9] for example, this is done by allowing μ_a to be supported on a tubular neighborhood of $\mathcal{M}_a$, defined as:

$$B(\mathcal{M}_a, \tau) = \left\{ \mathbf{x} \in \mathbb{R}^D \ : \ \min_{\mathbf{y} \in \mathcal{M}_a} \|\mathbf{x} - \mathbf{y}\|_2 \leq \tau \right\}.$$

for some parameter $\tau > 0$. We leave the extension of our results to such models to future work.

4.3.2 Paths between points in different clusters

Here we prove that power weighted path distances maintain a separation between points in different clusters.

Lemma 4.3.1. *Let ϵ_2 denote the minimal distance between points in different clusters. That is:*

$$\epsilon_2 := \min_{\substack{a,b \in [k] \\ a \neq b}} \min_{\substack{\mathbf{x}_\alpha \in \mathcal{X}_a \\ \mathbf{x}_\beta \in \mathcal{X}_b}} d_{\mathcal{X}}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$$

Then $\epsilon_2 \geq \delta$ with δ as defined in (4.4)*.*

Proof. For any $\mathbf{x}_\alpha \in \mathcal{X}_a$ and $\mathbf{x}_\beta \in \mathcal{X}_b$ let $\gamma = \{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_m}\}$ be any path from $\mathbf{x}_\alpha$ to $\mathbf{x}_\beta$ through $\mathcal{X}$, where again we are using the convention that $\mathbf{x}_{i_0} := \mathbf{x}_\alpha$ and $\mathbf{x}_{i_{m+1}} = \mathbf{x}_\beta$. If $\mathbf{x}_\alpha \in \mathcal{X}_a$ and $\mathbf{x}_\beta \in \mathcal{X}_b$ there must exist (at least one) $j^* \in [m]$ such that $\mathbf{x}_{i_{j^*}} \in \mathcal{X}_a$ while $\mathbf{x}_{i_{j^*+1}} \in \mathcal{X}_b$. By the assumptions on the generative model, $\mathcal{X}_a \subset \mathcal{M}_a$ and $\mathcal{X}_b \subset \mathcal{M}_b$ and so:

$$\|\mathbf{x}_{i_{j^*+1}} - \mathbf{x}_{i_{j^*}}\|^p \geq (\mathrm{dist}(\mathcal{M}_a, \mathcal{M}_b))^p = \delta^p$$

thus:

$$L^{(p)}(\gamma) := \left(\sum_{j=0}^{m} \|\mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}\|^p\right)^{1/p} \geq \left(\|\mathbf{x}_{i_{j^*+1}} - \mathbf{x}_{i_{j^*}}\|^p\right)^{1/p} \geq \delta.$$

Because this holds for all such γ:

$$d_{\mathcal{X}}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) := \min_{\gamma} \left\{L^{(p)}(\gamma)\right\} \geq \delta$$

and because this holds for all such $\mathbf{x}_\alpha$ and $\mathbf{x}_\beta$:

$$\min_{\mathbf{x}_\alpha \in \mathcal{X}_a, \mathbf{x}_\beta \in \mathcal{X}_b} d_{\mathcal{X}}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \geq \delta$$

Finally, this holds for all $a \neq b$, yielding the lemma. □

4.3.3 Asymptotic Limits of power weighted shortest paths

For all $a \in [k]$, define $d_{\mathcal{X}_a}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$ as the minimum p-weighted length of paths from $\mathbf{x}_\alpha$ to $\mathbf{x}_\beta$ *through* $\mathcal{X}_a$ (*i.e.* we are excluding paths that may pass through points in $\mathcal{X} \setminus \mathcal{X}_a$). Because $\mathcal{X}_a \subset \mathcal{X}$, it follows that $d_{\mathcal{X}}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq d_{\mathcal{X}_a}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$[1]. In this section we address the asymptotic behaviour of the power weighed shortest path distances $d_{\mathcal{X}_a}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$. Here is where we make critical use of the main theorem of [50], which we state as Theorem 4.3.2. Recall that μ_a is the probability density function with respect to which $\mathcal{X}_a$ is sampled from $\mathcal{M}_a$, and that by assumption $\mu_a^{\min} := \min_{x \in \mathcal{M}_a} \mu_a(x) > 0$. Define the following power-weighted geodesic distance on $\mathcal{M}_a$:

$$\mathrm{dist}_a^{(p)}(\mathbf{x}, \mathbf{y}) = \inf_{\eta} \int_0^1 \frac{\sqrt{g_a(\eta_t', \eta_t')}}{\mu_a(\eta_t)^{p-1/d_a}} dt \tag{4.5}$$

where here the infimum is over all piecewise smooth paths $\eta : [0,1] \to \mathcal{M}_a$ with $\eta(0) = \mathbf{x}$ and $\eta(1) = \mathbf{y}$. As in §4.3.1, for the Riemannian manifold $(\mathcal{M}_a, g_a)$ let $\mathrm{dist}_a(\mathbf{x}, \mathbf{y})$ denotes the

[1]More generally the reader is invited to check that for any $\mathcal{Y} \subset \mathcal{X}$ we have that $d_{\mathcal{X}}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq d_{\mathcal{Y}}^{(p)}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$.

geodesic distance from $\mathbf{x}$ to $\mathbf{y}$ on $\mathcal{M}_a$ with respect to g_a.

In order to bound $d^{(p)}_{\mathcal{X}_a}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$ we study an auxiliary shortest path distance $d^{(p)}_{\mathcal{M}_a}(\cdot,\cdot)$. This distance will also be defined as a minimum over p-weighted path lengths, but instead of measuring the length of the legs using the Euclidean distance $\|\cdot\|$, we measure them *with respect to the intrinsic metric* $\mathrm{dist}_a(\cdot,\cdot)$:

$$d^{(p)}_{\mathcal{M}_a}(\mathbf{x},\mathbf{y}) := \min_{\gamma} \left(\sum_{j=0}^{m} \mathrm{dist}_a(\mathbf{x}_{i_{j+1}}, \mathbf{x}_{i_j})^p \right)^{1/p} \tag{4.6}$$

where again the min is over all paths γ from $\mathbf{x}$ to $\mathbf{y}$ through $\mathcal{X}_a$.

Theorem 4.3.2 (Theorem 1 in [50] [2])**.** *Let $\mathcal{M}_a$ be a compact Riemannian manifold, and assume that $\mathcal{X}_a$ is drawn from $\mathcal{M}_a$ with probability distribution μ_a. Let $n_a := |\mathcal{X}_a|$. For all n_a, let $r_{n_a} := n_a^{(1-p)/pd_a}$ Then for any fixed $\epsilon > 0$:*

$$\mathbb{P}\left[\max_{\substack{\mathbf{x},\mathbf{y}\in\mathcal{M}_a \\ dist_a(\mathbf{x},\mathbf{y})\geq r_{n_a}}} \left| \frac{\left(d^{(p)}_{\mathcal{M}_a}(\mathbf{x},\mathbf{y})\right)^p}{n_a^{(1-p)/d_a}\, dist^{(p)}_a(\mathbf{x},\mathbf{y})} - C(d_a,p) \right| > \epsilon \right] = o_{n_a}(1) \tag{4.7}$$

where $C(d_a,p)$ is a constant depending only on d_a and p, but not on n_a. Note that the dependency on ϵ is contained in the $o_{n_a}(1)$ term.

