CLUSTERING AND NETWORK ANALYSIS WITH BIOLOGICAL APPLICATIONS

KONSTANTIN VOEVODSKI

Dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

BOSTON
UNIVERSITY

BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

CLUSTERING AND NETWORK ANALYSIS WITH BIOLOGICAL APPLICATIONS

by

KONSTANTIN VOEVODSKI

B.A., Boston University, 2005

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2011

Approved by

First Reader

Shang-Hua Teng, Ph.D.
Professor of Computer Science

Second Reader

Yu Xia, Ph.D.
Assistant Professor of Chemistry

Third Reader

Steve Homer, Ph.D.
Professor of Computer Science

Acknowledgments

My experience as a graduate student at Boston University has been excellent. Having been an undergraduate here as well, I already knew a lot of the faculty when I decided to enroll in the Ph.D. program. Throughout the years I have felt at home here, and was always greeted with a smile whenever I saw professors around the building.

My interest in interdisciplinary research began as an undergraduate when I enrolled in an Undergraduate Research Opportunities Program during the summer between my junior and senior year. I was placed in the Tolan lab in the Biology department, where Professor Dean Tolan needed a programmer to implement some of their algorithms. That summer I created a small application that implemented their routines, which was used by me and the other people in the lab. Dean's enthusiasm for what I was doing encouraged me to continue my education as a computer scientist and showed me the value of interdisciplinary collaboration.

It was also during my junior year that I first met Professor Shang-Hua Teng. Shang-Hua was teaching an introductory algorithms course, and I was impressed by the elegance of many of the techniques that we studied, as well as the arguments for their correctness. During my senior year I told Shang-Hua that I was thinking about applying to the Ph.D. program at Boston University, and asked him to write a letter of recommendation for me. He told me that he was looking for another graduate student, and asked me about my research interests. I told him about my experience in the Tolan lab, and he told me that he was very interested in bioinformatics. At the time Shang-Hua was considering several other students, but I am very glad that he chose me, and he became my advisor.

Over the years Shang-Hua has been a great mentor, and has challenged me to think about many questions, a lot of which I had no good answers to. Still, he has taught me that research involves posing concrete problems and trying to solve them. I have gotten much more disciplined in my thinking, which is mainly because of him.

During my second year as a graduate student I sought the help of Professor Yu (Brandon) Xia with our bioinformatics research. Brandon was very friendly and we started to meet

on a regular basis; later he formally became my co-advisor. He has taught me how to effectively write interdisciplinary research papers, and we have had many discussions about different computational techniques and their application to bioinformatics. Brandon has helped me understand the challenges facing computational biologists, and has helped put our work in better perspective.

After my work on protein network analysis was completed, I began to search for very large networks to better apply algorithms that are designed for massive graphs. However, I soon realized that when the size of the data set is very large, building a network is quite difficult because we first have to decide which nodes are connected. This is quite challenging because in order to do so we have to perform a computation for each pair of nodes. I mentioned this issue to Shang-Hua, who at the time was working with Maria-Florina (Nina) Balcan and Heiko Röglin on a different clustering problem. Shang-Hua suggested that we think about the model that they were considering given that we do not know the distances between all the objects.

We then met with Heiko and Nina several times to discuss their model in the limited information setting. After these meetings we had some great insights, which ultimately led to the material presented in Chapter 2 of this thesis. Nina later suggested that I think about the min-sum objective in the limited information setting as well, which led to the algorithm presented in Chapter 3. I would like to thank Nina and Heiko for carefully proofreading the material in our manuscripts, a lot of which is presented in Chapters 2 and 3.

I would also like to thank the Advanced Computation in Engineering and Science training program at Boston University for giving me a two-year fellowship during my graduate studies. This fellowship allowed me to focus on research at a time when I really needed to. Moreover, Claudio Rebbi, Ilona Lappo, and Cheryl Endicott are very nice and supportive of the students and their needs. I would also like to thank Kyle Burke, another student of Shang-Hua, for being a bit of an older sibling to me while he was here. Kyle gave me some great advice about which classes to take, enthusiastically helped me prepare for my depth exam, and explained the other milestones to me.

CLUSTERING AND NETWORK ANALYSIS WITH BIOLOGICAL APPLICATIONS

(Order No.)

KONSTANTIN VOEVODSKI

Boston University, Graduate School of Arts and Sciences, 2011

Major Professor: Shang-Hua Teng, Professor of Computer Science

ABSTRACT

Clustering and network analysis are important areas of research in Computer Science and other disciplines. Clustering is broadly defined as finding sets of similar objects. It has many applications, such as finding groups of similar buyers given their product preferences, and finding groups of similar proteins given their sequences. Network analysis considers data represented by a collection of nodes (vertices), and edges that link these nodes. The structure of the network is studied to find central nodes, identify nodes that are similar to a particular vertex, and find well-connected groups of vertices. The World Wide Web and online social networks are some of the best studied networks today. Network analysis can also be applied to biological networks where nodes are proteins and edges represent relationships or interactions between them.

The size of real-world data sets presents many challenges to computational techniques that interpret them. A classic clustering problem is to divide the data set into groups, given the pairwise distances between the objects. However, computing all the pairwise distances may be infeasible if the data set is very large. In this thesis we consider clustering in a *limited information* setting where we do not know the distances between the objects in advance, and instead must query them during the execution of the algorithm. We present algorithms that find an accurate clustering in this setting using few queries.

The networks that we encounter in practice are quite large as well, making computations on the entire network difficult. In this thesis we present techniques for *locally exploring* networks, which are efficient but still give meaningful information about the local structure of the graph. We develop several tools for locally exploring a network, and show that they give meaningful

results when applied to protein networks.

Table of Contents

List of Figures

List of Abbreviations

PPI	Protein-protein interaction
BLAST	Basic Local Alignment Search Tool
Pfam	a classification database of protein families
SCOP	Structural Classification of Proteins
HMM	Hidden Markov Model
BioGRID	Biological General Repository for Interaction Datasets
VisANT	Visual Analysis Tool
AC-Western	Affinity Capture Western
Affinity Capture MS	Affinity Capture Mass Spectrometry
AC-MS	Affinity Capture Mass Spectrometry
ROC	Receiver operating characteristic
TP	True positives
FP	False positives

Chapter 1

Introduction

Designing computational techniques that work effectively on real-world data sets is challenging for several reasons. It is difficult to theoretically model all the characteristics of particular data sets, and the models that we create often involve assumptions that may not be true in practice. A lot of computational methods are also designed with no particular application in mind, and it is hard to expect them to work well on different kinds of data. Moreover, validating new methods is difficult because relevant data sets often lack a solid ground truth to compare against, and otherwise only miniature or outdated data sets are available.

This thesis presents new algorithms and tools with specific applications in the field of computational biology. It concerns both the development of algorithms and their validation on relevant data sets. Chapters 2 and 3 present novel clustering algorithms that work with limited distance information, and their application to clustering protein sequences. Our one versus all distance query models a sequence database search program such as BLAST (Basic Local Alignment Search Tool), which quickly compares a single sequence against an entire database of sequences. BLAST is used ubiquitously in boinformatics, and is known to produce very meaningful results. Our accuracy analysis involves theoretic assumptions about the approximation stability of different objective functions for clustering. We explicitly test these assumptions on protein sequence data, which clarifies whether we should expect our techniques to perform well. We also validate the performance of our methods using gold-standard classifications of protein evolutionary relatedness.

Chapter 4 of this thesis presents our tools for network analysis. While the algorithms that our tools implement are not designed specifically for biological networks, we conduct thorough experimental studies to show that they give meaningful results in protein networks. Unlike social networks, where for relevant data sets a ground truth is often built from implicit assumptions and weak associations, the proteins of model organisms such as yeast are very well-known, which gives us a solid ground truth for validation purposes. For the protein networks in our studies

we have access to a gold-standard listing of functional units, as well as a manually curated classification of the proteins, which we can use to derive meaningful functional distances. We can therefore evaluate how effective our techniques are at finding functionally related proteins.

The first part of this thesis concerns clustering in the limited information setting. Traditional algorithms require all pairwise distances between the objects as input, which may be difficult to obtain if the data set is very large. In practice computing all pairwise distances between the objects may take orders of magnitude more time than computing the actual clustering. Motivated by this observation, in this thesis we develop clustering algorithms that operate with limited distance information; in particular we consider clustering with a small number of *one versus all* queries. A one versus all query returns the distances between a specified point and all other points in the data set. We show that if the clustering instance has a certain structure, then $O(k)$ and $O(k \log k)$ one versus all queries is enough to produce an accurate clustering, where k is the number of clusters in the data.

In order to analyze the correctness of our algorithms we assume that the distance function is a metric, and that the clustering instance satisfies a natural approximation stability property of Balcan, Blum, and Gupta [BBG09] with respect to different objective functions for clustering. Balcan et al. assume that there is some relevant *target* clustering C_T, and optimizing a particular objective function for clustering (such as k-median or min-sum) gives clusterings that are structurally close to C_T. More precisely, the (c, ϵ) approximation stability property of Balcan et al. assumes that any c-approximation of the objective is ϵ-close to C_T, where the distance between two clusterings is the fraction of misclassified points under the optimum matching between the two sets of clusters. If this is true, instead of optimizing the objective function we can focus directly on finding a clustering that is structurally close to C_T.

Indeed, Balcan et al. show that given the (c, ϵ)-property we can efficiently find clusterings that are ϵ-close to C_T even when finding a c-approximation is very difficult [BBG09]. These approximation stability assumptions have also been recently studied by Awasthi, Blum, and Sheffet [ABS10], who have improved some of the results of Balcan et al. We extend the work of Balcan et al. by showing that given these assumptions it is still possible to find accurate clusterings while querying only some of the distances between the points. Moreover, we develop

algorithms that are more efficient, and show their practical use by applying them to relevant problems in computational biology.

The second part of this thesis presents our techniques for exploring networks. Instead of modeling data as a set of points with a pairwise distance function, we can often represent it as a network: a collection of nodes (vertices) and edges linking these nodes. We can then use the network topology to find out more about the nodes and the relationships between them. For social networks a centrality measure known as Alpha-Centrality has been proposed, which considers the number of paths between the nodes in the graph [Kat53, Bon87, BL01]. This centrality measure can be used to evaluate the influence of each node. Another approach to measure the importance of nodes in a network is the famous PageRank algorithm, which considers a random walk on the Web graph to rank pages by their importance [PBMW98, BP98]. The structure of the Web graph has also been used to classify Web pages as hubs and authorities [Kle98], and to find communities (clusters) of related pages [GKR98, KRRT99, FLGC02]. Community detection is also well-studied in the context of social networks [New04, GN02, PDFV05, Cla05].

In order to investigate a network, we can use two kinds of algorithms that differ in whether or not they consider the entire graph. A global algorithm performs a computation on the entire network, while a local algorithm only considers a small part of the graph close to a given vertex. A local algorithm may be much faster, but can still give meaningful information about the local structure of the network. The local clustering algorithms of Spielman and Teng [ST08], and Andersen, Chung, and Lang [ACL06] provably find high-quality clusters and have a runtime that is proportional to the size of the found cluster. The ApproximatePR algorithm, which is a subroutine of the algorithm of Andersen et al., finds the closest neighbors of a given vertex by computing an approximate personalized PageRank vector [1]. The runtime of this algorithm is proportional to the size of the neighborhood that is explored.

The same network analysis techniques used on social networks and the Web graph can also be applied to biological networks. Clusters are especially relevant in protein networks because

[1] A personalized PageRank vector generally refers to a PageRank vector with a non-uniform starting vector, but here we only consider starting vectors that are non-zero in exactly one entry. We use the same terminology for Alpha-Centrality vectors.

they often represent protein complexes or other modules with related function. There have been many studies that cluster protein-protein interaction (PPI) networks [KCY+06, BH03, SM03, CY06, KPJ04, BCM+03, VW09], all of which use some global algorithm. In this thesis we use local clustering techniques to find communities in protein networks. We build a tool that uses the algorithms of Spielman and Teng [ST08], and Andersen et al. [ACL06] to find a high-quality community near a given vertex in a network specified by the user. Our tool works very quickly on protein networks that are currently available, and easily scales to much larger networks. We conduct a thorough study in which we investigate the quality of the communities found by our local algorithms and compare them with other methods.

We also develop a measure of closeness in protein networks that uses personalized PageRank. We define the *PageRank Affinity* of two proteins a and b to be the minimum of $\mathrm{pr}(a \to b)$ and $\mathrm{pr}(b \to a)$, where $\mathrm{pr}(a \to b)$ is the amount of PageRank that b has in the personalized PageRank vector of a, which is proportional to the number of times b is visited in a random walk on the network that restarts at a. We perform a thorough study that shows that this measure of closeness is very effective at inferring functional ties between proteins. Based on our measure we build a tool that quickly finds nodes closest to a queried vertex in a network specified by the user, which uses the ApproximatePR algorithm of Andersen et al. [ACL06] as a subroutine.

In this thesis we also present a novel technique for locally exploring a graph that uses approximate personalized Alpha-Centrality vectors. We develop an algorithm to approximate Alpha-Centrality, and give its proof of correctness. Our Approximate-Centrality algorithm has only a single parameter that controls both the runtime and the quality of the produced approximation. We show that just like PageRank, Alpha-Centrality with personalized starting vectors can be used to measure the closeness of nodes in a network. Therefore we can use our algorithm to approximate a personalized Alpha-Centrality vector in order to find the closest neighbors of a given node. Our Approximate-Centrality algorithm will only explore a small part of the graph close to the starting vertex (based on the choice of the approximation parameter). We also give some intuition for when Alpha-Centrality may be more meaningful than PageRank.

1.1 Contribution

In this section we briefly summarize our contributions, which are as follows.

- In Chapter 2 we describe our *Landmark-Clustering* algorithm, which given the (c, ϵ)-property for the *k-median* objective function finds a clustering that is ϵ-close to the target by using only $O(k)$ one versus all queries. We use the same assumptions as Balcan, Blum, and Gupta [BBG09], and we obtain the same performance guarantees, but by only using a very small number of one versus all queries. In addition to handling this more difficult scenario, we also provide a much faster algorithm. The algorithm of Balcan et al. can be implemented in $O(n^3)$ time, while the one proposed here runs in time $O(nk \log n)$.

- In Chapter 3 we describe the *Landmark-Clustering-Min-Sum* algorithm, which given the (c, ϵ)-property for the *min-sum* objective function finds a clustering that is close to the target by using only $O(k \log k)$ one versus all queries. If the approximation stability property is satisfied for the *min-sum* objective, the structure of the clustering instance is quite different, and the algorithm given in Chapter 2 fails to find an accurate clustering in such cases. The min-sum objective is also considerably harder to approximate.

- We apply these algorithms to cluster proteins by sequence similarity using BLAST (Basic Local Alignment Search Tool) as the one versus all distance query, and compare our results to gold-standard manual classifications given in the Pfam [FMT+10] and SCOP [MBHC95] databases. We find that for one of these sources we obtain clusterings that usually closely match the given classification, and for the other the performance of our algorithms is comparable to that of the best known algorithms using the full distance matrix. Both of these classification databases have limited coverage, so completely automated methods such as ours can be useful in clustering proteins that have yet to be classified. Moreover, our methods can cluster very large data sets because they are efficient and do not require the full distance matrix as input, which may be infeasible to obtain for a very large data set.

- In Chapter 4 we present our tool for finding high-quality local communities in a large

network. Our application quickly finds a community close to a queried vertex in any network constructed from a large repository of protein interaction data or manually input by the user, and easily scales to very large networks. Our tool uses the local clustering algorithms Nibble [ST08] and PageRank-Nibble [ACL06], which find a cluster by exploring only a part of the graph close to the starting vertex. The quality of a cluster is measured by the ratio of the number of its outgoing edges to the sum of the degrees of its nodes, known as conductance [SJ89]. We perform an experimental study that compares the techniques that our tool uses to other partitioning algorithms. We show that among the algorithms considered, Nibble finds better clusters in terms of conductance and functional coherence.

- In Chapter 4 we present an approach to evaluate pairwise closeness in networks using personalized PageRank. We conduct a rigorous study of protein networks that shows that *PageRank Affinity* is more biologically meaningful than other commonly used measures of closeness in terms of predicting co-complex membership and correlation with functional distance. Based on our method we build a tool that quickly finds nodes closest to a queried vertex in a network input by the user.

- In Chapter 4 we describe an algorithm that approximates Alpha-Centrality, and give its proof of correctness. We show that Alpha-Centrality with personalized starting vectors can also be used to measure the closeness of nodes in a network. We can use our Approximate-Centrality algorithm to locally explore a graph by computing an approximate personalized Alpha-Centrality vector. We also give some intuition for when closeness based on personalized Alpha-Centrality is likely to give more meaningful results than *PageRank Affinity*.

1.2 Related Work

We next give a brief overview of the related work. Section 1.2.1 lists other clustering algorithms that use sampling, and describes similar techniques for choosing an initial set of points for clustering. Section 1.2.2 lists other graph partitioning algorithms, applications of PageRank, and efforts to measure closeness of nodes in protein networks.

1.2.1 Clustering with Limited Distance Information

Approximate clustering using sampling has been studied extensively in recent years [MOP01, BD07, CS07]. The methods proposed in these papers yield constant factor approximations to the k-median objective with high probability using $O(k)$ one versus all distance queries. However, these approximation algorithms may not be relevant given our approximation stability assumptions. The constant factor of these approximations is at least 2, therefore the proposed sampling methods do not necessarily yield clusterings close to the target clustering C_T if the (c, ϵ) approximation stability property holds only for some small constant $c < 2$, which is the interesting case in our setting.

A property that is related to (c, ϵ) is ϵ-separability, introduced by Ostrovsky, Rabani, Schulman, and Swamy [ORSS06]. A clustering instance is ϵ-separated if the cost of the optimal k-clustering is at most ϵ^2 times the cost of the optimal clustering using $k-1$ clusters. This property is satisfied if we have chosen k well, because it is common practice to choose the number of clusters by incrementing k until the cost of the clustering stops (significantly) decreasing. The ϵ-separability and (c, ϵ) properties are related: in the case when the clusters are large the Ostrovsky et al. condition implies the Balcan et al. condition (see [BBG09]).

Ostrovsky et al. also present a sampling method for choosing initial centers that is similar to the one used in *Landmark-Clustering*. When followed by a single Lloyd-type descent step, their technique gives a constant factor approximation of the k-means objective if the instance is ϵ-separated. However, their sampling method needs information about the full distance matrix because the probability of picking two points as two cluster centers is proportional to their squared distance. A very similar (independently proposed) strategy is used by Arthur and Vassilvitskii to obtain an $O(\log k)$-approximation of the k-means objective on arbitrary instances [AV07]. Their work was further extended by Ailon, Jaiswal, and Monteleoni to give a constant factor approximation using $O(k \log k)$ centers [AJM09]. The latter two algorithms can be implemented with k and $O(k \log k)$ one versus all distance queries, respectively.

Awasthi, Blum, and Sheffet [ABS10] have since improved the approximation guarantee of Ostrovsky et al. and some of the results of Balcan et al. In particular, they show a way to arbitrarily closely approximate the k-median and k-means objective when the Balcan et

al. condition is satisfied and all the target clusters are large. In their analysis they use a property called weak deletion-stability, which is implied by the Ostrovsky et al. condition and the Balcan et al. condition when the target clusters are large. However, in order to find a c-approximation (and given our assumption a clustering that is ϵ-close to the target) the runtime of their algorithm is $n^{O(1/(c-1)^2)}k^{O(1/(c-1))}$. On the other hand, the runtime of our *Landmark-Clustering* algorithm is completely independent of c, so it remains efficient even when the (c, ϵ)-property holds only for some very small constant c.

Our landmark selection strategy is related to the *farthest first traversal* used by Dasgupta [Das02]. In each iteration this traversal selects the point that is farthest from the ones chosen so far, where distance from a point s to a set X is given by $\min_{x \in X} d(s, x)$. This traversal was originally used by Gonzalez to give a 2-approximation to the k-center problem [Gon85]. It is used by Dasgupta to produce a hierarchical clustering where for each k the induced k-clustering is a constant factor approximation of the optimal k-center clustering [Das02]. Our selection strategy is somewhat different from farthest first traversal because in each iteration we uniformly at random choose one of the furthest points from the ones selected so far. In addition, the theoretical guarantees we provide are quite different from those of Gonzales and Dasgupta.

1.2.2 Network Analysis

Graph partitioning is often used to find clusters in a network. This approach seeks to divide the nodes of the network into clusters such that there are many within-cluster edges but few between-cluster edges. Spectral methods partition the graph by using the eigenvectors of the adjacency or Laplacian matrices of the graph [MS01, KVV00, HK92, ST07, Kel06, BLR08, AKY99b, AKY99a]. Metis is another effective graph partitioning algorithm. It is very efficient and works better than commonly used spectral clustering methods in terms of the size of the resulting edge cut [AK06].

PageRank with personalized starting vectors was introduced by Haveliwala [Hav03], and has been used for context-sensitive search on the Web [FR04, JW03]. PageRank has also been used for biological applications [MBHG05], and personalized PageRank has been applied to protein

networks by Can et al. [CcS05] and Chipman and Singh [CS09].

There has been considerable work done in evaluating pairwise closeness in PPI networks. Measures of interconnectedness between protein pairs have been used to find functionally similar proteins [YH07, CSW06, OKA05, SL03]. Different notions of interconnectedness have also been used to predict false negative interactions in protein networks [GR03]. All of these measures consider the density of the interactions in the immediate neighborhood of two proteins, and some also normalize by the number of interactions of each protein, or the number of interactions in the neighborhood expected by chance.

A problem that is related to evaluating the closeness of two proteins in a PPI network is finding the closest neighbors of a set of proteins. This is addressed by Li and Horvath [LH07] by generalizing pairwise notions of interconnectedness, and by Can et al. [CcS05] by using personalized PageRank. A similar problem is considered in the context of probabilistic PPI networks, where reachability [AKGR04] and shortest path distance [HZRB07] in instantiated networks are used to recover protein complexes when only some of their proteins are known.

