

October 2021

Robust Algorithms for Clustering with Applications to Data Integration

Sainyam Galhotra
University of Massachusetts Amherst

Follow this and additional works at: https://scholarworks.umass.edu/dissertations_2



Part of the [Data Science Commons](#)

Recommended Citation

Galhotra, Sainyam, "Robust Algorithms for Clustering with Applications to Data Integration" (2021).
Doctoral Dissertations. 2319.
<https://doi.org/10.7275/24169035> https://scholarworks.umass.edu/dissertations_2/2319

This Open Access Dissertation is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.

ROBUST ALGORITHMS FOR CLUSTERING WITH APPLICATIONS TO DATA INTEGRATION

A Dissertation Presented

by

SAINYAM GALHOTRA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2021

College of Information and Computer Sciences

© Copyright by Sainyam Galhotra 2021

All Rights Reserved

ROBUST ALGORITHMS FOR CLUSTERING WITH APPLICATIONS TO DATA INTEGRATION

A Dissertation Presented

by

SAINYAM GALHOTRA

Approved as to style and content by:

Barna Saha, Chair

Arya Mazumdar, Member

Gerome Miklau, Member

Divesh Srivastava, Member

James Allan, Chair of the Faculty
College of Information and Computer Sciences

ACKNOWLEDGMENTS

The last six years at UMass have been some of the most memorable years of my life. During this time, I have been lucky to meet some of the best professors, collaborators, mentors, friends and housemates. Firstly, I would like to thank my advisor Barna Saha, who has been a constant source of inspiration through the graduate school. She has been a very supportive advisor who patiently listened to me and was available to help whenever I needed it. I would not have been able to enjoy graduate school without her efforts.

I would like to thank Divesh Srivastava, who has been no less than an advisor. He has always been available to help me with his wisdom and taught me to learn from mistakes. His calm and positive attitude to deal with paper rejections kept me motivated to work harder and persevere through tough times. I have learned a lot from him.

I would like to thank Arya Mazumdar and Gerome Miklau for serving on my thesis committee and giving insightful feedback. I also had the pleasure to collaborate with Arya Mazumdar on generative models for clustering. He has been very encouraging, helpful and a great mentor.

I am extremely grateful to Donatella Firmani, who has been a constant pillar of support through rejections, worked late nights and has always been an excellent mentor through my PhD.

During summers, I had the privilege to intern with Alkis Polyzotis, Behzad Golshan, Xin Luna Dong, Xiang He, Jun Ma, Wang-Chiew Tan, Udayan Khurana, Oktie Hassanzadeh, Kavitha Srinivas, Horst Samulowitz, Sudip Roy, and Steven Whang. I

am fortunate to have worked with such experienced researchers and learn from their diverse viewpoints. I would especially like to thank Wang-Chiew Tan for hiring me as an intern at Megagon and providing a supportive environment to explore ideas. She has been a great mentor and a close friend who has not only taught me research but also gave great advice on being a dog parent.

My time at Amherst would not have been fun and enjoyable without housemates and friends: Anirudh, Anna, Raghav, Sarwar, Suhas, Soumyabrata, Anirudha, Sandhya, Sandy, Vini, Roopa, Manish and Haresh. I would not have tried doing research during undergrad, if Tarun Mangla would not have convinced me to work with him on a project with my BTech advisor, Amitabha Bagchi. I am grateful to Amitabha for patiently teaching the best practices of doing research. I would also like to thank all my mentors and friends during my time at IIT Delhi and Xerox: Akhil Arora, Cristina Pinotti, Shourya Roy, Sayan Ranu, Geetha Manjunath, Manish Gupta, Raj Sharma, Deepali Semwal, Sonal Patil.

Finally and most importantly, I thank my parents and brother for endless love, support and encouragement. The thesis is dedicated to them.

ABSTRACT

ROBUST ALGORITHMS FOR CLUSTERING WITH APPLICATIONS TO DATA INTEGRATION

SEPTEMBER 2021

SAINYAM GALHOTRA

B.Tech., INDIAN INSTITUTE OF TECHNOLOGY DELHI

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Barna Saha

A growing number of data-based applications are used for decision-making that have far-reaching consequences and significant societal impact. Entity resolution, community detection and taxonomy construction are some of the building blocks of these applications and for these methods, clustering is the fundamental underlying concept. Therefore, the use of accurate, robust and scalable methods for clustering cannot be overstated. We tackle the various facets of clustering with a multi-pronged approach described below.