Corollary 4.3.3. *With assumptions as in Theorem 4.3.2,*

$$\max_{\mathbf{x}_\alpha,\mathbf{x}_\beta\in\mathcal{X}_a} \left(d^{(p)}_{\mathcal{M}_a}(\mathbf{x}_\alpha,\mathbf{x}_\beta) \right) \leq C_a n_a^{(1-p)/pd_a}$$

with probability $1 - o_{n_a}(1)$, where C_a is a constant depending on $d_a, p, \epsilon, \mu_a^{\min}$ and $diam(\mathcal{M}_a)$ but not on n_a.

Proof. From Theorem 4.3.2 there are two cases to consider:

1. $\mathrm{dist}_a(\mathbf{x}_\alpha,\mathbf{x}_\beta) < r_{n_a} = n_a^{(1-p)/pd_a}$, or

[2] There is a slight notational discrepancy here. What is called $d^{(p)}_{\mathcal{X}}(\mathbf{x}_\alpha,\mathbf{x}_\beta)$ in [50] is our $\left(d^{(p)}_{\mathcal{M}_a}(\mathbf{x}_\alpha,\mathbf{x}_\beta)\right)^p$

2. $\text{dist}_a(\mathbf{x}_\alpha, \mathbf{x}_\beta) \geq r_{n_a}$.

In the first case, the one leg path $\gamma_{\alpha\to\beta} = \mathbf{x}_\alpha \to \mathbf{x}_\beta$ is a path through $\mathcal{X}_a$, hence:

$$d^{(p)}_{\mathcal{M}_a}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq (\text{dist}_a(\mathbf{x}_\alpha, \mathbf{x}_\beta)^p)^{1/p} < n_a^{(1-p)/pd_a}$$

and so with probability 1

$$\max_{\substack{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a \\ \text{dist}_a(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq r_{n_a}}} d^{(p)}_{\mathcal{M}_a}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq n_a^{(1-p)/pd_a} \tag{4.8}$$

For the second case, recall that $\mu_a^{\min} := \inf_{x \in \mathcal{M}_a} \mu_a(x)$. By assumption (see §4.3.1) $\mu_a^{\min} > 0$. From the definition of $\text{dist}^{(p)}_a(\mathbf{x}, \mathbf{y})$ (see (4.5))

$$\text{dist}^{(p)}_a(\mathbf{x}, \mathbf{y}) \leq \frac{1}{(\mu_a^{\min})^{(p-1)/d_a}} \inf_\eta \int_0^1 \sqrt{g_a(\eta_t', \eta_t')} dt = \frac{1}{(\mu_a^{\min})^{(p-1)/d_a}} \text{dist}_a(\mathbf{x}, \mathbf{y}) \tag{4.9}$$

Because $\mathcal{M}_a$ is compact and embedded, its diameter (see §4.3.1) is finite, and $\text{dist}_a(\mathbf{x}, \mathbf{y}) \leq \text{diam}(\mathcal{M}_a)$. Because $\mathcal{X}_a \subset \mathcal{M}_a$:

$$\max_{\substack{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a \\ \text{dist}_a(\mathbf{x}_\alpha, \mathbf{x}_\beta) \geq r_{n_a}}} \left| \frac{\left(d^{(p)}_{\mathcal{M}_a}(\mathbf{x}_\alpha, \mathbf{x}_\beta)\right)^p}{n_a^{(1-p)/d_a} \text{dist}^{(p)}_a(\mathbf{x}_\alpha, \mathbf{x}_\beta)} - C(d_a, p) \right| \leq \max_{\substack{\mathbf{x}, \mathbf{y} \in \mathcal{M}_a \\ \text{dist}_a(\mathbf{x}, \mathbf{y}) \geq r_{n_a}}} \left| \frac{\left(d^{(p)}_{\mathcal{M}_a}(\mathbf{x}, \mathbf{y})\right)^p}{n_a^{(1-p)/d_a} \text{dist}^{(p)}_a(\mathbf{x}, \mathbf{y})} - C(d_a, p) \right|$$

Hence from Theorem 4.3.2 and Equation (4.9), with probability $1 - o_{n_a}(1)$:

$$\max_{\substack{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a \\ \text{dist}_a(\mathbf{x}_\alpha, \mathbf{x}_\beta) \geq r_{n_a}}} \left(d^{(p)}_{\mathcal{M}_a}(\mathbf{x}_\alpha, \mathbf{x}_\beta)\right)^p \leq \frac{(C(d_a, p) + \epsilon)\, n_a^{(1-p)/d_a} \text{diam}(\mathcal{M}_a)}{(\mu_a^{\min})^{(p-1)/d_a}} = C_a n_a^{(1-p)/d_a}$$

$$\Rightarrow \max_{\substack{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a \\ \text{dist}_a(\mathbf{x}_\alpha, \mathbf{x}_\beta) \geq r_{n_a}}} d^{(p)}_{\mathcal{M}_a}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq \tilde{C}_a n_a^{(1-p)/pd_a} \tag{4.10}$$

where now $\tilde{C}_a = \left(\frac{(C(d_a, p) + \epsilon)\text{diam}(\mathcal{M}_a)}{(\mu_a^{\min})^{(p-1)/d_a}} \right)^{1/p}$. Combining equations (4.8) and (4.10) and redefining $C_a = \max\{\tilde{C}_a, 1\}$ proves the corollary. □

Finally, we observe that the Euclidean path distance is always smaller then the intrinsic path distance:

Lemma 4.3.4. *For any* $\mathbf{x}, \mathbf{y} \in \mathcal{M}_a$, *and for all* $a \in [k]$, $d^{(p)}_{\mathcal{X}_a}(\mathbf{x}, \mathbf{y}) \leq d^{(p)}_{\mathcal{M}_a}(\mathbf{x}, \mathbf{y})$

Proof. Observe that for any $\mathbf{x}, \mathbf{y} \in \mathcal{M}_a$, $\|\mathbf{x} - \mathbf{y}\| \leq \mathrm{dist}_a(\mathbf{x}, \mathbf{y})$. It follows that for any path $\gamma = \mathbf{x} \to \mathbf{x}_{i_1} \to \ldots \to \mathbf{x}_{i_m} \to \mathbf{y}$ through $\mathcal{X}_a$:

$$\sum_{j=0}^{m} \|\mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}\|^p \leq \sum_{j=0}^{m} \mathrm{dist}_a(\mathbf{x}_{i_{j+1}}, \mathbf{x}_{i_j})^p$$

and so:

$$\begin{aligned}\left(d^{(p)}_{\mathcal{X}_a}(\mathbf{x}, \mathbf{y})\right)^p &= \min_{\gamma} \left\{ \sum_{j=0}^{m} \|\mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}\|^p \right\} \\ &\leq \min_{\gamma} \left\{ \sum_{j=0}^{m} \mathrm{dist}_a(\mathbf{x}_{i_{j+1}}, \mathbf{x}_{i_j})^p \right\} = \left(d^{(p)}_{\mathcal{M}_a}(\mathbf{x}, \mathbf{y})\right)^p\end{aligned}$$

whence the result follows. □

4.3.4 Main Result

Let us now return to considering the full distance function $d^{(p)}_{\mathcal{X}}(\cdot, \cdot)$.

Theorem 4.3.5. *Let $n_{\min} := \min_a |\mathcal{X}_a|$ and $d_{\max} = \max_a \dim(\mathcal{M}_a)$ and define ϵ_1 to be the maximal distance between points in the same cluster:*

$$\epsilon_1 := \max_{a \in [k]} \max_{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a} d^{(p)}_{\mathcal{X}}(\mathbf{x}_\alpha, \mathbf{x}_\beta)$$

Then $\epsilon_1 = O(n_{\min}^{(1-p)/pd_{\max}})$ with probability approaching 1 as $n_{\min} \to \infty$. In particular, for all $p > 1$, $\epsilon_1 \to 0$ almost surely.