Chapter 2

Clustering with Limited Distance Information

Clustering from pairwise distance information is a well-studied problem with many applications. Traditional clustering algorithms require all pairwise distances between the points as input, which may be infeasible to compute in practice. Here we consider clustering in a *limited information* setting. We assume that the distances between the points are not given in advance, and must be queried during the execution of the algorithm. Our objective is to find an accurate clustering using few queries.

We can imagine at least two different ways to query distances between points. One way is to ask for distances between pairs of points, and the other is to ask for distances between one point and all other points. Clearly, a one versus all query can be implemented as n pairwise queries, where n is the size of the data set, but we draw a distinction between the two because the former is often significantly faster in practice if the query is implemented as a database search.

Our main motivating example for considering one versus all distance queries is sequence similarity search in biology. A program such as BLAST [AGM+90] (Basic Local Alignment Search Tool) is optimized to search a single sequence against an entire database of sequences. On the other hand, performing n pairwise sequence alignments takes several orders of magnitude more time, even if the pairwise alignment is very fast. The disparity in runtime is due to the hashing that BLAST uses to identify regions of similarity between the input sequence and sequences in the database. The program maintains a hash table of all *words* in the database (substrings of a certain length), linking each word to its locations. When a query is performed, BLAST considers each word in the input sequence, and runs a local sequence alignment in each of its locations in the database. Therefore the program only performs a limited number of local sequence alignments, rather than aligning the input sequence to each sequence in the database. Of course, the downside is that we never consider alignments between sequences that do not share a word. However, in this case an alignment may not be relevant anyway, and we

can assign a distance of infinity to the two sequences. Even though the search performed by BLAST is heuristic, it has been shown that protein sequence similarity identified by BLAST is meaningful [BCH98].

Motivated by such scenarios, we consider the problem of clustering a data set with an unknown distance function, given only the capability to ask one versus all distance queries. We design efficient algorithms for clustering accurately with a small number of such queries. We analyze the accuracy of our algorithms in the framework of Balcan, Blum and Gupta [BBG09], which assumes that the clustering instance satisfies an approximation stability property with respect to some objective function for clustering. In particular, we consider approximation stability with respect to the *k-median* and *min-sum* objective functions. For the *k-median* objective we give an algorithm that finds an accurate clustering using a number of queries that is linear in the number of clusters.

This chapter is organized as follows. In Section 2.1 we formally define our problem, the k-median objective function, and the (c, ϵ) approximation stability property of Balcan, Blum, and Gupta. We also introduce some notation that is used in the analysis of our algorithm. Section 2.2 gives a high-level description of our *Landmark-Clustering* algorithm and states a theorem about its correctness. The analysis of our algorithm and its proof of correctness is given in Section 2.3. An efficient implementation of our procedure is given in Section 2.4. Finally, Section 2.5 describes the application of our algorithm to clustering protein sequences. We also include a discussion on setting the parameters of our procedure, and testing the approximation stability assumption on protein sequence data sets.

2.1 Approximation Stability of the k-median Objective Function

Given a metric space $M = (X, d)$ with point set X, an unknown distance function d satisfying the triangle inequality, and a set of points $S \subseteq X$, we would like to find a k-clustering C that partitions the points in S into k sets $C_1, \ldots, C_k$ by using *one versus all* distance queries.

In our analysis we assume that S satisfies the (c, ϵ)-property of [BBG09] for the k-median objective function. The k-median objective is to minimize $\Phi(C) = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, c_i)$, where c_i is the median of cluster C_i, which is the point $y \in C_i$ that minimizes $\sum_{x \in C_i} d(x, y)$. Let

$\mathrm{OPT}_\Phi = \min_C \Phi(C)$, where the minimum is over all k-clusterings of S, and denote by $C^* = \{C_1^*, \ldots, C_k^*\}$ a clustering achieving this value.

To formalize the (c, ϵ)-property we need to define a notion of distance between two k-clusterings $C = \{C_1, \ldots, C_k\}$ and $C' = \{C_1', \ldots, C_k'\}$. As in [BBG09], we define the distance between C and C' as the fraction of points on which they disagree under the optimal matching of clusters in C to clusters in C':

$$\mathrm{dist}(C, C') = \min_{\sigma \in S_k} \frac{1}{n} \sum_{i=1}^{k} |C_i - C'_{\sigma(i)}|,$$

where S_k is the set of bijections $\sigma\colon \{1, \ldots, k\} \to \{1, \ldots, k\}$. Two clusterings C and C' are *ϵ-close* if $\mathrm{dist}(C, C') < \epsilon$.

We assume that there exists some unknown "target" clustering C_T and given a proposed clustering C we define the error of C with respect to C_T as $\mathrm{dist}(C, C_T)$. Our goal is to find a clustering of low error.

For any objective function Ω we use OPT_Ω to denote its optimum objective value. The (c, ϵ) approximation stability property is defined as follows.

Definition 2.1. *We say that the instance (S, d) satisfies the (c, ϵ)-property for objective function Ω with respect to the target clustering C_T if any clustering of S that approximates OPT_Ω within a factor of c is ϵ-close to C_T, that is, $\Omega(C) \leq c \cdot \mathrm{OPT}_\Omega \Rightarrow \mathrm{dist}(C, C_T) < \epsilon$.*

Here we assume that the clustering instance satisfies the (c, ϵ)-property for the k-median objective function. In the analysis of the next section we denote by c_i^* the center point of C_i^*, and use OPT to refer to the value of C^* using the k-median objective, that is, $\mathrm{OPT} = \Phi(C^*)$. We define the *weight* of point x to be the contribution of x to the k-median objective in C^*: $w(x) = \min_i d(x, c_i^*)$. Similarly, we use $w_2(x)$ to denote x's distance to the second-closest cluster center among $\{c_1^*, c_2^*, \ldots, c_k^*\}$. In addition, let w be the average weight of the points: $w = \frac{1}{n} \sum_{x \in S} w(x) = \frac{\mathrm{OPT}}{n}$, where n is the cardinality of S.

2.2 Landmark-Clustering

Algorithm 2.1 Landmark-Clustering$(S, \alpha, \epsilon, \delta, k)$

$b = (1 + 17/\alpha)\epsilon n$;
$q = 2b$;
iter $= 4k + 16 \ln \frac{1}{\delta}$;
$s_{\min} = b + 1$;
$n' = n - b$;
$L =$ **Landmark-Selection**(q, iter);
$C' =$ **Expand-Landmarks** $(s_{\min}, n', L)$;
Choose some landmark l_i from each cluster C'_i;
for each $x \in S$ **do**
  Insert x into the cluster C''_j for $j = \text{argmin}_i d(x, l_i)$;
end for
return C'';

In this section we present a new algorithm that accurately clusters a set of points assuming that the clustering instance satisfies the (c, ϵ)-property for $c = 1 + \alpha$, and the clusters in the target clustering C_T are not too small. The algorithm presented here is much faster than the one given by Balcan, Blum, and Gupta [BBG09] and does not require all pairwise distances as input. Instead, we only require $O(k + \ln \frac{1}{\delta})$ one versus all distance queries to achieve the same performance guarantee as in [BBG09] with probability $1 - \delta$.

Our clustering method is described in Algorithm 2.1. We start by using the *Landmark-Selection* procedure to select a small set of landmarks. This procedure repeatedly chooses uniformly at random one of the q furthest points from the ones selected so far, for an appropriate q. We use $d_{\min}(s)$ to refer to the minimum distance between s and any point selected so far. Each time we select a new landmark l, we use a one versus all distance query to get the distances between l and all other points in the data set, and update $d_{\min}(s)$ for each point $s \in S$. To select a new landmark in each iteration, we choose a random number $i \in \{n - q + 1, \ldots, n\}$ and use a linear time selection algorithm to select the ith furthest point. We note that our algorithm only uses the distances between landmarks and other points to produce a clustering.

Algorithm 2.2 Landmark-Selection(q, iter)

Choose $l \in S$ uniformly at random;
$L = \{l\}$;
for each $d(l, s) \in$ QUERY-ONE-VS-ALL(l, S) **do**
  $d_{\min}(s) = d(l, s)$;
end for
for $i = 1$ to iter $- 1$ **do**
  Let $s_1, ..., s_n$ be an ordering of the points in S such that $d_{\min}(s_i) \leq d_{\min}(s_{i+1})$ for $i \in \{1, \ldots, n-1\}$;
  Choose $l \in \{s_{n-q+1}, \ldots, s_n\}$ uniformly at random;
  $L = L \cup \{l\}$;
  for each $d(l, s) \in$ QUERY-ONE-VS-ALL(l, S) **do**
    if $d(l, s) < d_{\min}(s)$ **then**
      $d_{\min}(s) = d(l, s)$;
    end if
  end for
end for
return L;

Expand-Landmarks then expands a ball B_l around each landmark $l \in L$ chosen by *Landmark-Selection*. We use the variable r to denote the radius of all the balls: $B_l = \{s \in S \mid d(s, l) \leq r\}$ for all $l \in L$. The algorithm starts with $r = 0$, and increments it until the balls satisfy a property described below. For each B_l there are n relevant values of r to try, each adding one more point to B_l, which results in at most $|L|n$ values to try in total.

The algorithm maintains a graph $G_B = (V_B, E_B)$, where vertices correspond to balls that have at least $s_{\min}$ points in them, and two vertices are connected by an (undirected) edge if the corresponding balls overlap on any point: $(v_{l_1}, v_{l_2}) \in E_B$ iff $B_{l_1} \cap B_{l_2} \neq \emptyset$. In addition, we maintain the set of points in these balls Clustered $= \{s \in S \mid \exists l\colon s \in B_l\}$ and a list of the connected components of G_B, which we refer to as Components$(G_B) = \{\text{Comp}_1, ..., \text{Comp}_m\}$.

In each iteration, after we expand one of the balls by a point, we update G_B, Components(G_B), and Clustered. If G_B has exactly k components, and |Clustered| $\geq n'$, we terminate and report points in balls that are part of the same component in G_B as distinct clusters. If this condition is never satisfied, we report **no-cluster**. A sketch of the algorithm is given below. We use (l^*, s^*) to refer to the next landmark-point pair that is considered, corresponding to expanding

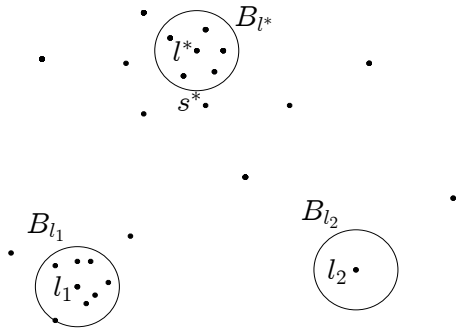


Figure 2.1: Balls around landmarks are displayed, with the next point to be added to a ball labeled as s^*.

B_{l^*} to include s^* (Figure 2.1).

Algorithm 2.3 Expand-Landmarks($s_{\min}, n', L$)

```
while ((l*, s*) = Expand-Ball()) != null do
    r = d(l*, s*);
    update G_B, Components(G_B), and Clustered
    if |Components(G_B)| = k and |Clustered| ≥ n' then
        return C = {C_1, ..., C_k} where C_i = {s ∈ S | ∃l: s ∈ B_l and v_l ∈ Comp_i}.
    end if
end while
return no-cluster;
```

The last step of our algorithm takes the clustering C' returned by *Expand-Landmarks* and improves it. We compute a set L' that contains exactly one landmark from each cluster $C'_i \in C'$ (any landmark is sufficient), and assign each point $x \in S$ to the cluster corresponding to the closest landmark in L'.

We now present our main theoretical guarantee for Algorithm 2.1.

Theorem 2.2. *Given a metric space $M = (X, d)$, where d is unknown, and a set of points S, if the instance (S, d) satisfies the $(1+\alpha, \epsilon)$-property for the k-median objective function and if each cluster in the target clustering C_T has size at least $(4+51/\alpha)\epsilon n$, then Landmark-Clustering outputs a clustering that is ϵ-close to C_T with probability $1-\delta$ in time $O((k + \ln \frac{1}{\delta})|S| \log |S|)$ using $O(k + \ln \frac{1}{\delta})$* one versus all *distance queries.*

2.3 Algorithm Analysis

Before we prove the theorem, we will introduce some notation and use an analysis similar to the one in [BBG09] to argue about the structure of the clustering instance that follows from our approximation stability assumption.

2.3.1 Structure of the Clustering Instance

Let $\epsilon^* = \text{dist}(C_T, C^*)$. By our assumption that the k-median clustering of S satisfies the $(1+\alpha, \epsilon)$-property we have $\epsilon^* < \epsilon$. Since each cluster in the target clustering has at least $(4 + 51/\alpha)\epsilon n$ points, and the *optimal k-median clustering* C^* differs from the target clustering by $\epsilon^* n \leq \epsilon n$ points, each cluster in C^* must have at least $(3 + 51/\alpha)\epsilon n$ points.

Let us define the *critical distance* $d_{\text{crit}} = \frac{\alpha w}{17\epsilon}$. We call a point x *good* if both $w(x) < d_{\text{crit}}$ and $w_2(x) - w(x) \geq 17 d_{\text{crit}}$, else x is called *bad*. In other words, the *good* points are those points that are close to their own cluster center and far from any other cluster center. In addition, we will break up the *good* points into *good sets* X_i, where X_i is the set of the *good* points in the optimal cluster C_i^*. So each set X_i is the "core" of the optimal cluster C_i^*.

Note that the distance between two points $x, y \in X_i$ satisfies $d(x, y) \leq d(x, c_i^*) + d(c_i^*, y) = w(x) + w(y) < 2 d_{\text{crit}}$. In addition, the distance between any two points in different good sets is greater than $16 d_{\text{crit}}$. To see this, consider a pair of points $x \in X_i$ and $y \in X_{j \neq i}$. The distance from x to y's cluster center c_j^* is at least $17 d_{\text{crit}}$. By the triangle inequality, $d(x, y) \geq d(x, c_j^*) - d(y, c_j^*) > 17 d_{\text{crit}} - d_{\text{crit}} = 16 d_{\text{crit}}$.

If the k-median instance (M, S) satisfies the $(1+\alpha, \epsilon)$-property with respect to C_T, and each cluster in C_T has size at least $2\epsilon n$, then

1. less than $(\epsilon - \epsilon^*)n$ points $x \in S$ on which C_T and C^* agree have $w_2(x) - w(x) < \frac{\alpha w}{\epsilon}$.

2. at most $17\epsilon n/\alpha$ points $x \in S$ have $w(x) \geq \frac{\alpha w}{17\epsilon}$.

The first part is proved by [BBG09]. The intuition is that if too many points on which C_T and C^* agree are close enough to the second-closest center among $\{c_1^*, c_2^*, \ldots, c_k^*\}$, then we can move them to the clusters corresponding to those centers, producing a clustering that is

far from C_T, but whose objective value is close to OPT, violating the $(1+\alpha, \epsilon)$-property. The second part follows from the fact that $\sum_{x \in S} w(x) = OPT = wn$.

Then using these facts it follows that at most $\epsilon^* n + (\epsilon - \epsilon^*)n + 17\epsilon n/\alpha = \epsilon n + 17\epsilon n/\alpha = (1 + 17/\alpha)\epsilon n = b$ points are bad. Hence each $|X_i| = |C_i^* \backslash B| \geq (2 + 34/\alpha)\epsilon n = 2b$.

In the remainder of this section we prove that given this structure of the clustering instance, *Landmark-Clustering* finds an accurate clustering. We first show that almost surely the set of landmarks returned by *Landmark-Selection* has the property that each of the cluster cores has a landmark near it. We then argue that given a set of landmarks with this property, *Expand-Landmarks* finds a partition C' that clusters most of the points in each core correctly. We conclude with the proof of the theorem, which argues that the clustering returned by the last step of our procedure is a further improved clustering that is very close to C^* and C_T.

2.3.2 Proof of Theorem 2.2

The *Landmark-Clustering* algorithm first uses *Landmark-Selection*(q, iter) to choose a set of landmark points. The following lemma proves that for an appropriate choice of q after selecting only iter $= O(k + \ln \frac{1}{\delta})$ landmarks with probability at least $1 - \delta$ there is a landmark closer than $2d_{\text{crit}}$ to some point in each good set.

Lemma 2.3. *Given L = Landmark-Selection $(2b, 4k + 16 \ln \frac{1}{\delta})$, with probability at least $1 - \delta$ there is a landmark closer than $2d_{crit}$ to some point in each good set.*

Proof. Because there are at most b bad points and in each iteration we uniformly at random choose one of $2b$ points, the probability that a good point is added to L is at least $1/2$ in each iteration. Using a Chernoff bound we show that the probability that fewer than k good points have been added to L after $t > 2k$ iterations is less than $e^{-t(1-\frac{2k}{t})^2/4}$ (Lemma 2.4). For $t = 4k + 16 \ln \frac{1}{\delta}$

$$e^{-t(1-\frac{2k}{t})^2/4} < e^{-(4k+16\ln\frac{1}{\delta})0.5^2/4} < e^{-16\ln\frac{1}{\delta}/16} = \delta.$$

Therefore after $t = 4k + 16 \ln \frac{1}{\delta}$ iterations this probability is smaller than δ.

We argue that once we select k good points using our procedure, one of them must be closer than $2d_{\text{crit}}$ to some point in each good set. Note that the selected good points must be distinct because we must have chosen at least k good points after $b + k$ iterations and we cannot choose the same point twice in the first $n - 2b$ iterations. There are two possibilities regarding the first k good points added to L: they are either selected from distinct good sets, or at least two of them are selected from the same good set.

If the former is true then the statement trivially holds. If the latter is true, consider the first time that a second point is chosen from the same good set X_i. Let us call these two points x and y, and assume that y is chosen after x. The distance between x and y must be less than $2d_{\text{crit}}$ because they are in the same good set. Therefore when y is chosen, $\min_{l \in L} d(l, y) \leq d(x, y) < 2d_{\text{crit}}$. Moreover, y is chosen from $\{s_{n-2b+1}, ..., s_n\}$, where $\min_{l \in L} d(l, s_i) \leq \min_{l \in L} d(l, s_{i+1})$. Therefore when y is chosen, at least $n-2b+1$ points $s \in S$ (including y) satisfy $\min_{l \in L} d(l, s) \leq \min_{l \in L} d(l, y) < 2d_{\text{crit}}$. Since each good set satisfies $|X_i| \geq 2b$, it follows that there must be a landmark closer than $2d_{crit}$ to some point in each good set.

□

Lemma 2.4. *The probability that fewer than k good points have been chosen as landmarks after $t > 2k$ iterations of Landmark-Selection is less than $e^{-t(1-\frac{2k}{t})^2/4}$.*

Proof. Let X_i be an indicator random variable defined as follows: $X_i = 1$ if point chosen in iteration i is a good point, and 0 otherwise. Let $X = \sum_{i=1}^{t} X_i$, and μ be the expectation of X. In other words, X is the number of good points chosen after t iterations of the algorithm, and μ is its expected value.

Because in each round we uniformly at random choose one of $2b$ points and there are at most b bad points in total, $\mathrm{E}[X_i] \geq 1/2$ and hence $\mu \geq t/2$. By the Chernoff bound, for any $\delta > 0$, $\Pr[X < (1-\delta)\mu] < e^{-\mu\delta^2/2}$.

If we set $\delta = 1 - \frac{2k}{t}$, we have $(1-\delta)\mu = (1-(1-\frac{2k}{t}))\mu \geq (1-(1-\frac{2k}{t}))t/2 = k$. Assuming that $t > 2k$, it follows that $\Pr[X < k] \leq \Pr[X < (1-\delta)\mu] < e^{-\mu\delta^2/2} = e^{-\mu(1-\frac{2k}{t})^2/2} \leq e^{-t/2(1-\frac{2k}{t})^2/2}$. □

The algorithm then uses the *Expand-Landmarks* procedure to find a k-clustering C'. The following lemma states that C' is an accurate clustering, and has an additional property that is relevant for the last part of the algorithm.

Lemma 2.5. *Given a set of landmarks L chosen by Landmark-Selection so that the condition in Lemma 2.3 is satisfied, Expand-Landmarks$(b+1, n-b, L)$ returns a k-clustering $C' = \{C'_1, C'_2, \ldots C'_k\}$ in which each cluster contains points from a distinct good set X_i. If we let σ be a bijection mapping each good set X_i to the cluster $C'_{\sigma(i)}$ containing points from X_i, the distance between c^*_i and any landmark l in $C'_{\sigma(i)}$ satisfies $d(c^*_i, l) < 5d_{\text{crit}}$.*

Proof. Lemma 2.6 argues that since the good sets X_i are well-separated, for $r < 4d_{\text{crit}}$ no ball of radius r can overlap more than one X_i, and two balls that overlap different X_i cannot share any points. Moreover, since we only consider balls that have more than b points in them, and the number of bad points is at most b, each ball in G_B must overlap some good set. Lemma 2.7 argues that since there is a landmark near each good set, there is a value of $r^* < 4d_{\text{crit}}$ such that each X_i is contained in some ball around a landmark of radius r^*. We can use these facts to argue for the correctness of the algorithm.

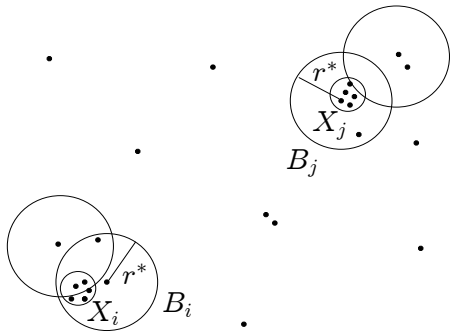


Figure 2.2: Balls B_i and B_j of radius r^* are shown, which contain good sets X_i and X_j, respectively. The radius of the balls is small in comparison to the distance between the good sets.