1. While identification of clusters that refer to different entities is challenging for automated strategies, it is relatively easy for humans. We study the robustness of clustering methods that leverage supervision through an oracle i.e an abstraction of crowdsourcing. Additionally, we focus on scalability to handle web-scale datasets.

2. In community detection applications, a common setback in evaluation of the quality of clustering techniques is the lack of ground truth data. We propose a generative model that considers dependent edge formation and devise techniques for efficient cluster recovery.

TABLE OF CONTENTS

| | Page |
|---|------|
| ACKNOWLEDGMENTS | iv |
| ABSTRACT | vi |
| LIST OF TABLES | xii |
| LIST OF FIGURES | xiv |
| CHAPTER | |
| 1. INTRODUCTION | 1 |
| 1.1 Facets of Clustering | 4 |
| 1.2 Entity Resolution with Supervision | 5 |
| 1.2.1 Robustness | 6 |
| 1.2.2 Scalability | 6 |
| 1.3 Objective-based Clustering with Supervision | 7 |
| 1.4 Clustering with Generative Models | 8 |
| 1.5 Layout | 9 |
| 2. BACKGROUND AND RELATED WORK | 10 |
| 2.1 Clustering with Supervision | 10 |
| 2.1.1 Crowdsourcing and Oracle Abstraction | 11 |
| 2.1.2 Types of Interactions | 11 |
| 2.1.3 Binary Oracle for Entity Resolution | 13 |
| 2.1.4 Comparison Oracle for Objective-based Clustering | 15 |
| 2.2 Clustering with Generative Models for Community Detection | 16 |
| 2.3 Notation | 17 |

Part I: CLUSTERING WITH SUPERVISION

| | |
|---|-----------|
| 3. ENTITY RESOLUTION WITH SUPERVISION: ROBUSTNESS TO NOISE | 20 |
| 3.1 Introduction | 20 |
| 3.2 Preliminaries | 22 |
| 3.3 Random Graph Toolkit | 24 |
| 3.3.1 Algorithms | 27 |
| 3.4 Pipelines and Trade offs | 34 |
| 3.4.1 Progressive F-score: Lazy Pipeline | 35 |
| 3.4.2 Final F-score: Eager Pipeline | 39 |
| 3.4.3 Adaptive Pipeline | 40 |
| 3.4.4 Application Scenarios. | 42 |
| 3.5 Empirical Evaluation | 42 |
| 3.5.1 Comparison with Previous Strategies | 45 |
| 3.5.2 Sparse Graphs | 52 |
| 3.6 Related Work | 54 |
| 3.7 Summary and Future Work | 55 |
| 4. ENTITY RESOLUTION WITH SUPERVISION: SCALABILITY | 57 |
| 4.1 Introduction | 57 |
| 4.2 Blocking Preliminaries | 61 |
| 4.3 Overview of pBlocking | 64 |
| 4.3.1 Computational Complexity | 66 |
| 4.4 Block Building | 67 |
| 4.5 Block Cleaning | 71 |
| 4.6 Analysis of pBlocking | 75 |
| 4.6.1 Pair Recall without Feedback | 75 |
| 4.6.2 Pair Recall with Feedback | 78 |
| 4.7 Experiments | 83 |
| 4.7.1 Benefits of Progressive Blocking | 87 |