Proof. For any $a \in [k]$, Lemma 4.3.4 and Corollary 4.3.3 give us that:

$$\max_{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a} d^{(p)}_{\mathcal{X}}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq \max_{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a} d^{(p)}_{\mathcal{X}_a}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq \max_{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a} d^{(p)}_{\mathcal{M}_a}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq C_a n_a^{(1-p)/pd_a}$$

with probability $1 - o_{n_a}(1)$, where the first inequality follows from the fact that $\mathcal{X}_a \subset \mathcal{X}$. Taking the maximum over all $a \in [k]$ yields:

$$\max_{a \in [k]} \max_{\mathbf{x}_\alpha, \mathbf{x}_\beta \in \mathcal{X}_a} d^{(p)}_{\mathcal{X}}(\mathbf{x}_\alpha, \mathbf{x}_\beta) \leq C_{\max} n_{\min}^{(1-p)/pd_{\max}}$$

Where the constant $C_{\max}$ now depends on the geometry of the $\mathcal{M}_a$ (in particular their dimension and diameter) and the probability distributions μ_a (in particular the minimal values of μ_a) but not on the number of points per cluster, n_a. Finally, observe that for $p > 1$ we indeed have that $n_{\min}^{(1-p)/pd_{\max}} \to 0$ as $n_{\min} \to \infty$ □

4.4 A fast algorithm for computing K-nearest neighbors in the p-WSPD

In this section we focus on the computational issue of determining, for any $\mathbf{x}_\alpha \in \mathcal{X}$, the K-nearest neighbors of $\mathbf{x}_\alpha$ in the p-WSPD. We start with in a slightly more general situation, thus let $G = (V, E)$ be a weighted graph with weighted adjacency matrix A. We shall require all weights to be positive. Occasionally, it will prove more convenient to not fix an ordering of the vertices, in which case $A(u, v)$ will represent the weight of the edge $\{u, v\}$ (and again $A(u, v) = 0$ if there is no such edge). By $\gamma = u \to w_1 \to \ldots \to w_m \to v$ we shall mean the path from u to v in G through $w_1, \ldots, w_m$. Here, this is only valid if $\{u, w_1\}, \ldots, \{w_i, w_{i+1}\}, \ldots, \{w_m, v\}$ are all edges in G. In analogy with §4.1 we maintain the convention that for such a path γ, $w_0 = u$ and $w_{m+1} = v$. Define the *length* of γ as the sum of all its edge weights:

$$L(\gamma) := \sum_{i=0}^{m} A(w_i, w_{i+1})$$

and similarly define the longest-leg length of γ as:

$$L^{(\infty)}(\gamma) = \max_{i=0}^{m} A(w_i, w_{i+1})$$

For any $u, v \in V$ define the *shortest path distance* as:

$$d_G(u, v) = \min\{L(\gamma) : \ \gamma \text{ a path from } u \text{ to } v\}$$

and analogously define the longest-leg path distance as:

$$d_G^{(\infty)}(u, v) = \min\{L^{\infty}(\gamma) : \ \gamma \text{ a path from } u \text{ to } v\}$$

Let us relate this to the discussion in previous sections. For any set of data points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^D$ and any power weighting $p \geq 1$ one can form a complete weighted graph G on n vertices (which we referred to in §4.1 as a p-weighted distance graph of $\mathcal{X}$) with edge weights $A_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^p$. Then:

$$d_G(v_i, v_j) = \left(d_{\mathcal{X}}^{(p)}(\mathbf{x}_i, \mathbf{x}_j)\right)^p$$

Definition 4.4.1. Let $\mathcal{N}_{K,G}(v)$ denote the set of K nearest neighbors of $v \in V$. That is, $\mathcal{N}_{K,G}(v) = \{w_1, \ldots, w_K\}$ with $A(v, w_1) \leq A(v, w_2) \leq \ldots \leq A(v, w_K) \leq A(v, w)$ for all $w \in V \setminus \{w_1, \ldots, w_K\}$

Definition 4.4.2. For any graph G, define a directed K-Nearest Neighbors graph $G^{(K)}$ with directed edges (u, v) if $v \in \mathcal{N}_{K,G}(u)$.

In practice we do not compute the entire edge set of $G^{(K)}$, but rather just compute the sets $\mathcal{N}_{K,G}(u)$ as it becomes necessary.

Definition 4.4.3. Let $\mathcal{N}^{d_G}_{K,G}(v)$ denote the set of K nearest neighbors of v, with respect to the path distance d_G, in the graph G. That is, $\mathcal{N}^{d_G}_{K,G}(v) = \{w_1, \ldots, w_K\}$ and $d_G(v, w_1) \leq d_G(v, w_2) \leq \ldots d_G(v, w_K) \leq d_G(v, w)$ for all $w \in V \setminus \mathcal{N}^{d_G}_{K,G}(v)$. By convention, we take v to be in $\mathcal{N}^{d_G}_{K,G}(v)$ as $d_G(v, v) = 0$.

We have not specified how to break ties in the definition of $\mathcal{N}_{K,G}(v)$ or $\mathcal{N}^{d_G}_{K,G}(v)$. For the results of this section to hold, any method will suffice, as long as we use the same method in both cases. To simplify the exposition, we shall assume henceforth that all distances are distinct.

Let us briefly review how Dijkstra's algorithm works. The algorithm makes use of a data structure called a min-priority queue. The following implementation is as in [32]. The min-priority queue operations `decreaseKey`, `insert` and `extractMin` have their standard definitions (see, for example Chpt. 6 of [32]). For any vertex $s \in V$ and any subset $W \subset V$, we shall also use the shorthand `makeQueue`(W, s) to denote the process of initializing a min-priority queue with `key`$[s] = 0$ and `key`$[v] = +\infty$ for all $v \in W \setminus s$.

Note that once u is popped in step 4 of Algorithm 11, `key`$[u]$ is the shortest path distance from s to u. The key observation is the following:

Lemma 4.4.4. *Suppose that all weights are non-negative: $w_{ij} \geq 0$. If u_i is the i-th vertex to be removed from Q at step 11, then u_i is the i-th closest vertex to s.*

Algorithm 11 Dijkstra

1: **Input:** weighted graph G, source vertex s.
2: **Initialize:** $Q \leftarrow$ `makeQueue`(V, s). Empty list S.
3: **while** Q is non-empty **do**
4: $u \leftarrow$ `extractMin`(Q)
5: Append $(u,$ `key`$[u])$ to S. ▷ Once u is extracted `key`$[u]$ is shortest path length from s.
6: **for** $v \in \mathcal{N}(u)$ **do** ▷ $\mathcal{N}(u)$ is the set of all vertices adjacent to u
7: tempDist $\leftarrow$ `key`$[u] + A(u, v)$
8: **if** tempDist $<$ `key`$[v]$ **then**
9: `key`$[v] \leftarrow$ tempDist ▷ Update the distance from s to v if path through u is shorter
10: **end if**
11: **end for**
12: **end while**
13: **Output:** S

Proof. See, for example, the discussion in [32]. □

It follows that, if one is only interested in finding the K nearest neighbors of s in the path distance d_G, one need only iterate through the 'while' loop $3 \to 12$ K times. There is a further inefficiency, which was also highlighted in [13]. The 'for' loop 6–10 iterates over all neighbors of u. The graphs we are interested in are, implicitly, fully connected, hence this for loop iterates over all $n - 1$ other vertices at each step. We fix this with the following observation:

Lemma 4.4.5. *For any graph G, let $G^{(K)}$ denote its K-Nearest-Neighbor graph (see Definition 4.4.2). Then:*

$$\mathcal{N}_{K,G}^{d_G}(v) = \mathcal{N}_{G^{(K)},K}^{d_{G^{(K)}}}(v) \quad \textit{for all } v$$

Note that in the directed graph $G^{(K)}$, we consider only paths that traverse each edge in the 'correct' direction.