First we observe that for $r = r^*$, G_B has exactly k components and each good set X_i is contained within a distinct component. Each ball in G_B overlaps with some X_i, and by Lemma 2.6, since $r^* < 4d_{\text{crit}}$, we know that each ball in G_B overlaps with exactly one X_i. From Lemma 2.6 we also know that balls that overlap different X_i cannot share any points and are thus not connected in G_B. Therefore balls that overlap different X_i will be in different components in G_B. Moreover, by Lemma 2.7 each X_i is contained in some ball of radius r^*. For each good set X_i let us designate by B_i a ball that contains all the points in X_i (Figure 2.2), which is in G_B since the size of each good set satisfies $|X_i| > b$. Any ball in G_B that overlaps X_i will be connected to B_i, and will thus be in the same component as B_i. Therefore for $r = r^*$, G_B has exactly k components, one for each good set X_i that contains all the points in X_i.

Since there are at least $n-b$ good points that are in some X_i, this means that for $r = r^*$ the number of points that are in some ball in G_B (which are in Clustered) is at least $n - b$. Hence the condition in line 4 of *Expand-Landmarks* will be satisfied and the algorithm will terminate and return a k-clustering in which each cluster contains points from a distinct good set X_i.

Now let us suppose that we start with $r = 0$. Consider the first value of $r = r'$ for which the condition in line 4 is satisfied. At this point G_B has exactly k components and the number of points that are not in these components is at most b. It must be the case that $r' \leq r^* < 4d_{\text{crit}}$ because we know that the condition is satisfied for $r = r^*$, and we are considering all relevant values of r in ascending order. As before, each ball in G_B must overlap some good set X_i. Again using Lemma 2.6 we argue that since $r < 4d_{\text{crit}}$, no ball can overlap more than one X_i and two balls that overlap different X_i cannot share any points. It follows that each component of G_B contains points from a single X_i (so we cannot merge the good sets). Moreover, since the size of each good set satisfies $|X_i| > b$, and there are at most b points left out of G_B, each component must contain points from a distinct X_i (so we cannot split the good sets). Thus we will return a k-clustering in which each cluster contains points from a distinct good set X_i.

To prove the second part of the statement, let σ be a bijection matching each good set X_i to the cluster $C'_{\sigma(i)}$ containing points from X_i. Clearly, $C'_{\sigma(i)}$ is made up of points in balls of radius $r < 4d_{\text{crit}}$ that overlap X_i. Consider any such ball B_l around landmark l and let s^* denote any point on which B_l and X_i overlap. By the triangle inequality, the distance between c_i^* and l satisfies $d(c_i^*, l) \leq d(c_i^*, s^*) + d(s^*, l) < d_{\text{crit}} + r < 5d_{\text{crit}}$. Therefore the distance between c_i^* and any landmark $l \in C'_{\sigma(i)}$ satisfies $d(c_i^*, l) < 5d_{\text{crit}}$. □

Lemma 2.6. *A ball of radius $r < 4d_{\text{crit}}$ cannot contain points from more than one good set X_i, and two balls of radius $r < 4d_{\text{crit}}$ that overlap different X_i cannot share any points.*

Proof. To prove the first part, consider a ball B_l of radius $r < 4d_{\text{crit}}$ around landmark l. In other words, $B_l = \{s \in S \mid d(s, l) \leq r\}$. If B_l overlaps more than one good set, then it must have at least two points from different good sets $x \in X_i$ and $y \in X_j$. By the triangle inequality it follows that $d(x, y) \leq d(x, l) + d(l, y) \leq 2r < 8d_{\text{crit}}$. However, we know that $d(x, y) > 16d_{\text{crit}}$, giving a contradiction.

To prove the second part, consider two balls B_{l_1} and B_{l_2} of radius $r < 4d_{\text{crit}}$ around landmarks l_1 and l_2. In other words, $B_{l_1} = \{s \in S \mid d(s, l_1) \leq r\}$, and $B_{l_2} = \{s \in S \mid d(s, l_2) \leq r\}$. Assume that they overlap with different good sets X_i and X_j: $B_{l_1} \cap X_i \neq \emptyset$ and $B_{l_2} \cap X_j \neq \emptyset$. For the purpose of contradiction, let's assume that B_{l_1} and B_{l_2} share at least one point: $B_{l_1} \cap B_{l_2} \neq \emptyset$, and use s^* to refer to this point. By the triangle inequality, it follows that the distance between any point $x \in B_{l_1}$ and $y \in B_{l_2}$ satisfies $d(x, y) \leq d(x, s^*) + d(s^*, y) \leq [d(x, l_1) + d(l_1, s^*)] + [d(s^*, l_2) + d(l_2, y)] \leq 4r < 16d_{\text{crit}}$.

Since B_{l_1} overlaps with X_i and B_{l_2} overlaps with X_j, it follows that there is a pair of points $x \in X_i$ and $y \in X_j$ such that $d(x, y) < 16d_{\text{crit}}$, a contradiction. Therefore if B_{l_1} and B_{l_2} overlap different good sets, $B_{l_1} \cap B_{l_2} = \emptyset$. □

Lemma 2.7. *Given a set of landmarks L chosen by Landmark-Selection so that the condition in Lemma 2.3 is satisfied, there is some value of $r^* < 4d_{\text{crit}}$ such that each X_i is contained in some ball B_l around landmark $l \in L$ of radius r^*.*

Proof. For each good set X_i choose a point $s_i \in X_i$ and a landmark $l_i \in L$ that satisfy $d(s_i, l_i) < 2d_{\text{crit}}$. The distance between l_i and each point $x \in X_i$ satisfies $d(l_i, x) \leq d(l_i, s_i) + d(s_i, x) < 2d_{\text{crit}} + 2d_{\text{crit}} = 4d_{\text{crit}}$.

Consider $r^* = \max_{l_i} \max_{x \in X_i} d(l_i, x)$. Clearly, each X_i is contained in a ball B_{l_i} of radius r^* and $r^* < 4d_{\text{crit}}$. □

Given Lemma 2.3 and Lemma 2.5 we are now ready to prove Theorem 2.2.

Proof. After using Landmark-Selection to choose $O(k + \ln \frac{1}{\delta})$ points, with probability at least $1 - \delta$ there is a landmark closer than $2d_{\text{crit}}$ to some point in each good set. Given a set of landmarks with this property, each cluster in the clustering $C' = \{C'_1, C'_2, \ldots C'_k\}$ output by

Expand-Landmarks contains points from a distinct good set X_i. This clustering can exclude up to b points, all of which may be good. Nonetheless, this means that C' may disagree with C^* on only the bad points and at most b good points. The number of points that C' and C^* disagree on is therefore at most $2b = O(\epsilon n/\alpha)$. Thus, C' is at least $O(\epsilon/\alpha)$-close to C^*, and at least $O(\epsilon/\alpha + \epsilon)$-close to C_T.

Moreover, C' has an additional property that allows us to find a clustering that is ϵ-close to C_T. If we use σ to denote a bijection mapping each good set X_i to the cluster $C'_{\sigma(i)}$ containing points from X_i, any landmark $l \in C'_{\sigma(i)}$ is closer than $5d_{\text{crit}}$ to c_i^*. We can use this observation to find all points that satisfy one of the properties of the good points: points x such that $w_2(x) - w(x) \geq 17d_{\text{crit}}$. Let us call these points the *detectable* points. To clarify, the detectable points are those points that are much closer to their own cluster center than to any other cluster center in C^*, and the *good* points are a subset of the detectable points that are also very close to their own cluster center.

To find the detectable points using C', we choose some landmark l_i from each C'_i. For each point $x \in S$, we then insert x into the cluster C''_j for $j = \text{argmin}_i d(x, l_i)$. Lemma 2.8 argues that each detectable point in C_i^* is closer to every landmark in $C'_{\sigma(i)}$ than to any landmark in $C'_{\sigma(j\neq i)}$. It follows that C'' and C^* agree on all the detectable points. Since there are fewer than $(\epsilon - \epsilon^*)n$ points on which C_T and C^* agree that are not detectable, it follows that $\text{dist}(C'', C_T) < (\epsilon - \epsilon^*) + \text{dist}(C_T, C^*) = (\epsilon - \epsilon^*) + \epsilon^* = \epsilon$.

Therefore using $O(k + \ln \frac{1}{\delta})$ landmarks we get an accurate clustering with probability at least $1 - \delta$. The runtime of *Landmark-Selection* is $O(|L|n)$, where $|L|$ is the number of landmarks. Using a min-heap to store all landmark-point pairs and a disjoint-set data structure to keep track of the connected components of G_B, *Expand-Landmarks* can be implemented in $O(|L|n \log n)$ time. A detailed description of this implementation is given in the next section. The last part of our procedure takes $O(kn)$ time, so the runtime of our implementation is $O(|L|n \log n)$. Therefore to get an accurate clustering with probability $1 - \delta$ the runtime of our algorithm is $O((k + \ln \frac{1}{\delta})n \log n)$. Moreover, we only consider the distances between the landmarks and other points, so we only use $O(k + \ln \frac{1}{\delta})$ one versus all distance queries. □

Lemma 2.8. *Suppose the distance between c_i^* and any landmark l in $C'_{\sigma(i)}$ satisfies $d(c_i^*, l) < 5d_{\text{crit}}$. Then given point $x \in C_i^*$ that satisfies $w_2(x) - w(x) \geq 17d_{\text{crit}}$, for any $l_1 \in C'_{\sigma(i)}$ and $l_2 \in C'_{\sigma(j\neq i)}$ it must be the case that $d(x, l_1) < d(x, l_2)$.*

Proof. We will show that $d(x, l_1) < w(x) + 5d_{\text{crit}}$ **(1)**, and $d(x, l_2) > w(x) + 12d_{\text{crit}}$ **(2)**. This implies that $d(x, l_1) < d(x, l_2)$.

To prove **(1)**, by the triangle inequality $d(x, l_1) \leq d(x, c_i^*) + d(c_i^*, l_1) = w(x) + d(c_i^*, l_1) < w(x) + 5d_{\text{crit}}$. To prove **(2)**, by the triangle inequality $d(x, l_2) \geq d(x, c_j^*) - d(l_2, c_j^*)$. Since $d(x, c_j^*) \geq w_2(x)$ and $d(l_2, c_j^*) < 5d_{\text{crit}}$ we have

$$d(x, l_2) > w_2(x) - 5d_{\text{crit}}. \tag{2.1}$$

Moreover, since $w_2(x) - w(x) \geq 17d_{\text{crit}}$ we have

$$w_2(x) \geq 17d_{\text{crit}} + w(x). \tag{2.2}$$

Combining Equation 2.1 and Equation 2.2 it follows that $d(x, l_2) > (17d_{cri} + w(x)) - 5d_{\text{crit}} = w(x) + 12d_{\text{crit}}$. □

2.4 Implementation of Expand-Landmarks

A detailed description of our implementation is given in Algorithm 2.4. In order to efficiently expand balls around landmarks, we build a min-heap H of landmark-point pairs (l, s), where the key of each pair is the distance between l and s. In each iteration we find $(l^*, s^*) = H$.deleteMin(), and then add s^* to items(l^*), which stores the points in B_{l^*}. We store points that have been clustered (points in balls of size larger than $s_{\min}$) in the set Clustered.

Our implementation assigns each clustered point s to a "representative" landmark, denoted by $l(s)$. The representative landmark of s is the landmark l of the first large ball B_l that contains s. To efficiently update the components of G_B, we maintain a disjoint-set data structure U that contains sets corresponding to the connected components of G_B, where each ball B_l is represented by landmark l. In other words, U contains a set $\{l_1, l_2, \ldots, l_i\}$ iff $B_{l_1}, B_{l_2}, \ldots, B_{l_i}$ form a connected component in G_B.

For each large ball B_l our algorithm considers all points $s \in B_l$ and performs Update-Components(l, s), which works as follows. If s does not have a representative landmark we assign it to l, otherwise s must already be in $B_{l(s)}$, and we assign B_l to the same component as $B_{l(s)}$. If none of the points in B_l are assigned to other landmarks, it will be in its own component.

Algorithm 2.4 Expand-Landmarks($s_{\min}, n', L$)

```
1: A = ();
2: for each s ∈ S do
3:    l(s) = null;
4:    for each l ∈ L do
5:       A.add((l, s), d(l, s));
6:    end for
7: end for
8: H = build-heap(A);
9: for each l ∈ L do
10:   items(l) = ();
11: end for
12: Set Clustered = ();
13: U = ();
14: while H.hasNext() do
15:   (l*, s*) = H.deleteMin();
16:   items(l*).add(s*);
17:   if items(l*).size() == s_min then
18:      Activate(l*);
19:   end if
20:   if items(l*).size() > s_min then
21:      Update-Components(l*, s*);
22:      Clustered.add(s*);
23:   end if
24:   if Clustered.size() ≥ n' and U.size() == k then
25:      return Format-Clustering();
26:   end if
27: end while
28: return no-cluster;
```

Algorithm 2.5 Update-Components(l, s)

```
1: if l(s) == null then
2:    l(s) = l;
3: else
4:    c1 = U.find(l);
5:    c2 = U.find(l(s));
6:    U.union(c1, c2);
7: end if
```

Algorithm 2.6 Activate(l)

```
1: U.MakeSet(l);
2: for each s ∈ items(l) do
3:    Update-Components(l, s);
4:    Clustered.add(s);
5: end for
```

Algorithm 2.7 Format-Clustering()

```
1: C = ();
2: for each Set L in U do
3:    Set Cluster = ();
4:    for each l ∈ L do
5:       for each s ∈ items(l) do
6:          Cluster.add(s);
7:       end for
8:    end for
9:    C.add(Cluster);
10: end for
11: return C;
```

During the execution of the algorithm the connected components of G_B correspond to the sets of U (where each ball B_l is represented by landmark l). Suppose that B_{l_1} and B_{l_2} are connected in G_B, then B_{l_1} and B_{l_2} must overlap on some point s. Without loss of generality, suppose s is added to B_{l_1} before it is added to B_{l_2}. When s is added to B_{l_1}, $l(s) = l_1$ if s does not yet have a representative landmark (lines 1-2 of Update-Components), or $l(s) = l'$ and both l_1 and l' are put in the same set (lines 4-6 of Update-Components). When s is added to B_{l_2}, if $l(s) = l_1$, then l_1 and l_2 will be put in the same set. If $l(s) = l'$, l' and l_2 will be put in the same set, which also contains l_1.

It follows that whenever B_{l_1} and B_{l_2} are in the same connected component in G_B, l_1 and l_2 will be in the same set in U. Moreover, if B_{l_1} and B_{l_2} are not in the same component in G_B, then l_1 and l_2 can never be in the same set in U because both start in distinct sets (line 1 of Activate), and it is not possible for a set containing l_1 to be merged with a set containing l_2.

It takes $O(|L|n)$ time to build H (linear in the size of the heap). Each deleteMin() operation takes $O(\log(|L|n))$ (logarithmic in the size of the heap), which is equivalent to $O(\log(n))$ be-

cause $|L| \leq n$. If U is implemented by a union-find algorithm Update-Components takes amortized time of $O(\alpha(|L|)$, where α denotes the inverse Ackermann function. Moreover, Update-Components may only be called once for each iteration of the while loop in Expand-Landmarks (it is either called immediately on l^* and s^* if B_{l^*} is large enough, or it is called when the ball grows large enough in Activate). All other operations also take time proportional to the number of landmark-point pairs. So the runtime of this algorithm is $O(|L|n) + \text{iter} \cdot O(\log n + \alpha(|L|))$, where iter is the number of iterations of the while loop. As the number of iterations is bounded by $|L|n$, and $\alpha(|L|)$ is effectively constant, this gives a worst-case running time of $O(|L|n \log n)$.

2.5 Empirical Study

We use our *Landmark Clustering* algorithm to cluster proteins using sequence similarity. One versus all distance queries are particularly relevant in this setting because of sequence database search programs such as BLAST (Basic Local Alignment Search Tool) [AGM+90]. For each data set we first build a BLAST database containing all the sequences, and then compare only some of the sequences to the entire database. BLAST aligns the queried sequence to sequences in the database, and produces a "bit score" for each alignment, which is a measure of its quality (we invert the bit score to make it a distance). However, BLAST does not consider alignments with some of the sequences in the database, in which case we assign distances of infinity to the corresponding sequences. We observe that if we define distances in this manner they almost form a metric in practice: when we draw triplets of sequences at random and check the distances between them the triangle inequality is almost always satisfied. Moreover, BLAST is very successful at detecting sequence homology in large sequence databases, therefore it is plausible that clustering using these distances satisfies the (c, ϵ)-property for some relevant clustering C_T.

We perform experiments on data sets obtained from two classification databases: Pfam [FMT+10], version 24.0, October 2009; and SCOP [MBHC95], version 1.75, June 2009. Both of these sources classify proteins by their evolutionary relatedness, therefore we can use their classifications as a ground truth to evaluate the clusterings produced by our algorithm and other methods.

Pfam classifies proteins using hidden Markov models (HMMs) that represent multiple sequence alignments. There are two levels in the Pfam classification hierarchy: family and clan. In our clustering experiments we compare with a classification at the family level because the relationships at the clan level are less likely to be discerned with sequence alignment. In each experiment we randomly select several large families (of size between 1000 and 10000) from Pfam-A (the manually curated part of the classification), retrieve the sequences of the proteins in these families, and use our *Landmark-Clustering* algorithm to cluster the data set.

SCOP groups proteins on the basis of their 3D structures, so it only classifies proteins whose structure is known. Thus the data sets from SCOP are much smaller in size. The SCOP classification is also hierarchical: proteins are grouped by class, fold, superfamily, and family. We consider the classification at the superfamily level because this seems most appropriate given that we are only using sequence information. As with the Pfam data, in each experiment we create a data set by randomly choosing several superfamilies (of size between 20 and 200), retrieve the sequences of the corresponding proteins, and use our *Landmark-Clustering* algorithm to cluster the data set.

Once we cluster a particular data set, we compare the clustering to the manual classification using the distance measure from the theoretical part of our work. To find the fraction of misclassified points under the optimal matching of clusters in C to clusters in C' we solve a minimum weight bipartite matching problem where the cost of matching C_i to $C'_{\sigma(i)}$ is $|C_i - C'_{\sigma(i)}|/n$.

2.5.1 Choice of Parameters

To run *Landmark-Clustering*, we set k using the number of clusters in the ground truth clustering. For each Pfam data set we use $40k$ landmarks/queries, and for each SCOP data set we use $30k$ landmarks/queries. In addition, our algorithm uses three parameters $(q, s_{\min}, n')$ whose value is set in the proof based on α and ϵ, assuming that the clustering instance satisfies the $(1+\alpha, \epsilon)$-property. In practice we must choose some value for each parameter. In our experiments we set them as a function of the number of points in the data set, and the number of clusters. We set $q = 2n/k$, $s_{\min} = 0.05n/k$ for Pfam data sets, and $s_{min} = 0.1n/k$ for SCOP

data sets, and $n' = 0.5n$. Since the selection of landmarks is randomized, for each data set we perform several clusterings, compare each to the ground truth, and report the median quality.

Landmark-Clustering is most sensitive to the $s_{\min}$ parameter, and will not report a clustering if $s_{\min}$ is too small or too large. We recommend trying several reasonable values of $s_{\min}$, in increasing or decreasing order, until you get a clustering and none of the clusters are too large. If you get a clustering where one of the clusters is very large, this likely means that several ground truth clusters have been merged. This may happen because $s_{\min}$ is too small causing balls of outliers to connect different cluster cores, or $s_{\min}$ is too large causing balls in different cluster cores to overlap.

The algorithm is less sensitive to the n' parameter. However, if you set n' too large some ground truth clusters may be merged, so we recommend using a smaller value ($0.5n \leq n' \leq 0.7n$) because all of the points are still clustered during the last step. Again, for some values of n' the algorithm may not output a clustering, or output a clustering where some of the clusters are too large. Our algorithm is least sensitive to the q parameter. Using more landmarks (if you can afford it) can make up for a poor choice of q.

2.5.2 Results

Figure 2.3 shows the results of our experiments on the Pfam data sets. One can see that for most of the data sets (other than data sets 7 and 9) we find a clustering that is almost identical to the ground truth. These data sets are very large, so as a benchmark for comparison we can only consider algorithms that use a comparable amount of distance information (since we do not have the full distance matrix). A natural choice is the following algorithm: randomly choose a set of landmarks L, $|L| = d$; embed each point in a d-dimensional space using distances to L; use k-means clustering in this space (with distances given by the Euclidian norm). Our embedding scheme is a Lipschitz embedding with singleton subsets (see [TC03]), which gives distances with low distortion for points near each other in a metric space.

Notice that this procedure uses exactly d one versus all distance queries, so we can set d equal to the number of queries used by our algorithm. We expect this algorithm to work well, and if you look at Figure 2.3 you can see that it finds reasonable clusterings. Still, the

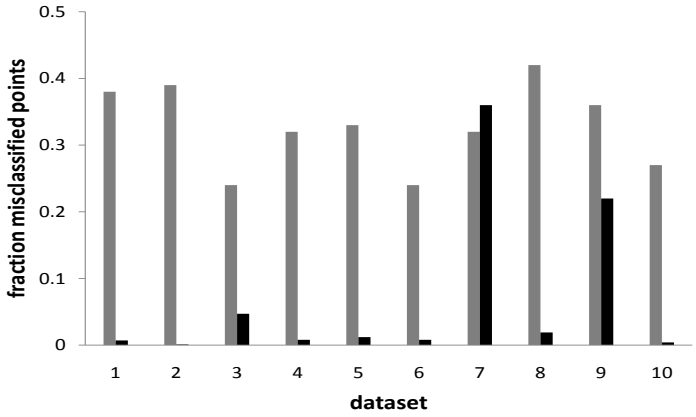


Figure 2.3: Comparing the performance of k-means in the embedded space (gray) and *Landmark-Clustering* (black) on 10 data sets from Pfam. Data sets **1-10** are created by randomly choosing 8 families from Pfam of size s, $1000 \leq s \leq 10000$.

clusterings reported by this algorithm do not closely match the Pfam classification, showing that our results are indeed significant.