| | | |
|-----------|--|------------|
| 4.7.2 | Robustness of Progressive Blocking | 93 |
| 4.7.3 | Progressive Behavior | 94 |
| 4.8 | Related Work | 96 |
| 4.9 | Summary and Future Work | 98 |
| 5. | CLUSTERING WITH COMPARISON ORACLE: DATA SUMMARIZATION | 100 |
| 5.1 | Introduction | 100 |
| 5.1.1 | Chapter Outline and Contributions | 104 |
| 5.2 | Preliminaries | 107 |
| 5.2.1 | Quadruplet Oracle Comparison Query | 108 |
| 5.2.2 | Noise Models | 110 |
| 5.3 | Greedy k -center Clustering Algorithm | 111 |
| 5.4 | Finding Maximum, Farthest and Nearest | 111 |
| 5.4.1 | Adversarial Noise | 112 |
| 5.4.1.1 | Helper Lemmas | 120 |
| 5.4.2 | Probabilistic Noise | 121 |
| 5.5 | k -center Clustering | 125 |
| 5.5.1 | Adversarial Noise | 125 |
| 5.5.2 | Probabilistic Noise | 130 |
| 5.6 | Hierarchical Clustering | 142 |
| 5.7 | Experiments | 145 |
| 5.7.1 | User study | 147 |
| 5.7.2 | Crowd Oracle: Solution Quality & Query Complexity | 149 |
| 5.7.3 | Simulated Oracle: Solution Quality & Query Complexity | 152 |
| 5.8 | Summary and Future Work | 155 |

Part II: GENERATIVE MODELS FOR CLUSTERING

| | | |
|-----------|--|------------|
| 6. | CLUSTERING WITH GENERATIVE MODELS | 158 |
|-----------|--|------------|

| | | |
|-----------|--|------------|
| 6.1 | Introduction | 158 |
| 6.2 | The Geometric Block Model | 162 |
| 6.3 | Real-world Validation | 163 |
| 6.3.1 | Academic Collaboration Network | 163 |
| 6.3.2 | Amazon Co-purchase Network | 164 |
| 6.4 | The Motif-Counting Algorithm | 166 |
| 6.5 | Analysis of the Algorithm | 168 |
| 6.5.1 | Results for Other Motifs | 175 |
| 6.6 | Experimental Results | 182 |
| 6.7 | Summary and Future Work | 186 |
| 7. | CONCLUSIONS AND FUTURE WORK | 187 |
| 7.1 | Future Directions | 188 |
| 7.1.1 | Alternate Forms of Supervision..... | 188 |
| 7.1.2 | Entity Resolution over Data Markets and Federated Data Sources | 188 |
| 7.1.3 | Hierarchical Clustering with Supervision: Taxonomy Construction | 189 |
| 7.1.4 | Overlapping Clusters with Supervision | 191 |
| 7.1.5 | Fair Clustering with Supervision | 192 |
| 7.1.6 | Interpretable Clustering | 192 |
| | BIBLIOGRAPHY | 194 |

LIST OF TABLES

| Table | Page |
|---|------|
| 1.1 Summary of contributions. Our work [91] is an extension of Chapter 6 and is not included in the thesis. | 9 |
| 2.1 Table of Notation. | 18 |
| 3.1 Text data associated with pictures in Figure 3.1. | 21 |
| 3.2 Number of nodes n (i.e., records), number of clusters k (i.e., entities), size of the largest cluster $ C_1 $, reference to the paper where they appeared first, and origin (real or synthetic). The scenario column matches the dataset with scenarios of Section 3.4.3. We provide experiments for both low and high error rate for cora and dblp , thus we match them with both LC and HC. | 44 |
| 4.1 Sample records (we omit schema information) referring to 4 distinct entities. r_i^e represents the i -th record referring to entity e . Records in the first two rows refer to a Chevrolet Corvette C6 ($c6$) and a Z6 ($z6$). Records in the last two rows to a Chevrolet Malibu (ma) and a Citroën C6 (ci) (same model name as Corvette C6 but different car). | 58 |
| 4.2 Number of nodes n (i.e., records), number of clusters k (i.e., entities), size of the largest cluster $ C_1 $, the total number of matches in the data set $ E^+ $ and the reference to the paper where they appeared first. | 82 |
| 4.3 Running time comparison of B and pBlocking with StB1 and DyB1 for BB , TF-IDF for BC and MB for CC . The ‘blocking’ column denotes the time taken to perform blocking and ‘ER’ denotes the time taken to identify matches over blocked pairs. | 89 |
| 4.4 (a) Pair recall of pBlocking on varying ER strategies. (b) Comparison of the final F-score of the Eager method. The blocking graph is computed with pBlocking(StB1, TF-IDF, MB) and B(StB1, TF-IDF, MB) (both with default settings). | 93 |