Concretely: the path-nearest-neighbors in G are the same as the path-nearest neighbors in $G^{(K)}$, hence one can find $\mathcal{N}_{K,G}^{d_G}(v)$ by running a Dijkstra-style algorithm on $G^{(K)}$ instead

of G. As each vertex in $G^{(K)}$ has a small number of neighbors (*i.e.* K), this alleviates the computational burden of the 'for' loop 6–10 highlighted above. Before proving this lemma, let us explain why it may seem counterintuitive. If $w \in \mathcal{N}_{K,G}^{d_G}(v)$ there is a path γ from v to w that is short (at least shorter than the shortest paths to all $u \notin \mathcal{N}_{K,G}^{d_G}(v)$). In forming $G^{(K)}$ from G, one deletes a lot of edges. Thus it is not clear that γ is still a path in $G^{(K)}$ (some of its edges may now be 'missing'). Hence it would seem possible that w is now far away from v in the shortest-path distance in $G^{(K)}$. The lemma asserts that this cannot be the case.

Proof. Since the sets $\mathcal{N}_{K,G}^{d_G}(v)$ and $\mathcal{N}_{G^{(K)},K}^{d_{G^{(K)}}}(v)$ have the same cardinality (*i.e.* K), to prove equality it suffices to prove one containment. We shall show that $\mathcal{N}_{K,G}^{d_G}(v) \subset \mathcal{N}_{G^{(K)},K}^{d_{G^{(K)}}}(v)$. Consider any $w \in \mathcal{N}_{K,G}^{d_G}(v)$. Let $\gamma^* = v \to u_1 \to \ldots \to u_m \to w$ be the shortest path from v to w. That is, $L(\gamma^*) = \min\{L(\gamma) : \gamma \text{ a path from } v \text{ to } w\}$.

We claim that γ^* is a path in $G^{(K)}$. If this is not the case, then there is an edge $\{u_i, u_{i+1}\}$ that is in γ^* but (u_i, u_{i+1}) is not an edge in $G^{(K)}$ (we again adopt the convention that $u_0 := v$ and $u_{m+1} := w$). By the construction of $G^{(K)}$ this implies that there are K vertices $x_1, \ldots, x_K$ that are closer to u_i than u_{i+1}. (Note that the sets $\{u_0, \ldots, u_{i-1}\}$ and $\{x_1, \ldots, x_K\}$ need not be disjoint). But then the paths $\gamma_j = v \to u_1 \to \ldots \to u_i \to x_j$ in G are all shorter than the path $v \to u_1 \to \ldots \to u_{i+1}$ and hence shorter than γ^*, as all edge weights are assumed positive. It follows that $d_G(v, x_j) < d_G(v, w)$ for $j = 1, \ldots, K$, contradicting the assumption that $w \in \mathcal{N}_{K,G}^{d_G}(v)$.

Now, we claim that $w \in \mathcal{N}_{G^{(K)},K}^{d_{G^{(K)}}}(v)$. If this were not the case, there would exists K other vertices $w_1, \ldots, w_K$ that are closer in the shortest-path distance $d_{G^{(K)}}$ to v than w. That is, there would be paths $\gamma_1, \ldots, \gamma_K$ from v to $w_1, \ldots, w_K$ respectively that are shorter than γ. But every path in $G^{(K)}$ is also a path in G, hence $w_1, \ldots, w_K$ are also closer to v than w in the shortest-path distance d_G. This contradicts the assumption that $w \in \mathcal{N}_{G,K}^{d_G}(v)$. □

There is a final, minor, inefficiency in Algorithm 11 that we can improve upon; Q is initialized to contain all vertices V when it is actually only neccessary to initialize it to contain the neighbors of s. Combining these three insights we arrive at Algorithm 12. Note that we use `DecreaseOrInsert` as shorthand for the function that decreases `key`$[v]$ to tempDist if tempDist $<$ `key`$[v]$ and $v \in Q$, inserts v into Q with priority `key`$[v]$ = tempDist if $v \notin Q$ and does nothing if $v \in Q$ but tempDist $\geq$ `key`$[v]$. In fact, this is equivalent to inserting a copy of v into Q with priority `key`$[v]$ = tempDist, hence `DecreaseOrInsert` has the same computational complexity as `insert` (see also [66]). Note that in this implementation the size of Q grows by one on every iteration of the inner for loop, 10–13.

Algorithm 12 Dijkstra-with-early-stopping

1: **Input:** Graph G, source vertex s.
2: **Output:** List S containing $(v, d_G(s,v))$ for all $v \in \mathcal{N}_{G,k}^{d_G}(s)$.
3: Compute $\mathcal{N}_{K,G}(s)$
4: **Initialize:** $Q \leftarrow$ `makeQueue`$(\mathcal{N}_{K,G}(s), s)$. Empty list S.
5: **for** i = 1:k **do**
6: $u \leftarrow$ `extractMin`(Q)
7: Append $(u,$ `key`$[u])$ to S
8: Compute $\mathcal{N}_{K,G}(u)$
9: **for** $v \in \mathcal{N}_{K,G}(u)$ **do**
10: tempDist $\leftarrow$ `key`$[u] + A(u,v)$
11: `DecreaseOrInsert`$(v,$ tempDist$)$
12: **end for**
13: **end for**
14: **Output:** S

Theorem 4.4.6. *For any s and any G with positive weights, Algorithm 12 is correct.*

Proof. By only using $\mathcal{N}_{K,G}(u)$ in step 8, Algorithm 12 is essentially running Dijkstra's algorithm on $G^{(K)}$. By Lemma 4.4.4, the first K elements to be popped off the queue in line 9 are indeed the K path-closest vertices to s in the graph $G^{(K)}$. That is, S contains $(v, d_G(s,v))$ for all $v \in \mathcal{N}_{G^{(K)},K}^{d_{G^{(K)}}}(s)$. By lemma 4.4.5, $\mathcal{N}_{G^{(K)},K}^{d_{G^{(K)}}}(s) = \mathcal{N}_{G,K}^{d_G}(s)$ □

4.4.1 Analysis of complexity

Let us determine the computational complexity of Algorithm 12. We shall remain agnostic for the moment about the precise implementation of the min-priority queue, and hence shall use the symbols $\mathcal{T}_{in}, \mathcal{T}_{dk}$ and $\mathcal{T}_{em}$ to denote the computational complexity of `insert`, `decreaseKey` and `extractMin` respectively. As discussed above, the complexity of `DecreaseOrInsert` is also $\mathcal{T}_{in}$. Let $\mathcal{T}_{Knn}$ denote the cost of a K-Nearest Neighbors query in G (*i.e.* the cost of determining $\mathcal{N}_{K,G}(u)$, as in line 8.)

Initializing the queue in line 4 requires K insertions, for a cost of $K\mathcal{T}_{in}$. Precisely K `extractMin` operations are performed, for a total cost of $K\mathcal{T}_{em}$. `DecreaseOrInsert` is performed K^2 times, for a cost of $K^2\mathcal{T}_{in}$. Finally $K+1$ K-Nearest Neighbor queries are performed, for a cost of $(K+1)\mathcal{T}_{Knn}$. This gives a total cost of $K\mathcal{T}_{em}+(K+K^2)\mathcal{T}_{in}+(K+1)\mathcal{T}_{Knn}$.