Figure 2.4 shows the results of our experiments on the SCOP data sets. These results are not as good, which is likely because the SCOP classification at the superfamily level is based on biochemical and structural evidence in addition to sequence evidence. By contrast, the Pfam classification is based entirely on sequence information. Still, because the SCOP data sets are much smaller, we can compare our algorithm to methods that require distances between all the points. In particular, Paccanaro, Casbon, and Saqi showed that spectral clustering using sequence data works well when applied to the proteins in SCOP [PCS06]. Thus we use the exact method described by Paccanaro et al. as a benchmark for comparison on the SCOP data sets. Moreover, other than clustering randomly generated data sets from SCOP, we also consider the two main examples from Paccanaro et al., which are labeled **A** and **B** in the figure. From Figure 2.4 we can see that the performance of *Landmark-Clustering* is comparable to that of the spectral method, which is very good considering that the algorithm used by Paccanaro et al. significantly outperforms other clustering algorithms on this data [PCS06]. Moreover, the

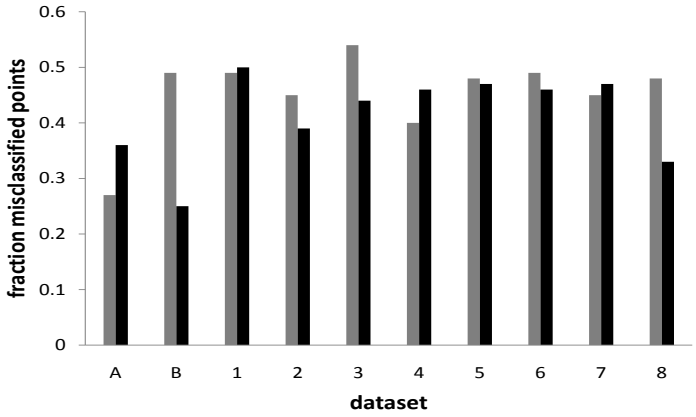


Figure 2.4: Comparing the performance of spectral clustering (gray) and *Landmark-Clustering* (black) on 10 data sets from SCOP. Data sets **A** and **B** are the two main examples from Paccanaro et al. [PCS06], the other data sets (**1-8**) are created by randomly choosing 8 superfamilies from SCOP of size s, $20 \leq s \leq 200$.

spectral clustering algorithm requires the full distance matrix as input, and takes much longer to run.

2.5.3 Testing the approximation stability assumption

To see whether the (c, ϵ) property is a reasonable assumption for our data, we look at whether our data sets have the structure implied by our assumption. We do this by measuring the separation of the ground truth clusters in our data sets. For each data set in our study, we sample some points from each ground truth cluster. We consider whether the sampled points are more similar to points in the same cluster than to points in other clusters. More specifically, for each point we record the median within-cluster similarity, and the maximum between-cluster similarity. If our data sets indeed have well-separated cluster cores, as implied by our assumption, then for a lot of the points the median within-cluster similarity should be significantly larger than the maximum between-cluster similarity. We can see that this is indeed the case for the Pfam data sets. However, this is not typically the case for the SCOP data sets, where most points have little similarity to the majority of the points in their

ground truth cluster. These observations explain our results on the two sets of data: we are able to accurately cluster the Pfam data sets, and our algorithm is much less accurate on the SCOP data sets. The complete results of these experiments can be found at http://cs-people.bu.edu/kvodski/clusteringProperties/description.html.

Chapter 3

Clustering with Limited Information Given a Different Structure

In this chapter we present a limited information algorithm that clusters accurately given that the approximation stability property is satisfied for the *min-sum* objective. If the (c, ϵ)-property holds for the *min-sum* objective, the structure of the clustering instance is quite different, and the algorithm presented in the previous chapter fails to find an accurate clustering. The min-sum objective is also considerably harder to approximate. For the k-median objective the best approximation guarantee is $(3 + \epsilon)$ given by Arya et al. [AGK+04]. For the min-sum objective when the number of clusters is arbitrary there is an $O(\delta^{-1} \log^{1+\delta} n)$-approximation algorithm with running time $n^{O(1/\delta)}$ due to Bartal, Charikar, and Raz [BCR01].

In this chapter we describe a different limited information algorithm that requires $O(k \log k)$ one versus all queries, where k is the number of clusters. Section 3.1 defines the *min-sum* objective function and the closely related *balanced k-median* objective. Our *Landmark-Clustering-Min-Sum* algorithm is presented in Section 3.2, with some details omitted. This section also gives our theoretical performance guarantee, and some high-level intuition for our theoretical arguments. Section 3.3 formally describes the structure of the clustering instance that follows from the (c, ϵ)-property for the min-sum objective function, and gives a full description of the algorithm and its proof of correctness.

Our theoretic arguments require that we know the optimum objective value OPT. This is usually not true in practice, and Section 3.3 contains additional analysis concerning what happens when we do not know OPT, and must estimate one of the parameters of the algorithm. This discussion also gives intution about how to choose this parameter in practice. We conclude with the results of our experimental study, which are presented in Section 3.4. We show that our algorithm can accurately cluster protein sequences if we use BLAST as the one versus all query. We also describe cases where *Landmark-Clustering-Min-Sum* is likely to produce a more meaningful clustering than the algorithm presented in the previous chapter.

3.1 Approximation Stability of the Min-Sum Objective Function

The *min-sum* objective function for clustering is to minimize $\Phi(C) = \sum_{i=1}^{k} \sum_{x,y \in C_i} d(x,y)$. We reduce the min-sum clustering problem to the related *balanced k-median* problem. The balanced k-median objective function seeks to minimize $\Psi(C) = \sum_{i=1}^{k} |C_i| \sum_{x \in C_i} d(x, c_i)$, where c_i is the median of cluster C_i, which is the point $y \in C_i$ that minimizes $\sum_{x \in C_i} d(x,y)$. As pointed out in [BCR01], in metric spaces the two objective functions are related to within a factor of 2: $\Psi(C)/2 \leq \Phi(C) \leq \Psi(C)$.

In our analysis we assume that S satisfies the (c, ϵ) approximation stability property of Balcan, Blum, and Gupta [BBG09] for the min-sum and balanced k-median objective functions. The (c, ϵ)-property is formally defined in the previous chapter.

We note that because any $(1+\alpha)$-approximation of the balanced k-median objective is a $2(1+\alpha)$-approximation of the min-sum objective, it follows that if the clustering instance satisfies the $(2(1+\alpha), \epsilon)$-property for the min-sum objective, then it satisfies the $(1+\alpha, \epsilon)$-property for balanced k-median.

3.2 Landmark-Clustering-Min-Sum

In this section we present a clustering algorithm that given the $(1+\alpha, \epsilon)$-property for the balanced k-median objective finds an accurate clustering in a limited distance information setting. Our algorithm is outlined in Algorithm 3.1, with some details omitted. We start by uniformly at random choosing n' points that we call *landmarks*, where n' is an appropriate number. For each landmark that we choose we use a *one versus all* query to get the distances between this landmark and all other points. These are the only distances used by our procedure.

Our algorithm then expands a ball B_l around each landmark $l \in L$ one point at a time. We first sort all landmark-point pairs (l, s) by $d(l, s)$. We then consider these pairs in order of increasing distance, skipping pairs where l or s have already been clustered; the clustered points are maintained in the set $\bar{S}$. In each iteration we check whether some ball B_{l^*} passes the test in line 13. Our test considers the size of the ball and the next largest landmark-point distance, and checks whether their product is greater than the threshold T. If this is the case, we consider all balls that overlap B_{l^*} on any points, and compute a cluster that contains all

the points in these balls. Points and landmarks in the cluster are then removed from further consideration by adding the clustered points to $\bar{S}$, and removing the clustered points from any ball.

Our procedure terminates once we find k clusters. If we reach the final landmark-point pair, we stop and report the remaining unclustered points as part of the same cluster. If the algorithm terminates without partitioning all the points, we assign each remaining point to the cluster containing the closest clustered landmark. In our analysis we show that if the clustering instance satisfies the $(1+\alpha, \epsilon)$-property for the balanced k-median objective function, our procedure will output exactly k clusters.

The most time-consuming part of our algorithm is sorting all landmark-points pairs, which takes $O(|L|n \log n)$, where n is the size of the data set and L is the set of landmarks. With a simple implementation that uses a hashed set to store the points in each ball, the total cost of computing the clusters and removing clustered points from active balls is at most $O(|L|n)$ each. All other operations take asymptotically less time, so the overall runtime of our procedure is $O(|L|n \log n)$.

Algorithm 3.1 Landmark-Clustering-Min-Sum(k, n', T)

1: L = Landmark-Selection(n');
2: **for** each $l \in L$ **do**
3: $B_l = \emptyset$;
4: **end for**
5: $i = 1$, $\bar{S} = \emptyset$;
6: **while** $i \leq k$ **do**
7: (l, s) = GetNextActivePair();
8: $B_l = B_l + \{s\}$;
9: (l', s') = PeekNextActivePair();
10: **if** $d(l, s) == d(l', s')$ **then**
11: continue;
12: **end if**
13: **while** $\exists l^* \in L - \bar{S} : |B_{l^*}| \cdot d(l', s') > T$ **do**
14: $L' = \{l \in L - \bar{S} : B_l \cap B_{l^*} \neq \emptyset\}$;
15: $C_i = \{s \in S : s \in B_l \text{ and } l \in L'\}$;
16: **for** each $s \in C_i$ **do**
17: $\bar{S} = \bar{S} + \{s\}$;
18: **for** each $l \in L$ **do**
19: $B_l = B_l - \{s\}$;
20: **end for**
21: **end for**
22: $i = i + 1$;
23: **end while**
24: **end while**
25: **return** $C = \{C_1, \ldots C_k\}$;

We now present our theoretical guarantee for Algorithm 3.1.

Theorem 3.1. *Given a metric space $M = (X, d)$, where d is unknown, and a set of points S, if the instance (S, d) satisfies the $(1 + \alpha, \epsilon)$-property for the balanced-k-median objective function, we are given the optimum objective value* OPT*, and each cluster in the target clustering C_T has size at least $(6 + 240/\alpha)\epsilon n$, then* Landmark-Clustering-Min-Sum$(k, n', \frac{\alpha \text{OPT}}{40\epsilon n})$ *outputs a clustering that is $O(\epsilon/\alpha)$-close to C_T with probability at least $1 - \delta$. The algorithm uses $n' = \frac{1}{(3+120/\alpha)\epsilon} \ln \frac{k}{\delta}$* one versus all distance queries*, and has a runtime of $O(n'n \log n)$.*

We note that $n' = O(k \ln \frac{k}{\delta})$ if the sizes of the target clusters are balanced. In addition, if we do not know the value of OPT, we can still find an accurate clustering by running Algorithm 3.1 from line 2 at most $n'n^2$ times with increasing estimates of T until enough points are clustered.

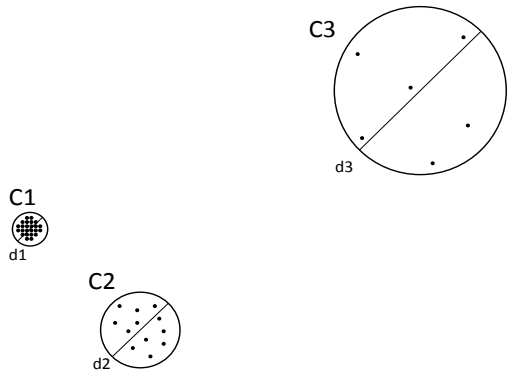


Figure 3.1: Cluster cores C_1, C_2 and C_3 are shown with diameters d_1, d_2 and d_3, respectively. The diameters of the cluster cores are inversely proportional to their sizes.

It is not necessary to recompute the landmarks, so the number of distance queries that are required remains the same. We next give some high-level intuition for how our procedures work.

Given our approximation stability assumption, the target clustering must have the structure shown in Figure 3.1. Each target cluster C_i has a "core" of well-separated points, where any two points in the cluster core are closer than a certain distance d_i to each other, and any point in a different core is farther than cd_i, for some constant c. Moreover, the diameters of the cluster cores are inversely proportional to the cluster sizes: there is some constant θ such that $|C_i| \cdot d_i = \theta$ for each cluster C_i. Given this structure, it is possible to classify the points in the cluster cores correctly if we extract the smaller diameter clusters first. For example, we can extract C_1, followed by C_2 and C_3 if we choose the threshold T correctly and we have selected a landmark from each cluster core. However, if we wait until some ball contains all of C_3, C_1 and C_2 may be merged.

3.3 Algorithm Analysis

In this section we present a formal analysis of our algorithm, and give the proof of Theorem 3.1. We first describe the structure of the clustering instance that is implied by the $(1+\alpha, \epsilon)$-property for the balanced k-median objective function. We then present a more complete description of the algorithm that we refer to in our proof. We then give a general overview of our argument,

which is followed by the complete proof.

3.3.1 Structure of the Clustering Instance

We denote by $C^* = \{C_1^*, \ldots, C_k^*\}$ the optimal balanced-k-median clustering with objective value OPT=$\Psi(C^*)$. For each cluster C_i^*, let c_i^* be the median point in the cluster. For $x \in C_i^*$, define $w(x) = |C_i^*|d(x, c_i^*)$ and let $w = \text{avg}_x w(x) = \frac{\text{OPT}}{n}$. Define $w_2(x) = \min_{j \neq i} |C_j^*| d(x, c_j^*)$.

It is proved in [BBG09] that if the instance satisfies the $(1+\alpha, \epsilon)$-property and each cluster in C^* has size at least $\max(6, 6/\alpha) \cdot \epsilon n$, then at most 2ϵ-fraction of points $x \in S$ have $w_2(x) < \frac{\alpha w}{4\epsilon}$. In addition, by definition of the average weight w at most $120\epsilon/\alpha$-fraction of points $x \in S$ have $w(x) > \frac{\alpha w}{120\epsilon}$.

We call point x *good* if both $w(x) \leq \frac{\alpha w}{120\epsilon}$ and $w_2(x) \geq \frac{\alpha w}{4\epsilon}$, else x is called bad. Let X_i be the *good* points in the optimal cluster C_i^*, and let $B = S \setminus \cup X_i$ be the bad points.

Lemma 3.3, which is very similar to Lemma 14 of [BBG09], proves that the optimum balanced k-median clustering must have the following structure:

1. For all x, y in the same X_i, we have $d(x, y) \leq \frac{\alpha w}{60\epsilon |C_i^*|}$.
2. For $x \in X_i$ and $y \in X_{j \neq i}$, $d(x, y) > \frac{\alpha w}{5\epsilon} / \min(|C_i^*|, |C_j^*|)$.
3. The number of bad points is at most $b = (2 + 120/\alpha)\epsilon n$.

3.3.2 Full Algorithm Description

We next give a more detailed description of our algorithm.

Algorithm 3.2 Landmark-Clustering-Min-Sum(k, n', T)

L = Landmark-Selection(n');
for each $l \in L$ **do**
  $B_l = \emptyset$;
end for
$i = 1$, $\bar{S} = \emptyset$;
while $i \leq k$ **do**
  (l, s) = GetNextActivePair();
  $r_1 = d(l, s)$;
  if $((l', s')$ = PeekNextActivePair$())$! = null **then**
    $r_2 = d(l', s')$;
  else
    $C_i = S - \bar{S}$;
    break;
  end if
  $B_l = B_l + \{s\}$;
  if $r_1 == r_2$ **then**
    continue;
  end if
  while $\exists l \in L - \bar{S} : |B_l| > T/r_2$ and $i \leq k$ **do**
    $l^* = \text{argmax}_{l \in L - \bar{S}} |B_l|$;
    $L' = \{l \in L - \bar{S} : B_l \cap B_{l^*} \neq \emptyset\}$;
    $C_i = \{s \in S : s \in B_l \text{ and } l \in L'\}$;
    for each $s \in C_i$ **do**
      $\bar{S} = \bar{S} + \{s\}$;
      for each $l \in L$ **do**
        $B_l = B_l - \{s\}$;
      end for
    end for
    $i = i + 1$;
  end while
end while
return $C = \{C_1, \ldots C_k\}$;

3.3.3 Proof of Theorem 3.1 and Additional Analysis

Our algorithm expands a ball around each landmark, one point at a time, until some ball is large enough. We use r_1 to refer to the current radius of the balls, and r_2 to refer to the next

relevant radius (next largest landmark-point distance). To pass the test in line 19, a ball must satisfy $|B_l| > T/r_2$. We choose T such that by the time a ball satisfies the conditional, it must overlap some good set X_i. Moreover, at this time the radius must be large enough for X_i to be entirely contained in some ball; X_i will therefore be part of the cluster computed in line 21. However, the radius is too small for a single ball to overlap different good sets and for two balls overlapping different good sets to share any points. Therefore the computed cluster cannot contain points from any other good set. Points and landmarks in the cluster are then removed from further consideration. The same argument can then be applied again to show that each cluster output by the algorithm entirely contains a single good set. Thus the clustering output by the algorithm agrees with C^* on all the good points, so it must be closer than $b+\epsilon = O(\epsilon/\alpha)$ to C_T. A more detailed argument is given below.

Proof. Since each cluster in the target clustering has more than $(6 + 240/\alpha)\epsilon n$ points, and the optimal balanced-k-median clustering C^* can differ from the target clustering by fewer than ϵn points, each cluster in C^* must have more than $(5+240/\alpha)\epsilon n$ points. Moreover, by Lemma 3.3 we may have at most $(2+120/\alpha)\epsilon n$ bad points, and hence each $|X_i| = |C_i^* \setminus B| > (3+120/\alpha)\epsilon n$. We will use s to refer to the $(3 + 120/\alpha)\epsilon n$ quantity.

Our argument assumes that we have chosen at least one landmark from each good set X_i. Lemma 3.4 argues that after selecting $n' = \frac{n}{s}\ln\frac{k}{\delta} = \frac{1}{(3+120/\alpha)\epsilon}\ln\frac{k}{\delta}$ landmarks the probability of this happening is at least $1 - \delta$. Moreover, if the target clusters are balanced in size: $\max_{C\in C_T} |C|/\min_{C\in C_T} |C| \leq c$ for some constant c, because the size of each good set is at least half the size of the corresponding target cluster, it must be the case that $2sc \cdot k \geq n$, so $n/s = O(k)$.

Suppose that we order the clusters of C^* such that $|C_1^*| \geq |C_2^*| \geq \ldots |C_k^*|$, and let $n_i = |C_i^*|$. Define $d_i = \frac{\alpha w}{60\epsilon|C_i^*|}$ and recall that $\max_{x,y\in X_i} d(x, y) \leq d_i$. Note that because there is a landmark in each good set X_i, for radius $r \geq d_i$ there exists some ball containing all of X_i. We use $B_l(r)$ to denote a ball of radius r around landmark l: $B_l(r) : \{s \in S \mid d(s, l) \leq r\}$.

If we apply Lemma 3.5 with all the clusters in C^*, we can see that as long as $r \leq 3d_1$, a ball cannot contain points from more than one good set and balls overlapping different good sets cannot share any points. We also observe that when both $r \leq 3d_1$ and $r < d_i$ are true, a ball $B_l(r)$ containing points from X_i does not satisfy $|B_l(r)| \geq T/r$. For $r \leq 3d_1$ a ball cannot contain points from different good sets; therefore any ball containing points from X_i has size at most $|C_i^*| + b < \frac{3n_i}{2}$. In addition, for $r < d_i$ the size bound $T/r > T/d_i = \frac{\alpha w}{40\epsilon}/\frac{\alpha w}{60\epsilon|C_i^*|} = \frac{3n_i}{2}$. Therefore for these values of r any ball containing points from X_i is too small to satisfy the conditional.

Finally, we observe that for $r = 3d_1$ some ball $B_l(r)$ containing all of X_1 does satisfy

$|B_l(r)| \geq T/r$. Clearly, for $r = 3d_1$ there is some ball containing all of X_1, which must have size at least $|C_1^*| - b \geq n_1/2$. For $r = 3d_1$ the size bound $T/r = n_1/2$, so this ball is large enough to satisfy this conditional. Moreover, for $r \leq 3d_1$ the size bound T/r is at least $n_1/2$. Therefore a ball containing only bad points cannot pass our test for $r \leq 3d_1$ because the number of bad points is at most $b < n_1/2$.

Consider the smallest radius r^* for which some ball $B_{l^*}(r^*)$ satisfies $|B_{l^*}(r^*)| \geq T/r^*$. It must be the case that $r^* \leq 3d_1$, and B_{l^*} overlaps with some good set X_i because we cannot have a ball containing only bad points for $r^* \leq 3d_1$. Moreover, by our previous argument because B_{l^*} contains points from X_i, it must be the case that $r^* \geq d_i$, and therefore some ball contains all the points in X_i. Consider a cluster $\hat{C}$ of all the points in balls that overlap B_{l^*}: $\hat{C} = \{s \in S \mid s \in B_l \text{ and } B_l \cap B_{l^*} \neq \emptyset\}$, which must include all the points in X_i. In addition, B_{l^*} cannot share any points with balls that overlap other good sets because $r^* \leq 3d_1$, therefore $\hat{C}$ does not contain points from any other good set. Therefore the cluster $\hat{C}$ entirely contains some good set and no points from any other good set.

These facts suggest the following algorithm for finding a clustering that classifies all the good points correctly: increment r until some ball satisfies $|B_l(r)| \geq T/r$, compute the cluster containing all points in balls that overlap $B_l(r)$, remove these points, and repeat until we find k clusters. We can argue that each cluster output by the algorithm entirely contains some good set and no points from any other good set. Each time we can consider the clusters $C \subseteq C^*$ *whose good sets have not yet been output*, order them by size, and apply Lemma 3.5 with C to argue that while $r \leq 3d_1$ the radius is too small for the computed cluster to overlap any of the *remaining* good sets. As before, we can argue that by the time we reach $3d_1$ we must output some cluster. In addition, when $r \leq 3d_1$ we cannot output a cluster containing only bad points and whenever we output a cluster overlapping some good set X_i, it must be the case that $r \geq d_i$. Therefore the computed cluster must contain all of X_i and no points from any other good set.