| | | |
|-----|---|-----|
| 5.1 | F-score comparison of k-center clustering. <code>0q</code> is marked with * as it was computed on a sample of 150 pairwise queries to the crowd ³ . All other techniques were run on the complete dataset using a classifier. | 152 |
| 5.2 | Running time (in minutes) and number of quadruplet comparisons (denoted by # Comp, in millions) of different techniques for <code>dblp</code> dataset under the adversarial noise model with $\mu = 1$. DNF denotes ‘did not finish’..... | 155 |
| 6.1 | On the left we count the number of inter-cluster edges when authors shared the same affiliation and different affiliations. On the right, we count the same for intra-cluster edges. | 164 |
| 6.2 | Distribution of motif count for an edge (u, v) conditioned on the distance between them $d_L(X_u, X_v) = x$, when there are two equal sized clusters. Here $\text{Bin}(n, p)$ denotes a binomial random variable with mean np | 169 |
| 6.3 | Performance on real world networks | 183 |

LIST OF FIGURES

| Figure | Page |
|---|------|
| 1.1 Monument images collected from different sources with pairwise similarity calculated over text description generated by Google Vision API. | 3 |
| 3.1 Example pictures from different travel websites. Edges represent matching probabilities (edges are colored green if probability is more than 0.50 and red otherwise). | 22 |
| 3.2 Comparison of tree-based clusters in the perfect oracle setting (left) and robust clusters with a noisy oracle (right). Positive (negative) answers are represented with solid (dashed) edges. Wrong answers are shown in blue. | 26 |
| 3.3 Sample execution of <code>node()</code> . Positive (negative) answers are represented with solid (dashed) edges. Node fill and stroke colors are representative of true and computed clusters, respectively. Thick edges highlight corrected errors. | 38 |
| 3.4 Comparison of SCC-based <code>adaptive()</code> variations and the original implementation on <code>dense()</code> | 45 |
| 3.5 Comparison of strategies over RR datasets. | 47 |
| 3.6 Comparison of strategies over RS and SS datasets, with constant error probabilities (a-c have $p^+ = p^- = 0.1$ and d has $p^+ = p^- = 0.2$), both for generation of answers and for input to our algorithms. | 47 |
| 3.7 (a) F-score of <code>adaptive()</code> for different error rates in crowd answers (<code>gym</code>). (b) Different settings of β for <code>adaptive()</code> over <code>cora</code> dataset with $p^+ = p^- = 0.1$ | 48 |
| 3.8 Comparison of <code>waldo()</code> and <code>adaptive()</code> | 50 |
| 3.9 (a-b) <code>adaptive()</code> strategy on different datasets. (c-d) Comparison of our strategies over our bibliographic dataset. | 51 |

| | | |
|------|---|-----|
| 4.1 | Illustration of a standard blocking pipeline. Block building, block cleaning and comparison cleaning sub-tasks are highlighted in white. The downstream ER algorithm is shown in gray. Description of each record is reported in Table 4.1. | 58 |
| 4.2 | Block size distribution (standard blocking) for the real cars dataset used in our experiments..... | 59 |
| 4.3 | Pair recall of \mathcal{B} and pBlocking with TF-IDF for \mathcal{BC} and varying \mathcal{BB} and \mathcal{CC} . (a-d) use MB and (e-h) use BLOSS for \mathcal{CC} . CaCl did not finish within 24 hrs on songs and citations data set. | 85 |
| 4.4 | Pair recall of \mathcal{B} and pBlocking with Unif for \mathcal{BC} and varying \mathcal{BB} and \mathcal{CC} . (a-d) use MB and (e-h) use BLOSS for \mathcal{CC} . CaCl did not finish within 24 hrs on songs and citations data set. | 87 |
| 4.5 | Pair recall of \mathcal{B} and pBlocking with varying \mathcal{BB} , TF-IDF for \mathcal{BC} and MB for \mathcal{CC} . CaCl did not finish within 24 hrs on febrl datasets. We observed similar results with BLOSS for \mathcal{CC} | 88 |
| 4.6 | Pair recall of \mathcal{B} and pBlocking with combination of two block building strategies and TF-IDF for \mathcal{BC} and MB for \mathcal{CC} | 90 |
| 4.7 | Time taken by pBlocking and \mathcal{B} with StBl for \mathcal{BB} and TF-IDF for \mathcal{BC} for varying dataset size. | 92 |
| 4.8 | Comparison of F-score of $\mathcal{B}(\text{DyBl}, \text{TF-IDF}, \text{MB})$ and pBlocking(DyBl,TF-IDF,MB) with respect to ER progress. | 92 |
| 4.9 | Progressive behavior of pBlocking with varying feedback frequency and errors in the feedback (cars). | 94 |
| 4.10 | Effect of feedback loop in cars dataset..... | 96 |
| 5.1 | Data summarization example..... | 101 |
| 5.2 | Example for Lemma 1 with $\mu = 1$ | 120 |
| 5.3 | Algorithm 17 returns ‘Yes’ as $d(u, v_i) < d(u, v_j) - 2\alpha$ | 123 |
| 5.4 | Accuracy values (denoted by the color of a cell) for different distance ranges observed during our user study. The diagonal entries refer to the quadruplets with similar distance between the corresponding pairs and the distance increases as we go further away from the diagonal. | 148 |