If the min priority queue is implemented using a Fibonacci heap, `insert` and `decreaseKey` both run in constant time (*i.e.* $\mathcal{T}_{in}, \mathcal{T}_{dk} = O(1)$) while for `extractMin` $\mathcal{T}_{em} = O(\log(|Q|))$ time. $|Q|$ never exceeds $K^2 + K$ as at most one element is added to Q during every pass through the inner for loop 9–12, which happens K^2 times. Hence $\mathcal{T}_{em} = O(\log(k))$. The value of $\mathcal{T}_{Knn}$ depends on the specifics of G. If the graph has no additional structure, finding the K nearest neighbors of any vertex v can be done with K-calls to the `Select` algorithm (cf. Chpt. 9 of [32]) for a cost of $O(Kn)$. However, if G is the (possible p-weighted) distance graph of a data set $\mathcal{X} \subset \mathbb{R}^D$ which is drawn from a subset $\mathcal{M} \subset \mathbb{R}^D$ of intrinsically low dimension such as a manifold (which we are assuming), and stored in an efficient data structure such as a k-d tree [11] or a cover tree [12] then one can complete a nearest neighbor query in $O(\log(n))$ time, hence one can find the K nearest neighbors in $O(K\log(n))$ time. Putting this all together, Algorithm 12 runs in $O(K\log(K) + (K+K^2) + K^2\log(n)) = O(K^2\log(n))$ time. Finding $\mathcal{N}^{d_G}_{G,K}(v)$ for all $v \in V$ requires running Algorithm 12 n times, for a total cost

of $O(K^2 n \log(n))$. We remark that the time required to initialize a k-d tree for $\mathcal{X} \subset \mathbb{R}^D$ with $|\mathcal{X}| = n$ is $O(Dn \log(n))$, which is of the same order of magnitude.

4.4.2 Extension to Longest-Leg Path Distance

A small modification to Algorithm 12 allows one to compute the K nearest neighbors in the longest-leg-path distance, simply change the '+' in line 10 to a 'max'. This guarantees that tempDist represents the longest-leg length of the path $s \to \ldots \to u \to v$. For completeness, we present this algorithm below as Algorithm 13. The proof of correctness is analogous to Theorem 4.4.6, and we leave it to the interested reader.

Algorithm 13 Dijkstra-with-early-stopping for LLPD

1: **Input:** Graph G, source vertex s.
2: **Output:** List S containing $(v, d_G(s, v)$ for all $v \in \mathcal{N}_{G,K}^{d_G^\infty}(v)$.
3: Compute $\mathcal{N}_{K,G}(s)$
4: **Initialize:** $Q \leftarrow$ `makeQueue`$(\mathcal{N}_{K,G}(s), s)$. Empty list S.
5: **for** i = 1:k **do**
6: $u \leftarrow$ `extractMin`(Q)
7: Append $(u,$ `key`$[u])$ to S
8: Compute $\mathcal{N}_{K,G}(u)$
9: **for** $v \in \mathcal{N}_{G^{(K)}}(u)$ **do**
10: tempDist $\leftarrow \max\{$`key`$[u], A(u, v)\}$
11: `DecreaseOrInsert`$(v,$ tempDist$)$
12: **end for**
13: **end for**
14: **Output:** S

4.5 Experimental Results

4.5.1 Comparing Clustering Accuracy

In this section we verify that using a p-WSPD in lieu of the Euclidean distance does indeed result in more accurate clustering results, at a modest increase in run time. Specifically, we compare $\|\cdot\|$ to $d_{\mathcal{X}}^{(2)}, d_{\mathcal{X}}^{(10)}$ and $d_{\mathcal{X}}^{(\infty)}$. For each distance function we employ the same methodology as in §2.6.5, which we now recall. For notational reasons it is convenient to denote $\|\cdot\|$ as $d^{(1)}$ (which is correct by Lemma 4.1.2).

- Fix parameters $r = 10$ and $K = 15$.
- For $\xi = 1, 2, 10, \infty$, and for all $i \in [n]$, define $\sigma_i^{(\xi)} := d_{\mathcal{X}}^{(\xi)}(\mathbf{x}_i, \mathbf{x}_{[r,i]})$, where $\mathbf{x}_{[r,i]}$ denotes the r-th closest point in $\mathcal{X}$ to $\mathbf{x}_i$ with respect to the distance $d_{\mathcal{X}}^{(\xi)}$. Let $\mathrm{NN}^{(\xi)}(\mathbf{x}_i, K) \subset \mathcal{X}$ denote the set of the K closest points in $\mathcal{X}$ to $\mathbf{x}_i$ with respect to $d_{\mathcal{X}}^{(\xi)}$.
- Define $\tilde{A}^{(\xi)}$ as: $\tilde{A}_{ij}^{(\xi)} = \begin{cases} \exp\left(-d_{\mathcal{X}}^{(\xi)}(\mathbf{x}_i, \mathbf{x}_j)^2 / \sigma_i \sigma_j\right) & \text{if } \mathbf{x}_j \in NN^{(\xi)}(\mathbf{x}_i, K) \\ 0 & \text{otherwise} \end{cases}$
- Symmetrize by defining $A_{ij}^{(\xi)} = \max\left\{\tilde{A}_{ij}^{(\xi)}, \tilde{A}_{ji}^{(\xi)}\right\}$

For each $A^{(\xi)}$ we perform normalized spectral clustering as described in Ng, Jordan and Weiss [73] using code freely available at `https://www.mathworks.com/matlabcentral/fileexchange/34412-fast-and-efficient-spectral-clustering` and calculate the accuracy by comparing the results of spectral clustering to the ground truth. We also record the time taken to create each $A^{(\xi)}$ (the running times of spectral clustering with any of the $A^{(\xi)}$ are all very similar).

Spectral Clustering for Synthetic Data Sets

We considered the three synthetic data sets described below. All results for synthetic data were averaged over 100 independent trials.

Three Lines. We draw data uniformly from three horizontal line segments of length 5 in the x-y plane, namely $y = 0$, $y = 1$ and $y = 2$. We draw 500 points from each line to create three clusters. We then embed the data into $\mathbb{R}^{50}$ by appending zeros to the coordinates, and add i.i.d. random Gaussian noise to each coordinate (with standard deviation $\sigma = 0.14$).

Three Moons. This data set is as described in [94] and elsewhere. It has three clusters, generated by sampling points uniformly at random from the upper semi-circle of radius 1 centered at $(0, 0)$, the lower semi-circle of radius 1.5 centered at $(1.5, 0.4)$ and the upper

semi-circle of radius 1 centered at $(3, 0)$. As for the Three Lines data set, we draw 500 data points from each semi-circle, embed the data into $\mathbb{R}^{50}$ by appending zeros, and then add Gaussian noise to each coordinate with standard deviation $\sigma = 0.14$.

Three Circles. Here we draw data points uniformly from three concentric circles, of radii $1, 2.25$ and 3.5. We draw 222 points from the smallest circle, 500 points from the middle circle and 778 points from the largest circle (the numbers are chosen so that the total number of points is 1500). As before, we embed this data into $\mathbb{R}^{50}$ and add i.i.d Gaussian noise to each component, this time with standard deviation of $\sigma = 0.14$.

Two dimensional projections of these data sets are shown in Figure 4.2, while results of spectral clustering are shown in Table 4.1. In Table 4.2 we record the time taken to create $A^{(1)}, A^{(2)}, A^{(10)}$ and $A^{(\infty)}$ respectively.