If there are any unclustered points upon the completion of the algorithm, we can assign the remaining points to any cluster. Still, we are able to classify all the good points correctly, so the reported clustering must be closer than $b + \text{dist}(C^*, C_T) < b + \epsilon = O(\epsilon/\alpha)$ to C_T.

It suffices to show that even though our algorithm only considers discrete values of r corresponding to landmark-point distances, the output of our procedure exactly matches the output of the conceptual algorithm described above. Consider the smallest (continuous) radius r^* for which some ball $B_{l_1}(r^*)$ satisfies $|B_{l_1}(r^*)| \geq T/r^*$. We use d_{real} to refer to the largest landmark-point distance such that $d_{real} \leq r^*$. Clearly, by the time our algorithm reaches $r_1 = d_{real}$ it must be the case that B_{l_1} passes the test on line 19: $|B_{l_1}| > T/r_2$, and this test is not passed by any ball at any prior time. Moreover, B_{l_1} must be the largest ball passing our test at this point because if there is another ball B_{l_2} that also satisfies our test when $r_1 = d_{real}$ it must be the case that $|B_{l_1}| > |B_{l_2}|$ because B_{l_1} satisfies $|B_{l_1}(r)| \geq T/r$ for a smaller r. Finally because there are no landmark-point pairs (l, s) with $r_1 < d(l, s) < r_2$, $B_l(r_1) = B_l(r^*)$ for each

landmark $l \in L$. Therefore the cluster that we compute on line 22 for $B_{l_1}(r_1)$ is equivalent to the cluster the conceptual algorithm computes for $B_{l_1}(r^*)$. We can repeat this argument for each cluster output by the conceptual algorithm, showing that Algorithm 3.2 finds exactly the same clustering.

We note that when there is only one good set left the test in line 19 may not be satisfied anymore if $3d_1 \geq \max_{x,y \in S} d(x,y)$, where d_1 is the diameter of the remaining good set. However, in this case if we exhaust all landmark-points pairs we report the remaining points as part of a single cluster (line 12), which must contain the remaining good set.

With a simple implementation that uses a hashed set to keep track of the points in each ball, the runtime of our procedure is $O(|L|n \log n)$, which is given by the time necessary to sort all landmark-point pairs by distance. All other operations take asymptotically less time. In particular, over the entire run of the algorithm, the cost of computing the clusters in lines 21-22 is at most $O(n|L|)$, and the cost of removing clustered points from active balls in lines 23-28 is also at most $O(n|L|)$.

□

Theorem 3.2. *If we are not given the optimum objective value* OPT*, then we can still find a clustering that is $O(\epsilon/\alpha)$-close to C_T with probability at least $1-\delta$ by running Landmark-Clustering-Min-Sum at most $n'n^2$ times with the same set of landmarks, where the number of landmarks $n' = \frac{1}{(3+120/\alpha)\epsilon} \ln \frac{k}{\delta}$ as before.*

Proof. If we are not given the value of OPT and therefore do not know the value of $w = \frac{\text{OPT}}{n}$, then we have to estimate the threshold parameter T for deciding when a cluster develops. Let us use T^* to refer to its correct value ($T^* = \frac{\alpha w}{40\epsilon}$). We first note that there are at most $n \cdot n|L|$ relevant values of T to try, where L is the set of landmarks. Our test in line 19 checks whether the product of a ball size and a ball radius is larger than T, and there are only n possible ball sizes and $|L|n$ possible values of a ball radius.

Suppose that we choose a set of landmarks L, $|L| = n'$, as before. We then compute all $n'n^2$ relevant values of T and order them in ascending order: $T_i \leq T_{i+1}$ for $1 \leq i < n'n^2$. Then we repeatedly execute Algorithm 3.2 starting on line 2 with increasing estimates of T. Note that this is equivalent to trying all continuous values of T in ascending order because the execution of the algorithm does not change for any T' such that $T_i \leq T' < T_{i+1}$. In other words, when $T_i \leq T' < T_{i+1}$, the algorithm will give the same exact answer for T_i as it would for T'.

Our procedure stops the first time we cluster at least $n-b$ points, where b is the maximum number of bad points. We give an argument that this gives an accurate clustering with an additional error of b.

As before, we assume that we have selected at least one landmark from each good set, which happens with probability at least $1-\delta$. Clearly, if we choose the right threshold T^* the algorithm must cluster at least $n-b$ points because the clustering will contain all the good points. Therefore the first time the algorithm clusters at least $n-b$ points for some estimated

threshold T, it must be the case that $T \leq T^*$. Lemma 3.6 argues that if $T \leq T^*$ and the number of clustered points is at least $n - b$, then the reported partition must be a k-clustering that contains a distinct good set in each cluster. This clustering may exclude up to b points, all of which may be good points. Still, if we arbitrarily assign the remaining points we will get a clustering that is closer than $2b + \epsilon = O(\epsilon/\alpha)$ to C_T.

□

Lemma 3.3. *If the balanced k-median instance satisfies the $(1+\alpha, \epsilon)$-property and each cluster in C^* has size at least $\max(6, 6/\alpha) \cdot \epsilon n$ we have:*

1. *For all x, y in the same X_i, we have $d(x, y) \leq \frac{\alpha w}{60\epsilon |C_i^*|}$.*
2. *For $x \in X_i$ and $y \in X_{j \neq i}$, $d(x, y) > \frac{\alpha w}{5\epsilon} / \min(|C_i^*|, |C_j^*|)$.*
3. *The number of bad points is at most $b = (2 + 120/\alpha)\epsilon n$.*

Proof. For part 1, since $x, y \in X_i \subseteq C_i^*$ are both good, they are at distance of at most $\frac{\alpha w}{120\epsilon |C_i^*|}$ to c_i^*, and hence at distance of at most $\frac{\alpha w}{60\epsilon |C_i^*|}$ to each other.

For part 2 assume without loss of generality that $|C_i^*| \geq |C_j^*|$. Both $x \in C_i^*$ and $y \in C_j^*$ are good; it follows that $d(y, c_j^*) \leq \frac{\alpha w}{120\epsilon |C_j^*|}$, and $d(x, c_j^*) > \frac{\alpha w}{4\epsilon |C_j^*|}$ because $|C_j^*| d(x, c_j^*) \geq w_2(x) > \frac{\alpha w}{4\epsilon}$. By the triangle inequality it follows that

$$d(x, y) \geq d(x, c_j^*) - d(y, c_j^*) \geq \frac{\alpha w}{\epsilon |C_j^*|} (\frac{1}{4} - \frac{1}{120}) > \frac{\alpha w}{5\epsilon} / \min(|C_i^*|, |C_j^*|),$$

where we use that $|C_j^*| = \min(|C_i^*|, |C_j^*|)$.

Part 3 follows from the maximum number of points that may not satisfy each of the properties of the good points and the union bound. □

Lemma 3.4. *After selecting $\frac{n}{s} \ln \frac{k}{\delta}$ points uniformly at random, where s is the size of the smallest good set, the probability that we did not choose a point from every good set is smaller than $1 - \delta$.*

Proof. We denote by s_i the cardinality of X_i. Observe that the probability of not selecting a point from some good set X_i after $\frac{nc}{s}$ samples is $(1 - \frac{s_i}{n})^{\frac{nc}{s}} \leq (1 - \frac{s_i}{n})^{\frac{nc}{s_i}} \leq (e^{-\frac{s_i}{n}})^{\frac{nc}{s_i}} = e^{-c}$. By the union bound the probability of not selecting a point from every good set after $\frac{nc}{s}$ samples is at most ke^{-c}, which is equal to δ for $c = \ln \frac{k}{\delta}$. □

Lemma 3.5. *Given a subset of clusters $C \subseteq C^*$, and the set of the corresponding good sets X, let $s_{max} = \max_{C_i \in C} |C_i|$ be the size of the largest cluster in C, and $d_{min} = \frac{\alpha w}{60\epsilon s_{max}}$. Then for $r \leq 3d_{min}$, a ball cannot overlap a good set $X_i \in X$ and any other good set, and a ball containing points from a good set $X_i \in X$ cannot share any points with a ball containing points from any other good set.*

Proof. By part 2 of Lemma 3.3, for $x \in X_i$ and $y \in X_{j \neq i}$ we have

$$d(x,y) > \frac{\alpha w}{5\epsilon} / \min(|C_i^*|, |C_j^*|).$$

It follows that for $x \in X_i \in X$ and $y \in X_{j \neq i}$ we must have $d(x,y) > \frac{\alpha w}{5\epsilon} / \min(|C_i^*|, |C_j^*|) \geq \frac{\alpha w}{5\epsilon} / |C_i^*| > \frac{\alpha w}{5\epsilon} / s_{max} = 12 d_{min}$, where we use the fact that $|C_i| \leq s_{max}$. So a point in a good set in X and a point in any other good set must be farther than $12 d_{min}$.

To prove the first part, consider a ball B_l of radius $r \leq 3 d_{min}$ around landmark l. In other words, $B_l = \{s \in S \mid d(s,l) \leq r\}$. If B_l overlaps a good set in X and any other good set, then it must contain a point $x \in X_i \in X$ and a point $y \in X_{j \neq i}$. It follows that $d(x,y) \leq d(x,l) + d(l,y) \leq 2r \leq 6 d_{min}$, giving a contradiction.

To prove the second part, consider two balls B_{l_1} and B_{l_2} of radius $r \leq 3 d_{min}$ around landmarks l_1 and l_2. Suppose B_{l_1} and B_{l_2} share at least one point: $B_{l_1} \cap B_{l_2} \neq \emptyset$, and use s^* to refer to this point. It follows that the distance between any point $x \in B_{l_1}$ and $y \in B_{l_2}$ satisfies $d(x,y) \leq d(x,s^*) + d(s^*,y) \leq [d(x,l_1) + d(l_1,s^*)] + [d(s^*,l_2) + d(l_2,y)] \leq 4r \leq 12 d_{min}$.

If B_{l_1} overlaps with $X_i \in X$ and B_{l_2} overlaps with $X_{j \neq i}$, and the two balls share at least one point, there must be a pair of points $x \in X_i$ and $y \in X_{j \neq i}$ such that $d(x,y) \leq 12 d_{min}$, giving a contradiction. Therefore if B_{l_1} overlaps with some good set $X_i \in X$ and B_{l_2} overlaps with any other good set, $B_{l_1} \cap B_{l_2} = \emptyset$.

□

Lemma 3.6. *If $T \leq T^* = \frac{\alpha w}{40\epsilon}$ and the number of clustered points is at least $n - b$, then the clustering output by Landmark-Clustering-Min-Sum using the threshold T must be a k-clustering that contains a distinct good set in each cluster.*

Proof. Our argument considers the points that are in each cluster that is output by the algorithm. Let us call a good set *covered* if any of the clusters $C_1, \ldots, C_{i-1}$ found so far contain points from it. We will use $\bar{C}^*$ to refer to the clusters in C^* whose good sets are not *covered*. It is critical to observe that if $T \leq T^*$ then if C_i contains points from an *uncovered* good set, C_i cannot overlap with any other good set.

To see this, let us order the clusters in $\bar{C}^*$ by decreasing size: $|C_1^*| \geq |C_2^*| \geq \ldots |C_j^*|$, and let $n_i = |C_i^*|$. As before, define $d_i = \frac{\alpha w}{60\epsilon |C_i^*|}$. Applying Lemma 3.5 with $\bar{C}^*$ we can see that for $r \leq 3 d_1$, a ball of radius r cannot overlap a good set in $\bar{C}^*$ and any other good set, and a ball containing points from a good set in $\bar{C}^*$ cannot share any points with a ball containing points from any other good set. Because $T \leq T^*$ we can also argue that by the time we reach $r = 3 d_1$ we must output some cluster.

Given this observation, it is clear that the algorithm can cover at most one new good set in each cluster that it outputs. In addition, if a new good set is covered this cluster may not contain points from a previously covered good set. If the algorithm is able to cluster at least $n - b$ points, it must cover every good set because the size of each good set is larger than b. So it must report k clusters where each cluster contains points from a distinct good set.

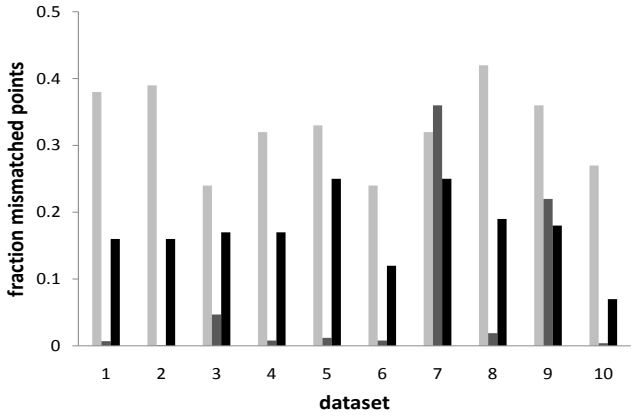


Figure 3.2: Comparing the performance of k-means in the embedded space (light gray), *Landmark-Clustering* (gray), and *Landmark-Clustering-Min-Sum* (black) on 10 data sets from Pfam. Data sets **1-10** are created by uniformly at random choosing 8 families from Pfam of size s, $1000 \leq s \leq 10000$.

□

3.4 Empirical Study

We present some preliminary results of testing our *Landmark-Clustering-Min-Sum* algorithm on protein sequence data. We use the same data sets as in Section 2.5, and compare with the same algorithms. We also show the results of the *Landmark-Clustering* algorithm from Chapter 2 on these data, and use the same number of distance queries for both limited information algorithms ($30k$ landmarks/queries for each data set, where k is the number of clusters).

In order to run *Landmark-Clustering-Min-Sum* we need to set the parameter T. Because in practice we do not know its correct value, we use increasing estimates of T until we cluster enough of the points in the data set; this procedure is similar to the algorithm for the case when we don't know the optimum objective value OPT and hence don't know T. As before, in order to compare a computationally derived clustering to the one given by the gold-standard classification, we use the distance measure from the theoretical part of our work.

Figure 3.2 shows the results of our experiments on the Pfam data sets. We can see that

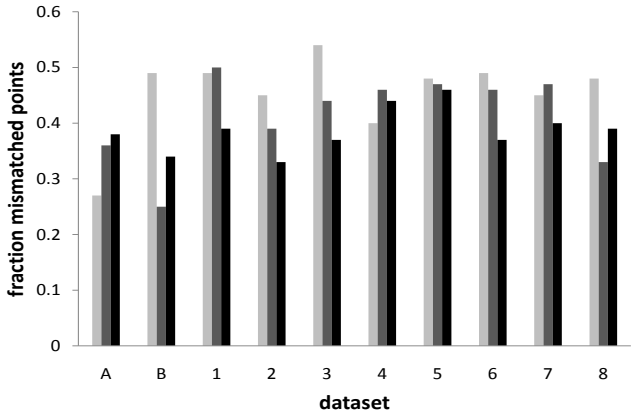


Figure 3.3: Comparing the performance of spectral clustering (light gray), *Landmark-Clustering* (gray), and *Landmark-Clustering-Min-Sum* (black) on 10 data sets from SCOP. Data sets **A** and **B** are the two main examples from Paccanaro et al. [PCS06], the other data sets (**1-8**) are created by uniformly at random choosing 8 superfamilies from SCOP of size s, $20 \leq s \leq 200$.

Landmark-Clustering-Min-Sum outperforms *k-means in the embedded space* on each data set. However, it does not perform better than the original *Landmark-Clustering* algorithm on most of these data sets. When we investigate the structure of the ground truth clusters in these data sets, we see that the diameters of the clusters are roughly the same. When this is the case the original algorithm will find accurate clusterings as well. Still, *Landmark-Clustering-Min-Sum* tends to give better results when the original algorithm does not work well (data sets 7 and 9).

Figure 3.3 shows the results of our computational experiments on the SCOP data sets. We can see that the three algorithms are comparable in performance here. These results are encouraging because the spectral clustering algorithm significantly outperforms other clustering algorithms on these data [PCS06]. Moreover, the spectral algorithm needs the full distance matrix as input and takes much longer to run. When we examine the structure of the SCOP data sets, we find that the diameters of the ground truth clusters vary considerably, which resembles the structure implied by the (c, ϵ)-property for the *min-sum* objective function, assuming that the target clusters vary in size. Still, most of the time the product of the cluster sizes and their diameters varies, so it does not quite look like what we assume in the theoretical part of this

work.

We plan to conduct further studies to find data where clusters have different scale and there is an inverse relationship between cluster sizes and their diameters. This may be the case for data that have many outliers, and the correct clustering groups sets of outliers together rather than assigns them to arbitrary clusters. The algorithm presented in this chapter will consider these sets to be large diameter, small cardinality clusters. More generally, *Landmark-Clustering-Min-Sum* is more robust because it will give an answer no matter what the structure of the data is like, whereas the *Landmark-Clustering* algorithm from Chapter 2 often fails to find a clustering if there are no well-defined clusters in the data.

Chapter 4

Network Analysis

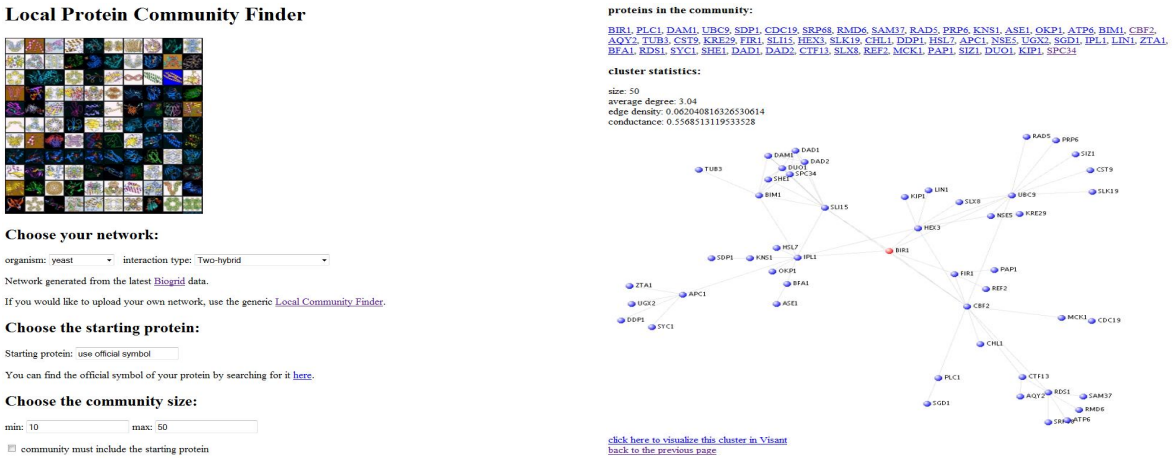
In this chapter we describe our techniques for locally exploring networks. Instead of performing a computation on the entire network our algorithms consider a small part of the graph close to a specified vertex. This approach allows our methods to be very efficient while still giving meaningful information about the local structure of the graph.

This chapter is organized as follows. Section 4.1 gives an overview of our tools to locally explore networks, which find the local community and the nearest neighbors of a queried node in a network input by the user. Sections 4.2, 4.3, and 4.4 describe the algorithms that these tools implement. We conduct a thorough experimental study to show that our methods give meaningful results when applied to protein networks, which is summarized in Section 4.5. Section 4.6 describes Alpha-Centrality, which is another technique that can be used to explore a network, and gives an algorithm to approximate it that can be used on very large networks. Finally, Section 4.7 gives an application of Alpha-Centrality to local search, and contrasts it with PageRank.

4.1 Tools to Locally Explore Networks

Based on the techniques described in the following sections, we build tools that allow users to locally explore protein networks and other networks input by the user. The networks can be constructed from a vast amount of PPI data available from BioGRID [SBR+06], or manually input by the user. The user can choose a network from BioGRID by selecting an organism and a set of interaction types, or upload a custom (undirected) network, which may be weighted. The applications are available through a Web interface and can also be downloaded as command-line programs in the form of single-file Java executables.

Our first tool, named Local Protein Community Finder, is accessible at http://xialab.bu.edu/resources/lpcf. This application uses the *Nibble* algorithm described in



(a) Setting the input **(b)** Visualizing the output

Figure 4.1: Local Protein Community Finder User Interface

Section 4.3, and finds a high-quality community close to the queried vertex.[1] In addition to entering the starting vertex, one can also select the desired cluster size, and whether the reported cluster must contain the starting vertex. The program takes only a few seconds to run, and generates an image of the returned cluster, as well as annotation of the found proteins (Figure 4.1). In addition, the found community can be displayed in VisANT [HHW+09], a popular protein interaction viewer. The other tool, named Protein Network Neighbor Search, is accessible at http://xialab.bu.edu/resources/pnns. It implements the *ApproximatePR-Affinity* algorithm described in Section 4.4.1, and quickly returns a list of nodes sorted by their approximate *PageRank Affinity* to the queried vertex.

4.2 Methods Background

We model a protein network as an undirected, unweighted graph where the nodes are the proteins, and two nodes are connected by an edge if the corresponding proteins are annotated as interacting with each other.

[1] Our tool also implements the *PageRank-Nibble* algorithm from [ACL06], and returns the cluster of lower conductance.