| | | |
|-----|---|-----|
| 5.5 | Comparison of farthest and NN techniques for crowdsourced oracle queries. | 150 |
| 5.6 | Comparison of Hierarchical clustering techniques with crowdsourced oracle. | 151 |
| 5.7 | Comparison of farthest identification techniques for adversarial and probabilistic noise models. | 152 |
| 5.8 | Comparison of nearest neighbor techniques for adversarial and probabilistic noise model (lower is better). | 153 |
| 5.9 | k -center clustering objective comparison for adversarial and probabilistic noise model. | 154 |
| 6.1 | Histogram: similarity of products bought together (mean ≈ 6) | 165 |
| 6.2 | Histogram: similarity of products not bought together (mean ≈ 2) | 165 |
| 6.3 | Histogram of common neighbors of edges and non-edges in the co-purchase network, from left to right: Book-DVD, Book-Book, DVD-DVD | 165 |
| 6.4 | Triangle motif varying b and minimum value of a that satisfies the accuracy bound for a synthetic dataset with 5000 nodes. | 184 |
| 6.5 | Results of the motif-counting algorithm on a synthetic dataset with 5000 nodes. | 184 |
| 6.6 | Results of the spectral clustering on a synthetic dataset with 5000 nodes. | 185 |
| 7.1 | Example collection of products. Records 1, 2 refer to Regular LED bulbs, 3, 4 to Smart LED bulbs, 5, 6 to halogen bulbs, 7, 8, 9 to tube lights and 10, 11, 12 to laptops. | 189 |
| 7.2 | (a) Hierarchical relationships for products in Figure 7.1. Dark gray nodes like e_9 represent entities. E.g., records 10, 11 and 12 indeed refer to the same ‘Vaio laptop’ (denoted by e_9), sold by different vendors. Light gray nodes ($u_1 - u_7$) represent types. Deepest light gray nodes (u_1, u_2, u_4, u_5) which are parents of dark gray nodes, correspond directly to the “Type” column of Figure 7.1. E.g., u_2 corresponds to Smart LED. (b) Example of a triplet query about nodes 1, 4 and 11. | 190 |

CHAPTER 1

INTRODUCTION

Recent technological advances have revolutionized the adoption of automated systems for high-stake decisions. Artificial Intelligence (AI) applications are used to make decisions such as loan approvals [153], hiring [3], medical diagnosis [179], criminal justice and sentence assessments [23]. The availability of large amounts of data from a variety of data sources has been one of the major contributors to improved quality of such systems. With more than 2.5 quintillion bytes of data generated on the web each day [5], both the amount of data and the number of data sources are growing at an unprecedented rate. For example, there are more than one billion websites on the internet out of which 24 million are e-commerce websites [149]. eBay, one of the largest e-commerce corporations lists more than 1.3 billion products [149]. Integrating data from different sources to construct a holistic representation of the entities enhances their value considerably. This increase in the number of data sources has raised the importance of designing efficient and scalable techniques for data integration and organization.