Note that the cost of creating $A^{(2)}, A^{(10)}$ and $A^{(\infty)}$ is dominated by the run time of Algorithm 12. From §4.4.1, if $T(n)$ denotes the run time of Algorithm 12 for a data set of size n, we expect $T(n) = C\log(n)n$ and hence $\log(T(n)) = \log(n)+\log(\log(n))+\log(C)$. In Figure 4.3 we verify that the time required to create $A^{(2)}$ agrees with this asymptotic prediction by plotting the logarithm of the time required to create $A^{(2)}$ for all three synthetic data sets as the number of data points varies from $n = 300$ to $n = 12\ 000$ (we keep the parameters $K = 15$ and $r = 10$ as before). We also plot the curve $y = \log(n) + \log(\log(n))$ and observe that this has the same shape as the plots of the run times.

Spectral Clustering for Real Data sets

We also considered three real data sets. We focus on a particular kind of data known to satisfy the manifold hypothesis, namely handwritten digits.

	$A^{(1)}$	$A^{(2)}$	$A^{(10)}$	$A^{(\infty)}$
3 Lines	67.53%	67.44%	**92.41%**	89.55%
3 Moons	94.60%	94.61%	**96.12%**	94.88%
3 Circles	52.36%	52.37%	69.91%	**72.20%**

Table 4.1: Classification accuracy of spectral clustering given K-NN adjacency matrices for various distance functions. Note that $A^{(1)}$ represents using the Euclidean metric.

	$A^{(1)}$	$A^{(2)}$	$A^{(10)}$	$A^{(\infty)}$
3 Lines	0.11	1.04	1.17	1.10
3 Moons	0.11	1.15	1.23	1.18
3 Circles	0.07	0.83	1.03	0.93

Table 4.2: Time taken to create the adjacency matrices, in seconds, for the three synthetic data sets.

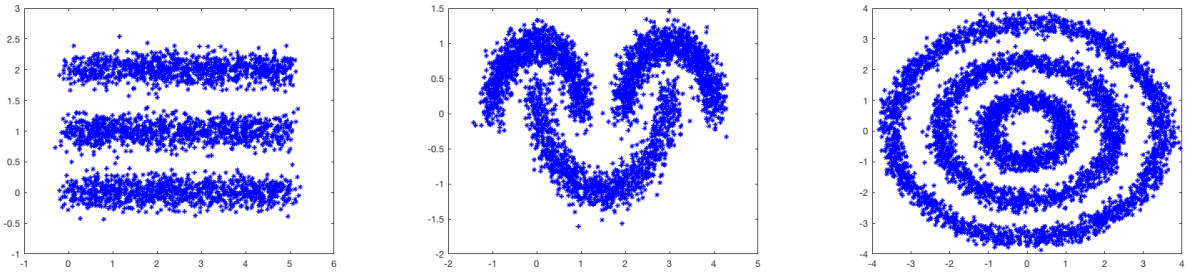

Figure 4.2: All three synthetic data sets, projected into $\mathbb{R}^2$. From left to right: Three Lines, Three Moons and Three Circles.

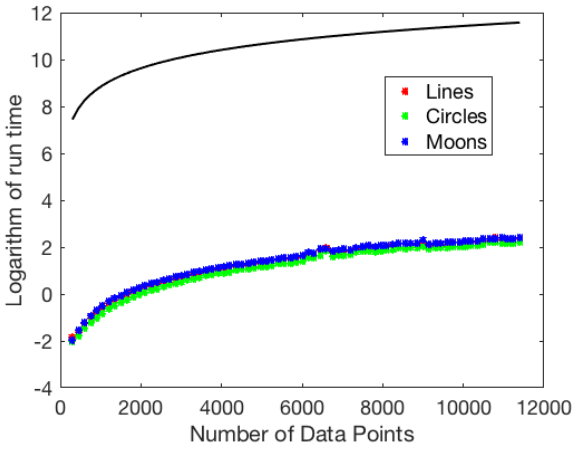

Figure 4.3: The run time required to construct $A^{(2)}$, shown in logarithmic scale, for all three synthetic data sets. This conforms closely to the expected run time of Algorithm 12, which is $O(kn\log(n))$.

	$A^{(1)}$	$A^{(2)}$	$A^{(10)}$	$A^{(\infty)}$
`OptDigits`	91.19%	**92.42**%	89.89%	87.82%
`USPS`	65.58%	66.07%	75.41%	**78.65**%
`MNIST`	76.59%	75.85%	**86.86**%	85.80%

Table 4.3: Classification accuracy of spectral clustering given the K-NN adjacency matrices constructed as described earlier, for the various distance functions. Again, note that $A^{(1)}$ represents using the Euclidean metric.

	$A^{(1)}$	$A^{(2)}$	$A^{(10)}$	$A^{(\infty)}$
`OptDigits`	3.47	7.23	7.66	7.68
`USPS`	47.62	62.03	58.49	59.68
`MNIST`	1325.64	1396.10	1404.02	1391.37

Table 4.4: Time taken to create the various adjacency matrices, in seconds.

`OptDigits` This data set consists of downsampled, 8×8 greyscale images of handwritten digits $0-9$. There are 150 images of zero, and approximately 550 images each of the remaining digits, for a total of 5620 images.

`USPS` This data set consists of 16×16, greyscale images of the handwritten digits 0–9. There are 1100 images per class for a total of 11 000 images.

`MNIST` This data set consists of 28×28 greyscale images of the handwritten digits 0–9. We combined the test and training sets to get a total of 70 000 images.

For all data sets, the only preprocessing that was performed was to turn the square arrays into vectors. We averaged all results over five trials. The results are shown in Tables 4.3 and 4.4.

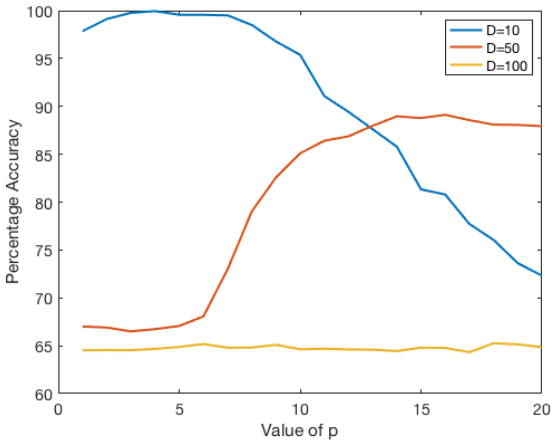

Figure 4.4: Varying p and recording the accuracy of spectral clustering on the Three Lines data set, for three different values of the ambient dimension.

4.5.2 Varying the Power Weighting

From the analysis of §4.3 it would appear that taking p to be as large as possible always results in the best clustering results. However, this is true only in an asymptotic sense. For finite sample size the constants C_a, depending on the geometry of the $\mathcal{M}_a$ and the distributions μ_a, may come into play. In Figure 4.4 we show the results of varying p from 1 to 20 for the three lines data set, this time with 300 points drawn from each cluster. We do this for three values of the ambient dimension, $D = 10, 50$ and 100. As is clear, the optimal value of p depends on the dimension[3]. In particular, observe that an intermediate value of p, say $p = 14$, is optimal when the ambient dimension is 50 but that a smaller power weighting ($p = 2$) is more appropriate when the ambient dimension is 10. When the ambient dimension is 100, no power weighting performs well, which is likely because for such a large value of D the noise drowns out any cluster structure.

[3]The observant reader will notice that we are only varying the ambient dimension, which according to the analysis of §4.3 should have no effect. Recall however, that we are adding Gaussian noise of the ambient dimension, which 'thickens' the data manifolds and makes their intrinsic dimension weakly dependent on D.