4.2.1 Graph Representation

Formally, a graph is given by a set of vertices V and a set of edges E. We use n to refer to the number of nodes in the graph. The degree of a node $u \in V$, denoted by $d(u)$, is the number of nodes adjacent to u. A graph is often represented by its adjacency matrix. The adjacency matrix of a graph $G = (V, E)$ is defined by

$$A(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise.} \end{cases}$$

4.2.2 Random Walks

We can learn a lot about the structure of a graph by taking a random walk on it. A random walk is a process where at each step we move from some node to one of its neighbors. The transition probabilities are given by edge weights, so in the case of an unweighted network the probability of transitioning from u to any adjacent node is $1/d(u)$. Thus the transition probability matrix (often called the random walk matrix) is the normalized adjacency matrix where each row sums to one:

$$W = D^{-1}A.$$

Here the D matrix is the degree matrix, which is a diagonal matrix given by

$$D(u, v) = \begin{cases} d(u) & \text{if } u = v \\ 0 & \text{otherwise.} \end{cases}$$

In a random walk it is useful to consider a probability distribution vector p over all the nodes in the graph. Here p is a row vector, where $p(u)$ is the probability that the walk is at node u, and $\sum_{u \in V} p(u) = 1$. Because we transition between nodes with probabilities given by W, if p_t is the probability distribution vector at time t, then $p_{t+1} = p_t W$.

In our methods we consider walks that start from a single vertex. We will denote by e_u the probability distribution vector that has all of its probability in vertex u, defined as follows:

$$e_u(i) = \begin{cases} 1 & \text{if } i = u \\ 0 & \text{otherwise.} \end{cases}$$

4.2.3 Conductance

Conductance measures proportion of outgoing edges of a set of nodes in the graph. Given a graph $G = (V, E)$, and a subset of vertices $S \in V$, let us call the edge boundary of S the collection of edges with one point in S and the other outside of S:

$$\partial(S) = \{\{x, y\} \in E \mid x \in S, y \notin S\}.$$

Let us also define the volume of S to be the sum of the degrees of its nodes:

$$\text{vol}(S) = \sum_{x \in S} d(x).$$

The conductance of S is then defined as the ratio of the size of its edge boundary to the volume of the smaller side of the partition:

$$\Phi(S) = \frac{|\partial(S)|}{\min(\text{vol}(S), \text{vol}(\bar{S}))}.$$

The lower the conductance, the better the cluster. Notice that a cluster can have low conductance without being dense.

4.2.4 PageRank

If we modify the random walk to reset at each step with nonzero probability α, it will have a unique steady-state probability distribution. This steady-state distribution, which is known as a PageRank vector, is useful because it tells us how much time we will spend at each vertex in a very long random walk on the graph. For starting vector s, and reset probability α, the PageRank vector $\text{pr}_\alpha(s)$ is the unique solution of the linear system

$$\text{pr}_\alpha(s) = \alpha s + (1 - \alpha)\text{pr}_\alpha(s)W. \tag{4.1}$$

The s vector specifies the probability distribution for where the walk transitions when it

resets. The original PageRank algorithm used a uniform starting vector ($s = \frac{1}{n}\vec{1}$), which gives the global PageRank of each vertex [PBMW98, BP98]. PageRank with non-uniform starting vectors is known as personalized PageRank, and has been used in context-sensitive search on the Web [FR04, JW03].

We can also verify that a PageRank vector can be expressed as a weighted average of random walk vectors [ACL06]:

$$\mathrm{pr}_\alpha(s) = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t (sW^t). \tag{4.2}$$

Here the sW^t term gives the probability distribution of the random walk after t steps. Equation 4.2 shows that the PageRank computation is linear with respect to the starting vector. In other words, $\mathrm{pr}_\alpha(s_1) + \mathrm{pr}_\alpha(s_2) = \mathrm{pr}_\alpha(s_1 + s_2)$, and $c \cdot \mathrm{pr}_\alpha(s) = \mathrm{pr}_\alpha(c \cdot s)$.

In our work we always use starting vectors that have all of their probability in a single vertex, denoted by $\mathrm{pr}_\alpha(e_u)$. This vector is the steady-state probability distribution of a walk that always returns to u at restart, and we will refer to it as the personalized PageRank vector of this vertex.[2] We will use $\mathrm{pr}_\alpha(e_u)[v]$ to denote the amount of probability that v has in $\mathrm{pr}_\alpha(e_u)$, and use a shorthand of $\mathrm{pr}(u \to v)$ for this quantity, dropping the α in the subscript because in our computations it is always fixed. Because the PageRank computation is linear with respect to the starting vector, the global PageRank of v, denoted by $\mathrm{PR}(v)$, satisfies

$$\mathrm{PR}(v) = \frac{1}{n} \sum_u \mathrm{pr}(u \to v). \tag{4.3}$$

Thus $\mathrm{pr}(u \to v)$ can be thought of as the contribution that u makes to the PageRank of v. We can also use Equation 4.2 to derive a more intuitive definition of $\mathrm{pr}(u \to v)$:

$$\mathrm{pr}(u \to v) = \mathrm{pr}_\alpha(e_u)[v] = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t W^t(u, v). \tag{4.4}$$

Here the $W^t(u, v)$ term gives the probability of being at vertex v in t steps, given that the walk starts at u.

[2] A personalized vector generally refers to a non-uniform vector, but here we use this term to refer to a vector which is non-zero in exactly one entry.

4.3 Nibble

Nibble, the local clustering algorithm of Spielman and Teng [ST08], works by conducting a lazy random walk from the starting vertex, and checking the probability distribution vector after each transition for a cluster of low conductance. It is described more formally below.

Algorithm 4.1 Nibble(G, v, ϵ, t_{last})

1: $p_0 = e_v$, $\phi = 1$, $C = \emptyset$
2: **for** $t = 1$ to t_{last} **do**
3: $\quad p_t = Mp_{t-1}$
4: $\quad p_t = [p_t]_\epsilon$
5: $\quad C' = \text{Sweep}(p_t)$
6: $\quad$ **if** $\Phi(C') < \phi$ **then**
7: $\quad\quad \phi = \Phi(C')$
8: $\quad\quad C = C'$
9: $\quad$ **end if**
10: **end for**
11: **return** C

The algorithm maintains a probability distribution vector p over the nodes in the graph, initially with $p_0 = e_v$, which has all of its probability in v. In each iteration we set $p_t = Mp_{t-1}$, where M is the lazy random walk transition probability matrix, and look for a cluster of low conductance by performing a "sweep" of p_t. Nibble iterates for t_{last} steps, and outputs the cluster with the lowest conductance over all iterations.

A sweep is a technique for producing a cut (partition) from a probability distribution vector. Given a vector p, we first order vertices by degree-normalized probability: let $v_1, \ldots, v_n$ be an ordering of the vertices such that $p(v_i)/d(v_i) \geq p(v_{i+1})/d(v_{i+1})$. We then consider sets of vertices v_1 through v_j in this order, which we call the sweep sets. Here j ranges from 1 to the number of vertices with non-zero probability in them. For each sweep set $S_j^p = \{v_1, \ldots, v_j\}$ we compute its conductance $\Phi(S_j^p)$, and report the cluster with lowest conductance.

Nibble uses the truncation operation $p = [p]_\epsilon$, which sets $p(v) = 0$ for every vertex v such that $p(v) < \epsilon d(v)$. We only consider vertices that have non-zero probability in them when performing the random walk and performing a sweep. Therefore we can control the runtime of the algorithm and the number of vertices that it can explore by adjusting the ϵ parameter.

4.4 PageRank Affinity

For two vertices u and v we define their *PageRank Affinity* to be the minimum of the PageRank that u contributes to v and v contributes to u:

$$\textbf{pr-aff}(u, v) = \min(\text{pr}(u \to v), \text{pr}(v \to u)).$$

This quantity can be computed by solving the PageRank equation for $\text{pr}_\alpha(e_u)$ and $\text{pr}_\alpha(e_v)$, and reporting the minimum of the two PageRank contributions. The restart probability of the random walk (α) must be greater than 0 to ensure that $\text{pr}_\alpha(e_u)$ and $\text{pr}_\alpha(e_v)$ have unique solutions, and must be much smaller than 1 to prevent the random walk from returning too often to the starting vertex and being too local. We set α to 0.15, which is typical for computations of PageRank.

4.4.1 Approximating PageRank Affinity

We can also use approximate PageRank to compute closeness between nodes. While it is possible to compute exact PageRank vectors for smaller graphs by solving the PageRank equation, it is computationally infeasible to do this for larger networks. To calculate approximate PageRank, we use the ApproximatePR algorithm from Andersen, Chung, and Lang [ACL06], which computes an ϵ-approximate PageRank vector for a random walk with restart probability α in time $O(\frac{1}{\epsilon\alpha}) = O(\frac{1}{\epsilon})$ if α is constant.

We develop an algorithm that approximates *PageRank Affinity*, which uses ApproximatePR as a subroutine. Our *ApproximatePR-Affinity* algorithm takes a queried vertex v, approximation parameter ϵ, and integer k as input, and returns the k nodes with highest approximate *PageRank Affinity* to v in the graph. The algorithm is outlined below.

Algorithm 4.2 ApproximatePR-Affinity(v, ϵ, k)

$\tilde{\text{pr}}(e_v) = \text{ApproximatePR}(v, \epsilon)$
for each u **do**
  $\tilde{\text{pr}}(v \to u) = \tilde{\text{pr}}(e_v)[u]$
end for
for each u **do**
  $\tilde{\text{pr}}(u \to v) = \tilde{\text{pr}}(v \to u)\frac{d(v)}{d(u)}$
end for
for each u **do**
  $\text{affinity}(u) = \min(\tilde{\text{pr}}(u \to v), \tilde{\text{pr}}(v \to u))$
end for
return k vertices with highest **affinity** scores

We first compute an approximate personalized PageRank vector of v, denoted by $\tilde{\text{pr}}(e_v)$, to approximate the amount of PageRank that v gives to each vertex u, denoted by $\tilde{\text{pr}}(v \to u)$. We then use the observation that for undirected graphs

$$\text{pr}(u \to v) = \text{pr}(v \to u)\frac{d(v)}{d(u)},$$

to approximate the PageRank contribution of each vertex in the graph to v. We then calculate the *affinity* to v of each vertex u as $\min(\tilde{\text{pr}}(u \to v), \tilde{\text{pr}}(v \to u))$, and return the k nodes with highest *affinity* values.

We can verify that the runtime of this procedure is $O(\frac{k}{\epsilon})$ and the amount of error in the *affinity* of vertices u and v, denoted by $\tilde{\text{pr-aff}}(u, v)$, is at most the product of ϵ and the larger of their degrees:

$$\text{pr-aff}(u, v) \geq \tilde{\text{pr-aff}}(u, v) \geq \text{pr-aff}(u, v) - \epsilon \cdot \max(d(u), d(v)).$$

4.4.2 Relationship with Cluster Co-Membership

If u and v are in the same cluster, both $\text{pr}(u \to v)$ and $\text{pr}(v \to u)$ are likely to be high. It is proved in [ACL06] that for any set C, there is a subset of vertices $C' \subseteq C$, such that for any vertex $u \in C'$, the personalized PageRank vector of u, denoted by $\text{pr}_\alpha(e_u)$, satisfies

$$\sum_{v \in C} \mathrm{pr}_\alpha(e_u)[v] \geq 1 - \frac{\Phi(C)}{\alpha}.$$

In other words, $\mathrm{pr}(u \to v) = \mathrm{pr}_\alpha(e_u)[v]$ is high on average if u and v are in the same good (low-conductance) cluster C and $u \in C'$. Moreover, the set C' is large, as the sum of degrees of its nodes, denoted by $\mathrm{vol}(C')$, satisfies $\mathrm{vol}(C') \geq \mathrm{vol}(C)/2$.

4.5 Empirical Study on Protein Networks

We conduct a series of computational experiments to determine whether our methods for locally exploring networks are successful at finding functionally related proteins in PPI networks, and compare their effectiveness to other techniques. To validate our methods we use a gold-standard listing of functional units, and a reliable measure of functional similarity, which is described in the next section. Section 4.5.2 describes the protein networks that we use in our study, and the results of our experiments follow in Section 4.5.3.

4.5.1 Measuring Functional Distance

In order to compute functional distances for pairs of proteins, we use functional distances from Yu et al. [YJG07]. These values are derived using the Gene Ontology (GO) Process classification scheme, where **functional-distance**(a, b) is the number of gene pairs that share all of the least common ancestors of a and b in the classification hierarchy. A low functional distance score means that two proteins are more functionally related, because there are few protein pairs that have the same functional relationship.

The functional distance measure of Yu et al., which the authors refer to as the "total ancestry measure for GO," has the obvious advantage that it considers all known functions of a pair of proteins, allowing for a great degree of precision in assessing functional similarity. Moreover, unlike other methods that derive distances from the GO classification scheme, this method is very resilient to rough functional descriptions, because it still assigns low distances to pairs of proteins that only share very broad terms, as long as there are few other protein pairs that share all of those terms.

Functional distances from Yu et al. [YJG07] can be quite large, yet differences in scores at

the low end are more significant than differences at the high end, which is why we take the logarithm in our calculations:

$$d(a, b) = log_{10}(\textbf{functional-distance}(a, b))$$

4.5.2 The Protein Networks

The protein interaction data that we use in our study is from BioGRID [SBR+06], Versions 2.0.44 and 2.0.53. BioGRID lists interacting protein pairs, and for each pair gives the experimental method used to observe the interaction, as well as the source that submitted it. In our study we use several **yeast** protein-protein interaction (PPI) networks formed from interactions detected by different methods.

Two of the networks, where protein interactions are detected from bait-and-prey type experiments are Affinity Capture Western (referred to as **AC-Western**), and Affinity Capture MS (**AC-MS**). These networks tend to be much more cliquish and contain dense components, which is due to the nature of the experiment used to detect the interactions. A single protein (bait) is used to pull in a set of other proteins (prey), and an interaction is predicted either between the bait and each prey (the spoke model), or between every protein in the group (the matrix model) [HDB+05]. We also use Two-Hybrid data in our study. Two-Hybrid methods detect binary interactions, therefore PPI networks based on Two-Hybrid data tend to be less dense and cliquish than ones derived from Affinity Capture experiments.

In addition to using a network formed from the union of all Two-Hybrid interactions listed in BioGRID (**Two-hybrid**), we also consider a subset of this data submitted by Yu et al. [YBY+08] (**Two-hybrid-2**). This network is sparser, but is believed to be of higher quality. We use this network in some of our experiments if the other results are inconclusive.

4.5.3 Results

We evaluate the biological significance of *PageRank Affinity* in protein networks and compare it to other graph-theoretic measures of closeness, which are summarized below.

- *Shortest Path* ranks node pairs by shortest path distance; we use the multiplicity of the

shortest path to break ties between pairs that are the same distance apart.

- *Common Neighbors* ranks node pairs by how many direct neighbors they share in the network. Pairs with more neighbors in common are ranked higher than pairs with fewer shared neighbors.

- *Cliques* ranks node pairs that are part of a larger clique higher than pairs that are part of a smaller clique. We find all maximal cliques to compute this measure, but we can evaluate the closeness of only a small number of node pairs because most pairs are not part of any clique.

- *Partitioning* ranks node pairs that are part of a denser cluster higher than pairs that are part of a less dense cluster. To cluster the network we use Metis, a partitioning algorithm that finds high-quality clusters in the graph [AK06]. As with *Cliques*, we can evaluate the closeness of only a small number of pairs because most node pairs do not share a cluster.

To calculate the *PageRank Affinity* of all pairs of nodes in a network, we compute a personalized PageRank vector of each vertex, and then calculate a *PageRank Affinity* score for each pair from their personalized PageRank vectors. In order to evaluate the output of our neighborhood search tool, in each network we also calculate the *Approximate PageRank Affinity* of protein pairs by running the *ApproximatePR-Affinity* algorithm from each vertex.

We first consider how well our measure of closeness reflects functional ties given by a gold-standard manual classification of protein complexes in Mewes et al. [MAA+04]. Figures 4.2, 4.3, and 4.4 displays the results as a ROC curve. We divide all protein pairs in each network in our study into positives and negatives, the **positives** are protein pairs that are co-complexed, and the **negatives** are all other pairs. We use T to refer to the number of positives and N to refer to the number of negatives. We then rank protein pairs by each measure of closeness. Each measure is evaluated by the number of true positives and false positives in a particular percentile of the ranking, where the **true positives** are the positive pairs and the **false positives** are the negative pairs. We use TP to refer to the number of true positives and FP to refer to the number of false positives. In each figure the x-axis lists the *false positive rate*, which is defined as FP/N, and the y-axis lists the *true positive rate*, which is defined as TP/T. Lines of different

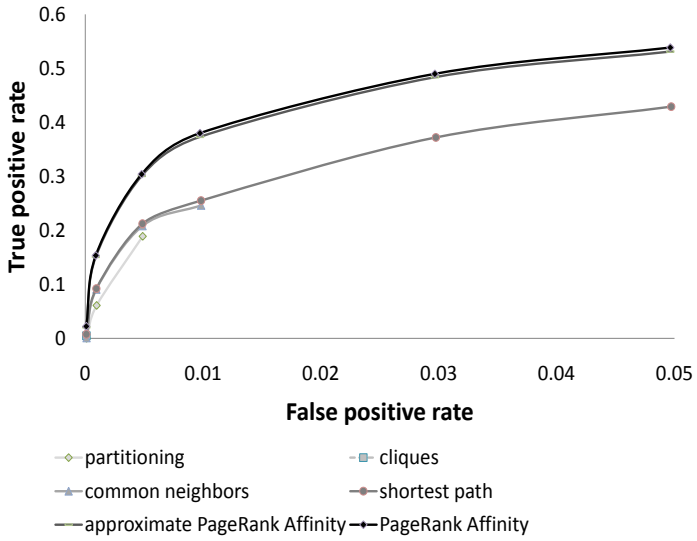


Figure 4.2: Which measure of closeness is best at predicting co-complex membership? The results for the Two-hybrid network.

shades of gray are used to represent the results for the different measures compared. The ROC curves are produced by considering proteins pairs in the top 0.01%, 0.1%, 0.5%, 1%, 3% and 5% of each closeness ranking. A measure that has a higher true positive rate for the same false positive rate is better.

We can see from Figure 4.2 that in the Two-hybrid network *PageRank Affinity* predicts co-complex pairs better than other measures. The same is true for the AC-Western network, although the contrast with other measures of closeness is smaller (Figure 4.3). The picture is different for the AC-MS network, as *Common Neighbors* and *Shortest Path* are as effective as *PageRank Affinity* at predicting co-complex pairs (Figure 4.4). We also note that in all three networks we do not lose much by approximating *PageRank Affinity* rather than computing it exactly.

We also use functional distance data to evaluate the meaning of our closeness measure in protein networks. In each PPI network in our study, we rank protein pairs by *PageRank Affinity* and other measures of closeness, and average the functional distances of the protein pairs in the

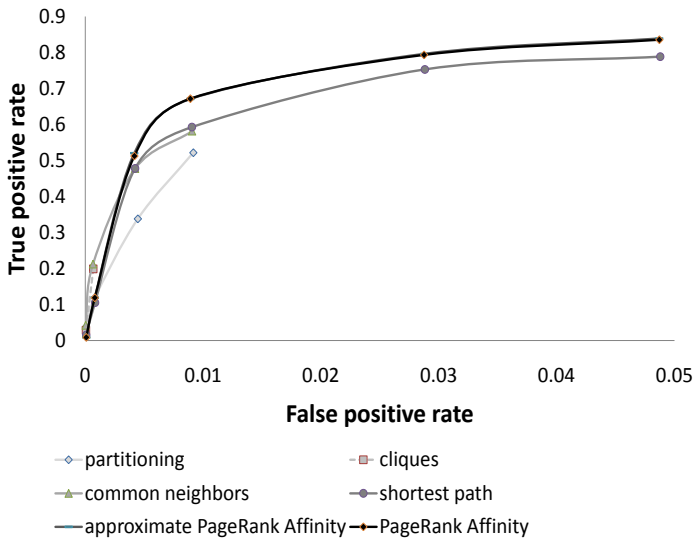


Figure 4.3: Which measure of closeness is best at predicting co-complex membership? The results for the AC-Western network.

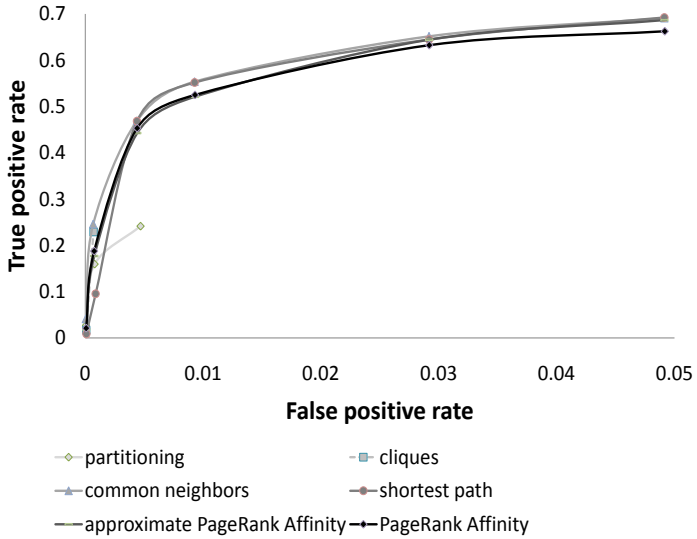


Figure 4.4: Which measure of closeness is best at predicting co-complex membership? The results for the AC-MS network.