The ease of availability of data has also led to many sources generating noisy, ambiguous, and incorrect data. The quality of data remains the critical underlying factor for data-based systems to perform competently [9]. Errors in data can be costly and disruptive, leading to loss of revenue from incorrect transactions, to even irrevocable loss of reputation from misguided policy decisions. According to a recent study by IBM [4], poor-quality data costs the US \$3 trillion per year. Gartner Research also

reports that major organizations lose around \$15 million per year due to incidents traced back to poor data quality [7].

Even when the data collected from different sources is not noisy, linking and integrating data is challenging due to the heterogeneous representation across sources. Identifying groups of records that refer to the same entity (known as Entity resolution, record linkage or de-duplication) is one of the fundamental steps in integrating data. Additionally, arranging the identified entities in the form of a hierarchy based on hypernym relationships (often referred to as taxonomy construction) and designing summarization techniques help to improve their usability and organization. These different components of modern data management systems identify groups of similar/same records in the input dataset, which is formally studied as clustering.

Clustering refers to a broad class of unsupervised learning techniques that originated in the 1930s to study similarities between cultural tribes in America [75]. It is a ubiquitous problem that has been studied for many decades to mine patterns and perform analytics. It has found applications in various fields like biology, medicine, business, marketing, social science and many others. However, the advent of data deluge and explosion of noisy data sources has posed novel challenges that have not been explored in the literature. Some of these challenges are listed below.

1. Sensitivity to noise: Different data sources generate and store data under varied assumptions and naming conventions. The heterogeneous representation of records across sources and increase in noise has exposed the lack of robustness of prior data integration strategies.
2. Lack of ground truth: Clustering is generally an unsupervised task and often suffers from lack of ground truth data to evaluate the quality of designed techniques. There are a plethora of clustering techniques like k-center, k-means, k-median, agglomerative clustering that optimize an objective function hoping to capture ground truth clusters. For a given dataset, all these techniques gen-



Figure 1.1: Monument images collected from different sources with pairwise similarity calculated over text description generated by Google Vision API.

erate different outputs and it is unclear which clustering technique performs better than the rest.

3. Scalability: The increasing volume of data generated by millions of data sources has exacerbated the importance of designing scalable clustering techniques that cater to web scale workloads.

We now present an example describing some of these challenges.

Example 1. Consider a collection of nine images of different tourist destinations collected from various travel websites. Suppose the goal is to identify groups of images that refer to the same tourist destination. The ground truth clusters are $\{1, 6, 8, 9\}$ referring to the Eiffel Tower in Paris, $\{3, 4, 7\}$ referring to Las Vegas, $\{5\}$ denoting

the Leaning Tower of Pisa and $\{2\}$ corresponding to the Colosseum in Rome. We calculated pairwise similarity between images using the visual features generated by the Google Vision API. The pairs $(4, 6)$ and $(7, 8)$ exhibit a similarity of 0.85 while $(8, 9)$ has a similarity of 0.23. The record 9 has the least similarity with other records because the Google API identifies many noisy features in the 9th image like birds, human legs, heels, dancer, etc. The similarity values calculated using such automated techniques are noisy and traditional clustering techniques (like k-center clustering) that operate on these similarity values generate incorrect sets of clusters.

This dissertation is devoted to (a) develop a formalism to study the different facets of clustering, (b) devise robust and scalable clustering techniques, and (c) study practical data integration applications that benefit from these algorithms.

1.1 Facets of Clustering

Clustering is a challenging task that suffers from various challenges pertaining to noise, scalability and lack of ground truth. The dissertation is divided into two parts. The first part explores robust and scalable techniques to handle noise in datasets by leveraging supervision from humans (abstracted as an oracle). These techniques have been directly motivated by the ability of humans to understand the domain and improve the quality of automated decision-making techniques. The rise of crowdsourcing platforms, such as Mechanical Turk [185], Appen [182], and Prolific [166] have enabled the use of humans to process data on demand. We consider two different ways to leverage supervision to improve clustering quality. a) The first setting focuses on applications like entity resolution (ER), where the clusters refer to the different entities in the dataset. The clusters in Example 1 correspond to the output of ER. b) The second setting considers objective-based clustering techniques where the goal is to generate clusters that optimize for an objective function. For example, k-center clustering is one of the popular metric-based techniques that aims

to identify k clusters and corresponding cluster centers such that the maximum radius of the points from their respective centers is minimized. These techniques have been widely used for data summarization tasks.