	$A^{(1)}$	$A^{(2)}$	$A^{(10)}$	$A^{(\infty)}$
0.1%	80.67%	80.55%	83.00%	**87.85**%
0.2%	90.25%	90.36%	88.28%	**91.26**%
0.3%	92.20%	**92.21**%	89.54%	91.82%
0.4%	93.15%	**93.19**%	90.62%	92.30%
0.5%	**95.29**%	93.42%	91.03%	92.49%

Table 4.5: Accuracy of using `IteratedSingleClusterPursuit` for semi-supervised classification on `MNIST`. Using a p-WSPD in place of Euclidean distance results in more accurate classification when the amount of labeled data is very small.

4.5.3 Combining p-WSPD's with `SingleClusterPursuit`

We now reconsider the experiments of §2.6.6 where `IteratedSingleClusterPursuit` was used for semi-supervised classification on the `MNIST` data set. In particular, we compare the performance of `IteratedSingleClusterPursuit` using a Euclidean-distance-based K-NN graph, namely $A^{(1)}$, to its performance using $A^{(2)}, A^{(10)}$ and $A^{(\infty)}$. We consider the regime where very little labeled data is available—we vary the percentage of labeled data from 0.1% to 0.5% in increments of 0.1%. As in §2.6.6 we take $R = 0.7$, $\epsilon = 0.13$ and $t = 3$. We draw Γ_a uniformly at random from C_a, and for each a have $|\Gamma_a| = \lceil g|C_a| \rceil$ where g varies from 0.001 to 0.005 in increments of 0.001 (in particular, for $g = 0.001$ we have $|\Gamma_a| = 8$ for all a). The results are displayed in Table 4.5.

4.5.4 Discussion of Numerical Results

It is clear that using a p-WSPD instead of Euclidean distance increases the accuracy of Spectral Clustering and `SingleClusterPursuit` on both real and synthetic data. This agrees with the theory of §4.3, which shows that points in the same cluster are "squeezed" together in a p-WSPD, while points in different clusters remain separated. Importantly, the experiments of this section verify that using a p-WSPD instead of Euclidean distance when building a K-NN graph results in a very modest increase in run time if Algorithm

12 is used. Indeed, on the largest data set, namely `MNIST`, the run times are practically the same. Thus, we recommend that, whenever data science practitioners would use a Euclidean K-NN graph on data that satisfies the manifold hypothesis, they consider using a p-WSPD and Algorithm 12 instead.

A curious phenomenon which we observed in the numerical results is the interaction between the power weighting, p, the ambient dimension D and the geometry of the data set $\mathcal{X}$. It seems as if the best value of p differs according to the specifics of the data. Tools to estimate the "optimal" p given the data $\mathcal{X}$ would enable data scientists to get the most out of the p-WSPD technology, but we leave this investigation to future work.

Bibliography

[1] Ameer Ahmed Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14-15):2826–2841, 2007.

[2] Emmanuel Abbe. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research*, 18(1):6446–6531, 2017.

[3] Emmanuel Abbe, Afonso S. Bandeira, Annina Bracher, and Amit Singer. Linear inverse problems on Erdős-Rényi graphs: Information-theoretic limits and efficient recovery. In *2014 IEEE International Symposium on Information Theory*, pages 1251–1255. IEEE, 2014.

[4] Emmanuel Abbe and Colin Sandon. Community detection in general stochastic block models: Fundamental limits and efficient algorithms for recovery. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 670–688. IEEE, 2015.

[5] Emmanuel Abbe and Colin Sandon. Recovering communities in the general stochastic block model without knowing the parameters. In *Advances in neural information processing systems*, pages 676–684, 2015.

[6] Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.

[7] Morteza Alamgir and Ulrike Von Luxburg. Shortest path distance in random k-nearest neighbor graphs. *arXiv preprint arXiv:1206.6381*, 2012.

[8] Reid Andersen, Fan Chung, and Kevin Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 4(1):35–64, 2007.

[9] Ery Arias-Castro. Clustering based on pairwise distances when the data is of mixed dimensions. *IEEE Transactions on Information Theory*, 57(3):1692–1706, 2011.

[10] Richard Baraniuk, Mark Davenport, Ronald DeVore, and Michael Wakin. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, 28(3):253–263, 2008.

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

[12] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104. ACM, 2006.

[13] Avleen S. Bijral, Nathan Ratliff, and Nathan Srebro. Semi-supervised learning with density based distances. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 43–50. AUAI Press, 2011.

[14] Thomas Blumensath and Mike E. Davies. Iterative hard thresholding for compressed sensing. *Applied and computational harmonic analysis*, 27(3):265–274, 2009.

[15] Béla Bollobás. Vertices of given degree in a random graph. *Journal of Graph Theory*, 6(2):147–155, 1982.

[16] Béla Bollobás. *Random graphs*. Number 73. Cambridge university press, 2001.

[17] T. Tony Cai and Anru Zhang. Sharp RIP bound for sparse signal and low-rank matrix recovery. *Applied and Computational Harmonic Analysis*, 35(1):74–93, 2013.

[18] Emmanuel J. Candes, Justin Romberg, and Terrence Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[19] Avishy Carmi, François Septier, and Simon J. Godsill. The gaussian mixture MCMC particle algorithm for dynamic cluster tracking. *Automatica*, 48(10):2454–2467, 2012.

[20] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 554–560. ACM, 2006.

[21] Hong Chang and Dit-Yan Yeung. Robust path-based spectral clustering. *Pattern Recognition*, 41(1):191–203, 2008.

[22] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 153–162. ACM, 2007.

[23] Timothy Chu, Gary Miller, and Donald Sheehy. Intrinsic metrics: Nearest neighbor and edge squared distances. *arXiv preprint arXiv:1709.07797*, 2017.

[24] Fan Chung. Spectral graph theory (CBMS regional conference series in mathematics, no. 92). 1996.

[25] Fan Chung. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, 2007.

[26] Fan Chung. Random walks and local cuts in graphs. *Linear Algebra and its applications*, 423(1):22–32, 2007.

[27] Fan Chung and Ron Graham. Quasi-random graphs with given degree sequences. *Random Structures & Algorithms*, 32(1):1–19, 2008.

[28] Fan Chung and Mary Radcliffe. On the spectra of general random graphs. *The Electronic Journal of Combinatorics*, 18(1):215, 2011.

[29] Fan Chung and Olivia Simpson. Computing heat kernel pagerank and a local clustering algorithm. *European Journal of Combinatorics*, 68:96–119, 2018.

[30] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006.

[31] Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. Large scale transductive SVMs. *Journal of Machine Learning Research*, 7(Aug):1687–1712, 2006.

[32] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[33] Wei Dai and Olgica Milenkovic. Subspace pursuit for compressive sensing signal reconstruction. *IEEE transactions on Information Theory*, 55(5):2230–2249, 2009.

[34] Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. Generalized Louvain method for community detection in large networks. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 88–93. IEEE, 2011.

[35] David L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[36] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

[37] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[38] Bernd Fischer and Joachim M. Buhmann. Path-based clustering for grouping of smooth curves and texture segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(4):513–518, 2003.

[39] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[40] Simon Foucart. Hard thresholding pursuit: an algorithm for compressive sensing. *SIAM Journal on Numerical Analysis*, 49(6):2543–2563, 2011.

[41] Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Springer Science & Business Media, 2013.

[42] Alan Frieze and Michał Karoński. *Introduction to random graphs*. Cambridge University Press, 2016.

[43] Amir Ghasemian, Pan Zhang, Aaron Clauset, Cristopher Moore, and Leto Peel. Detectability thresholds and optimal algorithms for community structure in dynamic networks. *Physical Review X*, 6(3):031005, 2016.

[44] Sariel Har-Peled. Computing the k nearest-neighbors for all vertices via Dijkstra. *arXiv preprint arXiv:1607.07818*, 2016.