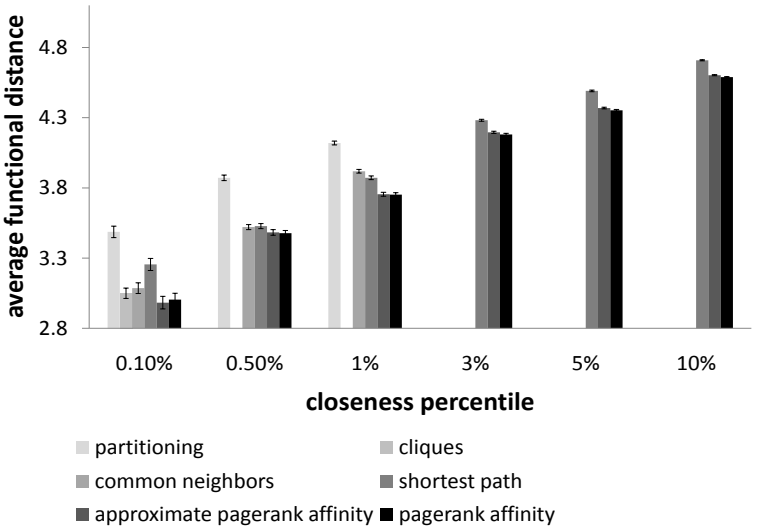


Figure 4.5: Which measure of closeness best correlates with functional distance? The results for the Two-hybrid network.

top k percent of each closeness ranking, for different values of k. The results are presented in Figures 4.5, 4.6, and 4.7. Bars of different shades of gray are used to represent the results for the different closeness measures compared. The x-axis lists different percentiles of the closeness rankings, and the y-axis displays the average functional distance of protein pairs in the top percentile of a particular ranking. Lower values indicate measures that are more biologically meaningful. The average functional distance of two proteins in the genome is **5.8**. We can see that in all three networks protein pairs with high *PageRank Affinity* are more functionally related (have smaller functional distances). Once again, *Approximate PageRank Affinity* is almost as biologically meaningful, significantly outperforming other measures.

Our conclusion is that we can learn a lot about the functional relationships of proteins by considering *PageRank Affinity* in a PPI network. Protein pairs with high *PageRank Affinity* are much more likely to be functionally related, as evidenced by membership in the same protein complex and low functional distance.

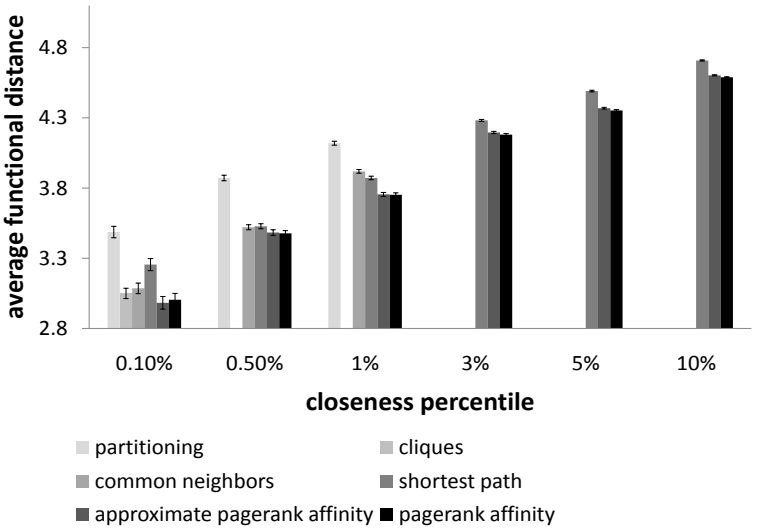


Figure 4.6: Which measure of closeness best correlates with functional distance? The results for the AC-Western network.

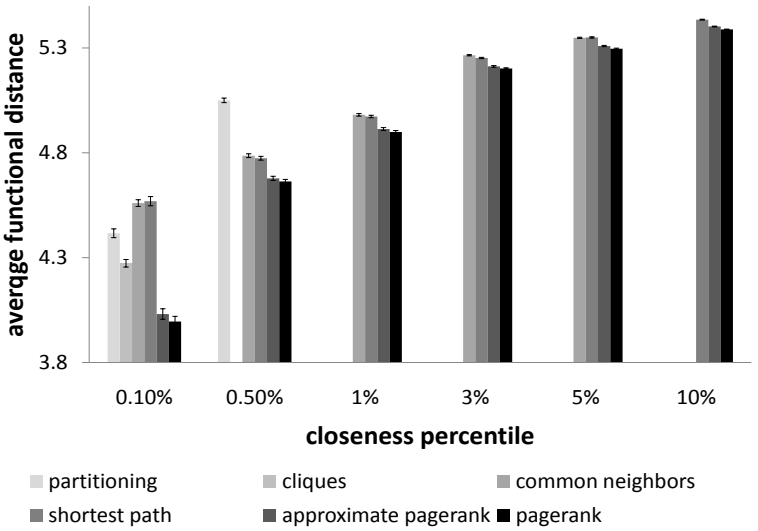


Figure 4.7: Which measure of closeness best correlates with functional distance? The results for the AC-MS network.

We perform additional experiments to determine whether our local clustering technique is relevant in protein networks. We evaluate the performance of *Nibble* and other partitioning algorithms by the graph-theoretic quality of the found clusters and their functional coherence. In a protein network represented by a graph $G = (V, E)$ we define the functional coherence of a cluster C to be the difference between the average functional distance of two proteins in the network and the average pairwise functional distance of proteins in the cluster:

$$\textbf{functional-coherence}(C) = \frac{\sum_{u \neq v \in V} d(u,v)}{|V|(|V|-1)} - \frac{\sum_{u \neq v \in C} d(u,v)}{|C|(|C|-1)}.$$

To compare the performance of the clustering algorithms, we run all of them from the same set of nodes in each PPI network, and record the conductance and functional coherence of the found clusters. We then average the statistics of every algorithm in each network, and report the standard error to see if the differences are statistically significant. In order to compare global algorithms to local ones, when we use a global clustering algorithm we partition the entire network once, and for starting vertex s consider the cluster containing s. Figures 4.8 and 4.9 display the results of our computational experiments. The algorithms compared are listed along the x-axis, the y-axis specifies the average conductance/functional coherence of clusters found by each algorithm. Bars of different shades of gray are used to represent the results for the four protein networks in which the computational experiments are performed. We can see that Nibble finds clusters of better conductance in all four of the networks. In addition, in three of the four networks *Nibble* finds communities that are more functionally coherent.

4.6 Alpha-Centrality

Alpha-Centrality was proposed by Bonacich [Bon87] to generalize eigenvector centrality to cases when some nodes do not have any in-neighbors. Given an *attenuation* parameter α and a starting vector s, the Alpha-Centrality vector $\mathrm{cr}_\alpha(s)$ is the solution to the following equation:

$$\mathrm{cr}_\alpha(s) = s + \alpha \cdot \mathrm{cr}_\alpha(s)A. \tag{4.5}$$

Solving this equation we can see that $\mathrm{cr}_\alpha(s) = s(I - \alpha A)^{-1}$, where I is the n by n identity matrix. Using the identity $\sum_{t=1}^{\infty} \alpha^t A^t = (I - \alpha A)^{-1} - I$, we can see that

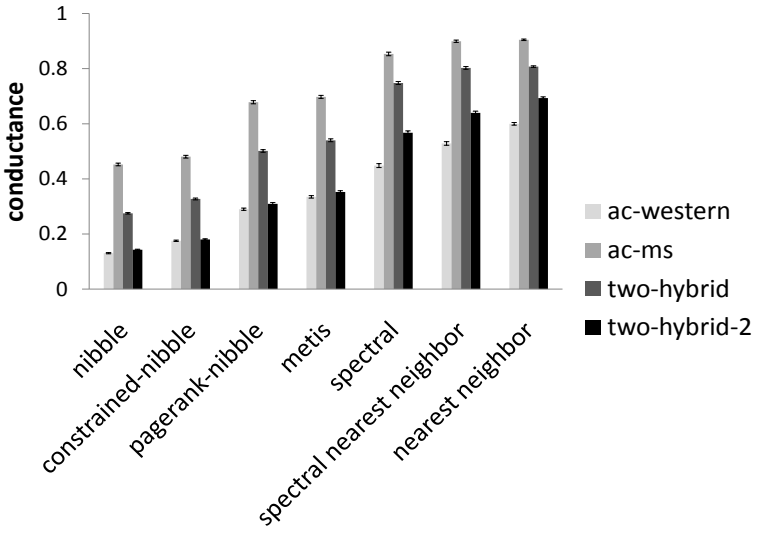


Figure 4.8: Average conductance of clusters found by each algorithm, lower values indicate better clusters.

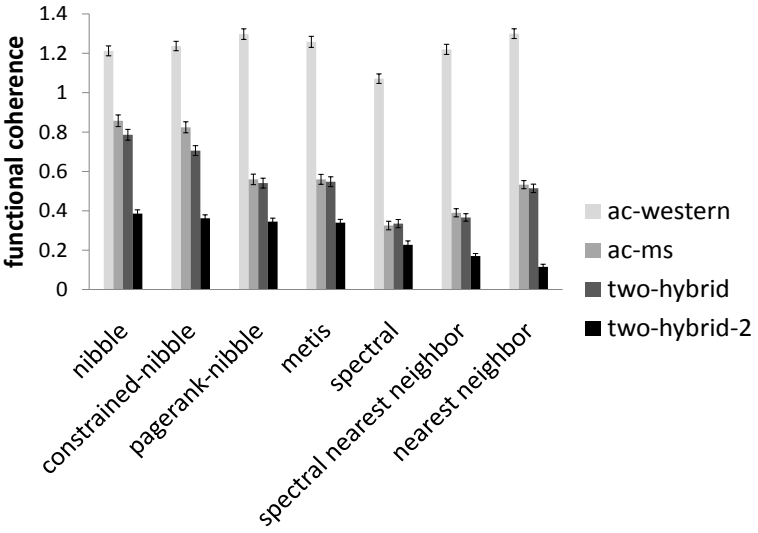


Figure 4.9: Average functional coherence of clusters found by each algorithm, higher values indicate more functionally coherent clusters.

$$\begin{aligned} \mathrm{cr}_\alpha(s) &= s(I-\alpha A)^{-1} \\ &= s(\textstyle\sum_{t=1}^{\infty} \alpha^t A^t + I) \\ &= s\textstyle\sum_{t=1}^{\infty} \alpha^t A^t + s \\ &= s\textstyle\sum_{t=0}^{\infty} \alpha^t A^t. \end{aligned} \tag{4.6}$$

We can verify that $A^t(u,v)$ gives the number of paths of length t between u and v. In order to compute the global centrality scores we set $s = \vec{1}$. Therefore, the centrality of node v is given by the number of paths from u to v for all $u \in V$, with longer paths given less weight based on the value of α.

4.6.1 Approximating Alpha-Centrality

In order to compute the exact Alpha-Centrality vector we have to solve Equation 4.5, which requires us to compute a matrix inverse. Computing a matrix inverse takes $O(n^3)$ time, so this is infeasible for large networks. One way to compute an approximate solution is to use the alternate formulation given in Equation 4.6, and compute $s(I+\alpha A+\alpha^2 A^2+\alpha^3 A^3+\ldots)$, until the α^i coefficient grows sufficiently small. While this technique is effective in practice, computing A^i in each iteration must take at least n^2 time, and it is not clear how many iterations we need to get a good approximation. In this section we present an algorithm for approximating Alpha-Centrality, which has a single parameter that controls both the runtime and the quality of the produced approximation.

A description of our algorithm is given in Algorithm 4.3. Our procedure is similar to the algorithm for approximating PageRank that is given by Andersen, Chung, and Lang [ACL06]. Our algorithm takes the network, the starting vector s, α, and an approximation parameter δ ($0 < \delta \leq 1$) as input, and computes an approximate Alpha-Centrality vector where each entry has error of at most δ (see Theorem 4.1).

In order to approximate a centrality vector with starting vector s, we maintain an *approximate* centrality vector $\tilde{cr}$ and a *residual* vector r. Initially r is equivalent to the staring vector s; the algorithm iteratively moves content from r to $\tilde{cr}$ until each entry in r is small. We give a proof that throughout the execution of the algorithm the error in the approximate centrality vector is the amount of content remaining in the residual vector.

Algorithm 4.3 Approximate-Centrality$(V, E, s, \alpha, \delta)$

```
1: ε = δ||s||_1/n;
2: r = s;
3: Queue q = new Queue();
4: for each u ∈ V do
5:    c̃r(u) = 0;
6:    if r(u) > ε then
7:       q.add(u);
8:    end if
9: end for
10: while q.size() > 0 do
11:    u = q.dequeue();
12:    c̃r(u) = c̃r(u) + r(u);
13:    T = α · r(u);
14:    r(u) = 0;
15:    for each v ∈ N(u) do
16:       r(v) = r(v) + T · w(u, v);
17:       if !q.contains(v) and r(v) > ε then
18:          q.add(v);
19:       end if
20:    end for
21: end while
22: return c̃r;
```

We assume that the graph may be directed and weighted, and use $w(u, v)$ to denote the weight of the edge from u to v. We use $N(u)$ to refer to the out-neighbors of u: $N(u) = \{v \in V | (u, v) \in E\}$. In addition, we use $d_{\text{out}}(u)$ to specify the out-degree of u: $d_{\text{out}}(u) = \sum_{v \in N(u)} w(u, v)$, and use $d_{\max}$ to refer to the maximum out-degree of any node in the graph.

We next give our formal performance guarantee for Algorithm 4.3.

Theorem 4.1. *Given an $\alpha \leq \frac{c}{d_{\max}}$ for some $c < 1$ and a uniform starting vector s, the vector $\tilde{cr}$ output by* Approximate-Centrality *satisfies $[\text{cr}(s)](u) \geq \tilde{cr}(u) \geq [\text{cr}(s)](u)(1 - \delta)$ for each vertex $u \in V$. The runtime of the algorithm is $O(\frac{n}{\delta} d_{max})$.*

4.6.2 Algorithm Analysis

We next give our proof of Theorem 4.1. Our arguments depend on the linearity of the Alpha-Centrality computation with respect to the starting vector, which is easy to verify. From

Equation 4.6 we can see that $\text{cr}_\alpha(s_1) + \text{cr}_\alpha(s_2) = \text{cr}_\alpha(s_1 + s_2)$, and $c \cdot \text{cr}_\alpha(s) = \text{cr}_\alpha(c \cdot s)$.

When the α parameter is fixed, we use $\text{cr}(s)$ to denote $\text{cr}_\alpha(s)$. We will also use $[\text{cr}(s)](u)$ to refer to how much content vertex u has in $\text{cr}(s)$.

Proof. Lemma 4.2 argues that $\tilde{cr} = \text{cr}(s - r) = \text{cr}(s) - \text{cr}(r)$ throughout the execution of the algorithm, so we have $\tilde{cr}(u) = [\text{cr}(s)](u) - [\text{cr}(r)](u)$ for all vertices $u \in V$. Given a uniform starting vector s, $s(u) = ||s||_1/n$ for all $u \in V$. The algorithm terminates when $r(u) \leq \epsilon$ for all $u \in V$, so we choose $\epsilon = \delta \cdot ||s||_1/n = \delta s(u)$ such that upon completion $r(u) \leq \delta s(u)$ for all $u \in V$.

Clearly, $[\text{cr}(s)](u) \geq \tilde{cr}(u)$ because r and $\text{cr}(r)$ are non-negative. We can also show that given that $r(u) \leq \delta s(u)$ for all $u \in V$, $[\text{cr}(r)](u) \leq \delta[\text{cr}(s)](u)$ for all vertices $u \in V$. It follows that $\tilde{cr}(u) = [\text{cr}(s)](u) - [\text{cr}(r)](u) \geq [\text{cr}(s)](u)(1 - \delta)$. Therefore we can see that indeed $[\text{cr}(s)](u) \geq \tilde{cr}(u) \geq [\text{cr}(s)](u)(1 - \delta)$ for all vertices $u \in V$.

We assume that α is chosen such that $\alpha \leq \frac{c}{d_{\max}}$ for some constant $c < 1$, where $d_{\max}$ is the largest out-degree of any node in the graph. In order to bound the runtime of the algorithm, consider that each iteration of the while-loop decreases the sum of the entries of r by $(1-\alpha \cdot d_{\text{out}}(u))r(u) > (1-\alpha \cdot d_{\text{out}}(u))\epsilon \geq (1-\alpha \cdot d_{\max})\epsilon \geq (1-c)\epsilon$. Because $r = s$ at initialization and each iteration decreases $||r||_1$ by at least $(1 - c)\epsilon$, the number of iterations i must satisfy $i(1 - c)\epsilon \leq ||s||_1$. Therefore the number of iterations may be at most $\frac{||s||_1}{(1-c)\epsilon} = O(||s||_1/\epsilon)$. The cost of each iteration is proportional to the out-degree of the node that is dequeued, so the worst-case runtime of the algorithm is $O(||s||_1/\epsilon \cdot d_{\max})$. For our choice of ϵ this is equivalent to $O(\frac{n}{\delta} d_{\max})$. □

Lemma 4.2. *The invariant $\tilde{cr} = \text{cr}(s - r)$ is maintained throughout the execution of the while-loop.*

Proof. Before the loop starts, we have $r = s$ and $\tilde{cr} = \vec{0}$, so $\text{cr}(s - r) = \text{cr}(\vec{0}) = \vec{0} = \tilde{cr}$. We can also show that if $\tilde{cr} = \text{cr}(s - r)$ holds prior to an iteration of the loop, then $\tilde{cr}' = \text{cr}(s - r')$ is still true after the iteration, where $\tilde{cr}'$ and r' are the updated approximate centrality and residual vectors.

We first observe that $\text{cr}(s)A = \text{cr}(sA)$. To see this, consider that by definition $\text{cr}(s) = s + \alpha \cdot \text{cr}(s)A$. Multiplying this equation by A we get $\text{cr}(s)A = sA + \alpha \cdot (\text{cr}(s)A)A$. This shows that $\text{cr}(s)A$ is by definition a centrality vector for starting vector sA. Moreover, we know that the solution to $\text{cr}(sA)$ is unique, so we have $\text{cr}(s)A = \text{cr}(sA)$. This observation shows that we can iteratively compute the centrality vector by expressing $\text{cr}(s)A$ as $\text{cr}(sA)$.

We will write the operations performed inside the while-loop using vector-matrix notation. We use e_u to denote a row vector that has all of its content in vertex u:

$$e_u(i) = \begin{cases} 1 & \text{if } i = u \\ 0 & \text{otherwise.} \end{cases}$$

After an iteration of the loop we have $\tilde{cr}' = \tilde{cr} + r(u)e_u$, and $r' = r - r(u)e_u + \alpha r(u)e_u A$, where u is the vertex that is dequeued in line 11. We next specify the relationship between the approximate centrality and residual vectors before and after an iteration of the while-loop. Consider that

$$\begin{aligned}
\mathrm{cr}(r) &= \mathrm{cr}(r - r(u)e_u) + \mathrm{cr}(r(u)e_u) \\
&= \mathrm{cr}(r - r(u)e_u) + r(u)e_u + \alpha \cdot \mathrm{cr}(r(u)e_u)A \\
&= \mathrm{cr}(r - r(u)e_u) + r(u)e_u + \alpha \cdot \mathrm{cr}(r(u)e_u A) \\
&= \mathrm{cr}(r - r(u)e_u) + r(u)e_u + \mathrm{cr}(\alpha r(u)e_u A) \\
&= \mathrm{cr}(r - r(u)e_u + \alpha r(u)e_u A) + r(u)e_u \\
&= \mathrm{cr}(r') + \tilde{cr}' - \tilde{cr}.
\end{aligned}$$

If $\tilde{cr} = \mathrm{cr}(s - r)$, we have $\mathrm{cr}(r) = \mathrm{cr}(r') + \tilde{cr}' - \mathrm{cr}(s - r)$. It follows that $\tilde{cr}' = \mathrm{cr}(r) - \mathrm{cr}(r') + \mathrm{cr}(s - r) = \mathrm{cr}(r - r' + (s - r)) = \mathrm{cr}(s - r')$. This completes the proof. □

4.7 Applications of Alpha-Centrality

We next show that Alpha-Centrality with personalized starting vectors can be used for local search in a network, and give some examples that illustrate the difference between PageRank and Alpha-Centrality.

4.7.1 Local Search

An Alpha-Centrality vector with a uniform starting vector gives a global centrality measure. Let us denote by $\mathrm{CR}(v)$ the centrality of vertex v, which is the entry corresponding to v in $\mathrm{cr}_\alpha(\vec{1})$.

We can also use Alpha-Centrality with personalized starting vectors to compute *local centrality vectors*. As before, let e_u denote a vector with a 1 in position u and zeroes everywhere else:

$$e_u(i) = \begin{cases} 1 & \text{if } i = u \\ 0 & \text{otherwise.} \end{cases}$$

We can use the vector e_u to compute a personalized centrality vector $\mathrm{cr}_\alpha(e_u)$. We denote by $[\mathrm{cr}_\alpha(e_u)](v)$ the score of v in $\mathrm{cr}_\alpha(e_u)$, and use a shorthand notation of $\mathrm{cr}(u \to v)$ for this quantity. Equation 4.6 shows that the Alpha-Centrality computation is linear with respect to

the starting vector. We can use this fact to show that indeed $\mathrm{cr}(u \to v)$ gives the amount that u contributes to the centrality of v:

$$\mathrm{CR}(v) = \sum_{u \in V} \mathrm{cr}(u \to v). \tag{4.7}$$

Moreover, using Equation 4.6 we can rewrite $\mathrm{cr}(u \to v)$ as:

$$\mathrm{cr}(u \to v) = [\mathrm{cr}_\alpha(e_u)](v) = \sum_{t=0}^{\infty} \alpha^t A^t(u, v). \tag{4.8}$$

The $A^t(u, v)$ term in Equation 4.8 gives the number of paths of length t from u to v. Therefore to compute $\mathrm{cr}(u \to v)$ we consider the number of paths of all lengths between u and v, with longer paths given less weight based on the value of α. In addition, Equation 4.8 shows that unlike PageRank contributions, Alpha-Centrality contributions are symmetric in undirected networks: $\mathrm{cr}(u \to v) = \mathrm{cr}(v \to u)$ if the adjacency matrix A is symmetric.