[45] Tanja Hartmann, Andrea Kappes, and Dorothea Wagner. Clustering evolving networks. In *Algorithm Engineering*, pages 280–329. Springer, 2016.

[46] Kun He, Yiwei Sun, David Bindel, John Hopcroft, and Yixuan Li. Detecting overlapping communities from local spectral subspaces. In *2015 IEEE International Conference on Data Mining*, pages 769–774. IEEE, 2015.

[47] Matthew A. Herman and Thomas Strohmer. General deviants: An analysis of perturbations in compressed sensing. *IEEE Journal of Selected Topics in Signal Processing*, 4(2):342–349, 2010.

[48] Qirong Ho, Le Song, and Eric Xing. Evolving cluster mixed-membership blockmodel for time-evolving networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 342–350, 2011.

[49] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[50] Sung Jin Hwang, Steven B. Damelin, and Alfred O. Hero, III. Shortest path through random points. *The Annals of Applied Probability*, 26(5):2791–2823, 2016.

[51] Matt Jacobs, Ekaterina Merkurjev, and Selim Esedoḡlu. Auction dynamics: A volume constrained MBO scheme. *Journal of Computational Physics*, 354:288–310, 2018.

[52] Anil K. Jain. Data clustering: 50 years beyond K-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[53] Durk P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[54] Kyle Kloster and David F. Gleich. Heat kernel based community detection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1386–1395. ACM, 2014.

[55] Ming-Jun Lai and Daniel Mckenzie. Semi-supervised cluster extraction via a compressive sensing approach. *arXiv preprint arXiv:1808.05780*, 2018.

[56] Ming-Jun Lai and Paul Wenston. ℓ_1 spline methods for scattered data interpolation and approximation. *Advances in Computational Mathematics*, 21(3-4):293–315, 2004.

[57] Haifeng Li. Improved analysis of SP and CoSaMP under total perturbations. *EURASIP Journal on Advances in Signal Processing*, 2016(1):112, 2016.

[58] Yixuan Li, Kun He, David Bindel, and John E. Hopcroft. Uncovering the small community structure in large networks: A local spectral approach. In *Proceedings of the 24th international conference on world wide web*, pages 658–668, 2015.

[59] Yixuan Li, Kun He, Kyle Kloster, David Bindel, and John E. Hopcroft. Local spectral clustering for overlapping community detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(2):17, 2018.

[60] Anna Little, Mauro Maggioni, and James M. Murphy. Path-based spectral clustering: Guarantees, robustness to outliers, and fast algorithms. *arXiv preprint arXiv:1712.06206*, 2017.

[61] Michael W. Mahoney, Lorenzo Orecchia, and Nisheeth K. Vishnoi. A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *Journal of Machine Learning Research*, 13(Aug):2339–2365, 2012.

[62] Shahar Mendelson, Alain Pajor, and Nicole Tomczak-Jaegermann. Uniform uncertainty principle for Bernoulli and subgaussian ensembles. *Constructive Approximation*, 28(3):277–289, 2008.

[63] Ekaterina Merkurjev, Andrea L. Bertozzi, and Fan Chung. A semi-supervised heat kernel pagerank MBO algorithm for data classification. Technical report, University of California, Los Angeles, 2016.

[64] Ekaterina Merkurjev, Tijana Kostic, and Andrea L. Bertozzi. An MBO scheme on graphs for classification and image processing. *SIAM Journal on Imaging Sciences*, 6(4):1903–1930, 2013.

[65] Ekaterina Merkurjev, Justin Sunu, and Andrea L. Bertozzi. Graph MBO method for multiclass segmentation of hyperspectral stand-off detection video. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 689–693. IEEE, 2014.

[66] Amit Moscovich, Ariel Jaffe, and Nadler Boaz. Minimax-optimal semi-supervised regression on unknown manifolds. In *Artificial Intelligence and Statistics*, pages 933–942, 2017.

[67] Elchanan Mossel, Joe Neeman, and Allan Sly. Consistency thresholds for the planted bisection model. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 69–75. ACM, 2015.

[68] Elchanan Mossel, Joe Neeman, and Allan Sly. A proof of the block model threshold conjecture. *Combinatorica*, 38(3):665–708, 2018.

[69] Peter J. Mucha, Thomas Richardson, Kevin Macon, Mason A. Porter, and Jukka-Pekka Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *Science*, 328(5980):876–878, 2010.

[70] Maria C.V. Nascimento and Andre C.P.L.F. De Carvalho. Spectral methods for graph clustering–a survey. *European Journal of Operational Research*, 211(2):221–231, 2011.

[71] Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

[72] Deanna Needell and Joel A. Tropp. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321, 2009.

[73] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2002.

[74] Alon Orlitsky and Sajama. Estimating and computing density based distance metrics. In *Proceedings of the 22nd international conference on Machine Learning*, pages 760–767. ACM, 2005.

[75] Marianna Pensky and Teng Zhang. Spectral clustering in the dynamic stochastic block model. *Electronic Journal of Statistics*, 13(1):678–709, 2019.

[76] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015.

[77] Giulio Rossetti and Rémy Cazabet. Community discovery in dynamic networks: a survey. *ACM Computing Surveys (CSUR)*, 51(2):35, 2018.

[78] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[79] Pan Shi, Kun He, David Bindel, and John E. Hopcroft. Locally-biased spectral approximation for community detection. *Knowledge-Based Systems*, 164:459–472, 2019.

[80] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the STOC*, volume 4, 2004.

[81] Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on Computing*, 42(1):1–26, 2013.

[82] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[83] Robert C. Thompson. Principal submatrices IX: Interlacing inequalities for singular values of submatrices. *Linear Algebra and its Applications*, 5(1):1–12, 1972.

[84] Amanda L. Traud, Peter J. Mucha, and Mason A. Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012.

[85] Joel A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

[86] Pascal Vincent and Yoshua Bengio. Density-sensitive metrics and kernels. In *Snowbird Learning Workshop*, 2003.

[87] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[88] James D. Wilson, Nathaniel T. Stevens, and William H. Woodall. Modeling and detecting change in temporal networks via a dynamic degree corrected stochastic block model. *arXiv preprint arXiv:1605.04049*, 2016.

[89] James D. Wilson, Simi Wang, Peter J. Mucha, Shankar Bhamidi, Andrew B. Nobel, et al. A testing based extraction algorithm for identifying significant communities in networks. *The Annals of Applied Statistics*, 8(3):1853–1891, 2014.

[90] Eric P. Xing, Wenjie Fu, Le Song, et al. A state-space mixed membership blockmodel for dynamic network tomography. *The Annals of Applied Statistics*, 4(2):535–566, 2010.

[91] Kevin S. Xu and Alfred O. Hero, III. Dynamic stochastic blockmodels for time-evolving social networks. *IEEE Journal of Selected Topics in Signal Processing*, 8(4):552–562, 2014.

[92] Kevin S. Xu, Mark Kliger, and Alfred O. Hero, III. Adaptive evolutionary clustering. *Data Mining and Knowledge Discovery*, 28(2):304–336, 2014.

[93] Tianbao Yang, Yun Chi, Shenghuo Zhu, Yihong Gong, and Rong Jin. Detecting communities and their evolutions in dynamic social networks–a Bayesian approach. *Machine Learning*, 82(2):157–189, 2011.

[94] Ke Yin and Xue-Cheng Tai. An effective region force for some variational models for learning and clustering. *Journal of Scientific Computing*, 74(1):175–196, 2018.

[95] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *Advances in neural information processing systems*, pages 1601–1608, 2005.

[96] Yunpeng Zhao, Elizaveta Levina, and Ji Zhu. Community extraction for social networks. *Proceedings of the National Academy of Sciences*, 108(18):7321–7326, 2011.