We can consider the centrality contribution $\mathrm{cr}(u \to v)$ to be a measure of closeness between vertices u and v. By computing $\mathrm{cr}_\alpha(e_u)$ we can calculate $\mathrm{cr}(u \to v)$ for all vertices v in the graph. This computation is feasible even for very large networks if we use Algorithm 4.3 to approximate $\mathrm{cr}_\alpha(e_u)$. This approach gives us another meaningful way to efficiently find the closest neighbors of a given vertex in a very large network. Even though the error analysis presented in the previous section assumes that the starting vector is uniform, giving a similar performance guarantee for personalized starting vectors can be the focus of future work.

4.7.2 Differences between PageRank and Alpha-Centrality

We conclude by giving some examples of the difference between the PageRank and Alpha-Centrality computations. If both of these measures produce similar results in most networks, then it is not important which one we use. However, we believe that it is not hard to find examples where one computation gives different results than the other.

Consider the graph given in Figure 4.10. Here if we compare the global PageRank to the global Alpha-Centrality ranking, we find that vertex 4 has the third highest PageRank, but the highest Alpha-Centrality score (for all reasonable settings of α that were tested). The reason for this discrepancy is that the PageRank of each vertex is proportional to how much it is visited

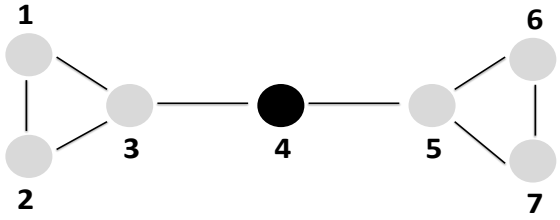
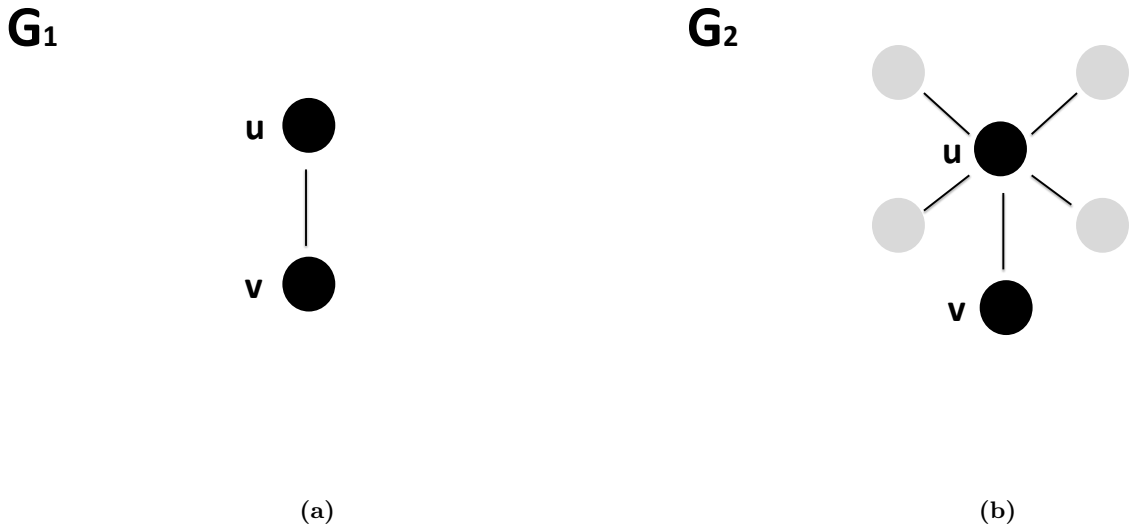


Figure 4.10: Comparing global PageRank and Alpha-Centrality rankings.

in a very long random walk on the network. Vertex 4, which is in-between the two 3-cliques in the graph, is not visited as often as vertices 3 and 5, so it has a lower PageRank. On the other hand, vertex 4 has more short paths to the other nodes in the graph, so it has the highest Alpha-Centrality. This example illustrates a general trend: nodes with highest PageRank tend to be in densely connected subgraphs that have few outgoing edges because a random walk that goes into these parts of the graph tends to stay there for a long time. On the other hand, nodes with highest Alpha-Centrality have more short paths to other nodes in the network, so they will be in the center of the network and may lie outside of the clusters.

We next give an example that shows the difference in computing personalized PageRank and Alpha-Centrality vectors. Let us suppose that we want to compute the closeness of v to u in graphs G_1 and G_2, whose subgraphs are shown in Figures 4.11a and 4.11b, using personalized PageRank and Alpha-Centrality. We compute the PageRank contribution $\text{pr}(u \to v)$, and the Alpha-Centrality contribution $\text{cr}(u \to v)$. Let's assume that G_2 is exactly the same as G_1 other than the additional edges that are shown, and that vertex u has no other neighbors than the ones that are shown.

If we consider PageRank contributions, v is much closer to u in G_1 than it is in G_2. If we look at the formulation of $\text{pr}(u \to v)$ in Equation 4.4, we can see that the value contributed to $\text{pr}(u \to v)$ from paths of length 1 will be much larger in G_1 than in G_2. This is true because $W^1(u, v)$, which is the probability of being at v in one step given that the walk starts at u, is much larger in G_1 than in G_2. Assuming that no short paths have been added between u and



(a) **(b)**

Figure 4.11: Comparing PageRank and Alpha-Centrality contributions.

v, and given that the weight of the paths decreases exponentially in their length (so paths of longer length are worth a lot less), it's likely that $\text{pr}(u \to v)$ is much larger in G_1 than in G_2.

On the other hand, if we consider centrality contributions, v is at least as close to u in G_2 as it is in G_1. If we look at the formulation of $\text{cr}(u \to v)$ in Equation 4.8, we can see that the value of $\text{cr}(u \to v)$ in G_2 must be at least as large as this quantity in G_1. This is true because no paths have been removed, so the number of paths from u to v of length k, which is given by $A^k(u, v)$, may only increase in G_2 for every path length k. It does not matter that the probability of taking some of these paths in a random walk (most notably the path of length 1 from u to v) has decreased significantly.

Whether we should use PageRank or Alpha-Centrality contributions to evaluate closeness between nodes depends on the meaning of the edges in the network. Is v less close to u if u has a lot of other neighbors? If the graph is a social network then perhaps the answer is yes, because the more connections a person has, the less attention he/she can devote to each acquaintance. However, if the links represent communication channels, then u can still broadcast to v no matter how many other neighbors u has. Similarly, if the network is a protein network and protein u can make many copies of itself, then the strength of the connection between u and v does not depend on how many other proteins u interacts with.

This simple example illustrates another critical difference between the computations of PageRank and Alpha-Centrality contributions. Alpha-Centrality contributions are symmetric in undirected networks (because a path from u to v is a path from v to u), so $\text{cr}(u \to v)$ is equivalent to $\text{cr}(v \to u)$ in both graphs. However, PageRank contributions are often very asymmetric, which is clearly illustrated in this example. In G_2 the probability of visiting v from u in a short random walk is likely much lower than the probability of visiting u from v, so $\text{pr}(u \to v)$ is likely much smaller than $\text{pr}(v \to u)$. In order to make the computations of pairwise closeness using PageRank contributions symmetric, we have proposed taking the minimum of the two quantities, which in an undirected network corresponds to considering a random walk from the larger-degree node to the smaller degree-node (see Section 4.4). So in our example we would use $\text{pr}(u \to v)$ to evaluate the closeness of u and v in G_2.

References

[ABS10] P. Awasthi, A. Blum, and O. Sheffet. Stability yields a PTAS for k-median and k-means clustering. In *Proc IEEE Sympos on Foundations of Computer Science*, 2010.

[ACL06] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Proc IEEE Foundations of Computer Science*, pages 475–486, 2006.

[AGK+04] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3), 2004.

[AGM+90] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[AJM09] N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *Advances in Neural Information Processing Systems*, 2009.

[AK06] A. Abou and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *Proc IEEE International Parallel and Distributed Processing Sympos*, 2006.

[AKGR04] Saurabh Asthana, Oliver King, Francis Gibbons, and Frederick Roth. Predicting protein complex membership using probabilistic network reliability. *Genome Research*, 14:1170–1175, 2004.

[AKY99a] C. Alpert, A. Kahng, and S. Yao. Spectral partitioning with multiple eigenvectors. *Discrete Applied Mathematics*, 90:3–26, 1999.

[AKY99b] C. Alpert, A. Kahng, and Z. Yao. Spectral partitioning: The more eigenvectors the better. *Discreet Applied Mathematics*, 90:3–26, 1999.

[AV07] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proc ACM-SIAM Sympos on Discrete Algorithms*, 2007.

[BBG09] M. F. Balcan, A. Blum, and A. Gupta. Approximate clustering without the approximation. In *Proc ACM-SIAM Sympos on Discrete Algorithms*, 2009.

[BCH98] S.E. Brenner, C. Chothia, and T.J. Hubbard. Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proceedings of the National Academy of Sciences of the United States of America*, 95(11):6073–6078, 1998.

[BCM+03] C. Brun, F. Chevenet, D. Martin, J. Wojcik, A. Guenoche, and B. Jacq. Functional classification of proteins for the prediction of cellular function from a protein-protein interaction network. *Genome Biology*, 5:R6, 2003.

[BCR01] Y. Bartal, M. Charikar, and D. Raz. Approximating min-sum k-clustering in metric spaces. In *Proc ACM Sympos on Theory of Computing*, 2001.

[BD07] S. Ben-David. A framework for statistical clustering with constant time approximation algorithms for k-median and k-means clustering. *Machine Learning*, 66(2-3):243–257, 2007.

[BH03] G. Bader and C. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(2), 2003.

[BL01] Phillip Bonacich and Paulette Lloyd. Eigenvector-like measures of centrality for asymmetric relations. *Social Networks*, 23(3):191–201, 2001.

[BLR08] P. Biswal, J. R. Lee, and S. Rao. Eigenvalue bounds, spectral partitioning, and metrical deformations via flows. In *Proc IEEE Foundations of Computer Science*, pages 751–760, 2008.

[Bon87] Phillip Bonacich. Power and centrality: a family of measures. *The American Journal of Sociology*, 92(5):1170–1182, 1987.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30:107–117, 1998.

[CcS05] Tolga Can, Orhan Çamoğlu, and Ambuj K. Singh. Analysis of protein-protein interaction networks using random walks. In *Proc International Workshop on Bioinformatics*, pages 61–68, New York, NY, USA, 2005. ACM.

[Cla05] A. Clauset. Finding local community structure in networks. *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, 72:026132, 2005.

[CS07] A. Czumaj and C. Sohler. Sublinear-time approximation algorithms for clustering via random sampling. *Random Structures and Algorithms*, 30(1-2):226–256, 2007.

[CS09] K.C. Chipman and A.K. Singh. Predicting genetic interactions with random walks on biological networks. *BMC Bioinformatics*, 10(17), 2009.

[CSW06] N. Chua, W. Sung, and L. Wong. Exploiting indirect neighbours and topological weight to predict protein function from protein-protein interactions. *Bioinformatics*, 22:1623–1630, 2006.

[CY06] J. Chen and B. Yuan. Detecting functional modules in the yeast protein-protein interaction network. *Bioinformatics*, 22:2283–2290, 2006.

[Das02] S. Dasgupta. Performance guarantees for hierarchical clustering. In Jyrki Kivinen and Robert Sloan, editors, *Computational Learning Theory*, volume 2375 of *Lecture Notes in Computer Science*, pages 235–254. Springer Berlin / Heidelberg, 2002.

[FLGC02] G. Flake, S. Lawrence, C. Giles, and F. Coetzee. Self-organization and identification of web communities. *Computer*, 35:66–70, 2002.

[FMT+10] R.D. Finn, J. Mistry, J. Tate, P. Coggill, A. Heger, J.E. Pollington, O.L. Gavin, P. Gunesekaran, G. Ceric, K. Forslund, L. Holm, E.L. Sonnhammer, S.R. Eddy, and A. Bateman. The pfam protein families database. *Nucleic Acids Research*, 38:D211–222, 2010.

[FR04] D. Fogaras and B. Racz. Towards scaling fully personalized pagerank. In *Proc Workshop on Algorithms and Models for the Web-Graph*, pages 105–117, 2004.

[GKR98] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Proc ACM Conf on Hypertext and Hypermedia*, pages 225–234, 1998.

[GN02] M. Girvan and MEJ Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99:7821–7826, 2002.

[Gon85] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[GR03] D. Goldberg and F. Roth. Assessing experimentally derived interactions in a small world. *Proceedings of the National Academy of Sciences of the United States of America*, 100:4372–4376, 2003.

[Hav03] T. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Trans on Knowledge and Data Engineering*, 15:784–796, 2003.

[HDB+05] J. Han, D. Dupuy, N. Bertin, M. Cusick, and M. Vidal. Effect of sampling on topology predictions of protein-protein interactions. *Nature Biotechnology*, 23:839–844, 2005.

[HHW+09] Z. Hu, J. Hung, Y. Wang, Y. Chang, C. Huang, M. Huyck, and C. DeLisi. Visant 3.5: multi-scale network visualization, analysis and inference based on the gene ontology. *Nucleic Acids Research*, 37:W115–121, 2009.

[HK92] L. Hagen and A. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11:1074–1085, 1992.

[HZRB07] H. Huang, L. Zhang, F. Roth, and J. Bader. *Probabilistic Paths for Protein Complex Inference*, volume 4532, pages 14–28. Springer Berlin / Heidelberg, 2007.

[JW03] G. Jeh and J. Widom. Scaling personalized web search. In *Proc World Wide Web Conf*, pages 271–279, 2003.

[Kat53] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18:39–43, 1953.

[KCY+06] Nevan Krogan, Gerard Cagney, Haiyuan Yu, Gouqing Zhong, Xinghua Guo, Alexandr Ignatchenko, Joyce Li, Shuye Pu, Nira Datta, Aaron Tikuisis, Thanuja Punna, Jose Peregrin-Alvarez, Michael Shales, Xin Zhang, Michael Davey, Mark Robinson, Alberto Paccanaro, James Bray, Anthony Sheung, Bryan Beattie, Dawn Richards, Veronica Canadien, Atanas Lalev, Frank Mena, Peter Wong, Andrei Starostine, Myra Canete, James Vlasblom, Samuel Wu, Chris Orsi, Sean Collins, Shamanta Chandran, Robin Haw, Jennifer Rilstone, Kiran Gandi, Natalie Thompson, Gabe Musso, Peter St Onge, Shaun Ghanny, Mandy Lam, Gareth Butland, Amin Altaf-U, Shigehiko Kanaya, Ali Shilatifard, Erin O'Shea, Jonathan Weissman, James Ingles, Timothy Hughes, John Parkinson, Mark Gerstein, Shoshana Wodak, Andrew Emili, and Jack Greenblatt. Global landscape of protein complexes in the yeast saccharomyces cerevisiae. *Nature*, 440:637–643, 2006.

[Kel06] J. Kelner. Spectral partitioning, eigenvalue bounds, and circle packings for graphs of bounded genus. *SIAM Journal on Computing*, 35:882–902, 2006.

[Kle98] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc ACM Conf on Hypertext and Hypermedia*, pages 604–632, 1998.

[KPJ04] A. King, N. Przulj, and I. Jurisica. Protein complex prediction via cost-based clustering. *Bioinformatics*, 20:3013–3020, 2004.

[KRRT99] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. *Computer Networks*, 31:1481–1493, 1999.

[KVV00] R. Kannan, S. Vempala, and A. Vetta. On clusterings: good, bad and spectral. In *Proc IEEE Foundations of Computer Science*, pages 367–377, 2000.

[LH07] A. Li and S. Horvath. Network neighborhood analysis with the multi-node topological overlap measure. *Bioinformatics*, 23:222–231, 2007.

[MAA+04] H. Mewes, C. Amid, R. Arnold, D. Frishman, U. Guldener, G. Mannhaupt, M. Munsterkotter, P. Pagel, N. Strack, V. Stumpflen, J. Warfsmann, and A. Ruepp. Mips: analysis and annotation of proteins from whole genomes. *Nucleic Acids Research*, 32:D41–44, 2004.

[MBHC95] A.G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

[MBHG05] J.L. Morrison, R. Breitling, D.J. Higham, and D.R. Gilbert. Generank: Using search engine technology for the analysis of microarray experiments. *BMC Bioinformatics*, 6(233), 2005.

[MOP01] N. Mishra, D. Oblinger, and L Pitt. Sublinear time approximate clustering. In *Proc ACM-SIAM Sympos on Discrete Algorithms*, 2001.

[MS01] M. Meila and J. Shi. Learning segmentation by random walks. In *Neural Information Processing Systems*, 2001.

[New04] Mark Newman. Fast algorithm for detecting community structure in networks. *The European Physical Journal B*, 38:321–330, 2004.

[OKA05] K. Okada, S. Kanaya, and K. Asai. Accurate extraction of functional associations between proteins based on common interaction partners and common domains. *Bioinformatics*, 21:2043–2048, 2005.

[ORSS06] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy. The effectiveness of lloyd-type methods for the k-means problem. In *Proc IEEE Foundations of Computer Science*, 2006.

[PBMW98] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[PCS06] A. Paccanaro, J. A. Casbon, and M. A. S. Saqi. Spectral clustering of protein sequences. *Nucleic Acids Research*, 34(5):1571–1580, 2006.

[PDFV05] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 2005.

[SBR+06] C. Stark, B. J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. Biogrid: a general repository for interaction datasets. *Nucleic Acids Research*, 34:D535–9, 2006.

[SJ89] A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82:93–113, 1989.

[SL03] M. Samanta and S. Liang. Predicting protein functions from redundancies in large-scale protein interaction networks. *Proceedings of the National Academy of Sciences of the United States of America*, 100:12579–12583, 2003.

[SM03] V. Spirin and L. Mirny. Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Sciences of the United States of America*, 100:12123–12128, 2003.

[ST07] D. A. Spielman and S. Teng. Spectral partitioning works: Planar graphs and finite element meshes. *Linear Algebra and its Applications*, 421:284–305, 2007.

[ST08] D. A. Spielman and S. Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. arXiv:0809.3232v1 [cs.DS], 2008.

[TC03] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Proc ACM SIGCOMM Conference on Internet Measurement*, pages 143–152, New York, NY, USA, 2003. ACM.

[VW09] J. Vlasblom and S.J. Wodak. Markov clustering versus affinity propagation for the partitioning of protein interaction graphs. *BMC Bioinformatics*, 10(99), 2009.

[YBY+08] H. Yu, P. Braun, M. A. Yildirim, I. Lemmens, K. Venkatesan, J. Sahalie, T. Hirozane-Kishikawa, F. Gebreab, N. Li, N. Simonis, T. Hao, J. F. Rual, A. Dricot, A. Vazquez, R. R. Murray, C. Simon, L. Tardivo, S. Tam, N. Svrzikapa, C. Fan, A. S. de Smet, A. Motyl, M. E. Hudson, J. Park, X. Xin, M. E. Cusick, T. Moore, C. Boone, M. Snyder, F. P. Roth, A. L. Barabasi, J. Tavernier, D. E. Hill, and M. Vidal. High-quality binary protein interaction map of the yeast interactome network. *Science*, 322:104–110, 2008.

[YH07] A. Yip and S. Horvath. Gene network interconnectedness and the generalized topological overlap measure. *BMC Bioinformatics*, 8(22), 2007.

[YJG07] H. Yu, R. Jansen, and M. Gerstein. Developing a similarity measure in biological function space. *Bioinformatics*, 23:2163–2173, 2007.

Curriculum Vitae

Konstantin Voevodski

Boston University
Computer Science Department
111 Cummington Street
Boston, MA 02215

187 Westminster Rd.
Weymouth, MA 02189

Education

- Ph.D. Computer Science, Boston University, 2011 (expected).
- B.A. Computer Science, Boston University, 2005.

Research Interests

My research interests are clustering and network analysis with applications to computational biology and the social sciences. In particular, I am interested in modeling and detecting community structure in biological and social networks, and designing efficient algorithms for analyzing large real-world datasets.

Publications

- **Efficient Clustering with Limited Distance Information**. K. Voevodski, M. F. Balcan, H. Roglin, S. Teng, and Y. Xia. Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence, *UAI* 2010, pages 632-641.
- **Quantitative Gene Expression Profiles in Real Time From Expressed Sequence Tag Databases**. V. Funari, K. Voevodski, D. Leyfer, L. Yerkes, D. Cramer, and D. Tolan. *Gene Expression*, 14(6): 321-336, 2010.
- **Spectral Affinity in Protein Networks**. K. Voevodski, S. Teng, and Y. Xia. *BMC Systems Biology*, 3:112, 2009.
- **Finding Local Communities in Protein Networks**. K. Voevodski, S. Teng, and Y. Xia. *BMC Bioinformatics*, 10:297, 2009.

Manuscripts

- **Non-Conservative Diffusion and its Application to Social Network Analysis**. R. Ghosh, K. Lerman, T. Surachawala, K. Voevodski, and S. Teng. arXiv:1102.4639v1[cs.SI], 2011. Submitted to the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, *KDD* 2011.

- **Clustering Protein Sequences Given the Approximation Stability of the Min-Sum Objective Function**. arXiv:1101.3620v1[cs.DS], 2011. K. Voevodski, M. F. Balcan, H. Roglin, S. Teng, and Y. Xia. Learning Workshop organized by the Computational and Biological Learning Society, 2011.

- **Efficient Clustering with Limited Distance Information**. K. Voevodski, M. F. Balcan, H. Roglin, S. Teng, and Y. Xia. arXiv:1009.5168v1[cs.DS], 2010. Full version of *UAI* 2010 paper, submitted to the *Journal of Machine Learning Research*.

Teaching

Instructor, Boston University

- Data Structures (CAS CS112), Spring 2008.

- Data Structures (CAS CS112), Summer 2007.

Teaching Assistant, Boston University

- Combinatoric Structures (Spring 2011), Data Structures (Spring 2010), Introduction to Computer Science (Spring 2007), Introduction to Computers (Fall 2005).

Awards

- NSF's IGERT fellowship from the ACES Training Program at BU Center for Computational Science.

- Boston University Trustee Scholar (undergraduate).