The second part addresses the limitation of lack of ground truth data by studying generative models. Generative models make simple modelling assumptions to construct synthetic observational data which can be used to benchmark the quality of different algorithms. We propose a generative model for social network-based applications like community detection and develop robust techniques to recover ground truth clusters under this setting.

We now discuss these facets along with their real-world applications in more detail.

1.2 Entity Resolution with Supervision

Entity Resolution (ER, also known as de-duplication or record linkage) refers to the task of identifying clusters of records that refer to the same entity. Entity resolution is a fundamental problem in data management and has been studied since the seminal work of Fellegi and Sunter in 1969 [82]. The goal of ER is to identify and group different manifestations of the same real-world object, e.g., different ways of addressing the same person (names, email address, Facebook accounts), web pages with different descriptions of the same business, different photos of the same object, etc. We refer the reader to the survey by Getoor and Machanavajjhala [97] and the book by Dong and Srivastava [74] for more details. ER has evolved from the use of simple rules to complex deep-learning based classifiers. The challenge of performing ER using automated strategies is elusive to researchers and practitioners.

To identify the correct set of entities in a dataset, recent frameworks have leveraged supervision from crowd workers to improve clustering quality. The majority of these techniques assume access to a pairwise optimal cluster oracle (an abstraction of the crowd) that answers queries of type ‘Do u and v refer to the same entity?’. Many

companies like Google, Yahoo! and Groupon have explored the use of crowdworkers to help with entity resolution [124, 123, 34]. Given access to such an oracle that answers every question correctly, prior techniques [194, 197, 83] proposed prioritization strategies to order oracle queries such that clusters can be recovered with fewer queries.

1.2.1 Robustness

Even though the oracle provides additional knowledge as compared to automated techniques, an oracle can be error-prone as humans can make mistakes and some oracle queries are harder than the rest. For example, a crowdworker may label records 4 and 6 as the same tourist destination in Example 1 if she is not aware of Eiffel Tower’s replica in Las Vegas, USA. In such a setting, prior techniques that assume access to a noise free oracle can generate arbitrarily poor quality clusters. Chapter 3 discusses the effect of noise in oracle answers and presents an error correction toolkit that can be applied on top of prior techniques to guarantee high accuracy.

1.2.2 Scalability

ER pipelines rely on comparison of record pairs to identify groups referring to the same entities. For million-scale datasets, enumerating all pairs of records is not feasible. To improve the efficiency, record pairs that are highly likely to be non-matches (refer to different entities) are pruned. This procedure is called blocking and is often performed as a pre-processing step. For example, token-based standard blocking techniques do not compare any pair of records that do not share any token.

However, blocking techniques are known to be sensitive to input parameter settings. Stricter values of the blocking parameter improves overall efficiency, but it tends to prune out many record-pairs that refer to the same entity, leading to poor recall. On changing the parameter, the recall can improve at the cost of worsening its efficiency. These techniques are known to suffer from poor efficiency-effectiveness

trade-off, making it harder to scale to datasets containing millions of records. Chapter 4 presents a progressive blocking framework that fine-tunes blocking by using feedback from the partial ER output. This approach constructs new blocks and ranks them based on their quality to quickly capture matching record pairs. This methodology is shown to adapt to large-scale datasets with varied cluster size distribution and noise.

1.3 Objective-based Clustering with Supervision

With the advent of data deluge, it is crucial to develop techniques to summarize datasets for improved analytics. k-center clustering, one of the popular metric based clustering techniques, groups records such that most similar records are clustered together. These techniques are popularly used to generate dataset summaries. However, the quality of the generated clusters is dependent on the accurate estimation of distance between record pairs. Due to the presence of noise, automated techniques for distance computation yield sub-optimal clusters.

To circumvent these challenges, we consider supervision in the form of a comparison oracle that compares the relative distance between any two pairs of records. For example, given two pairs of records (u, v) and (x, y) the oracle returns whether (u, v) is closer to each other than (x, y) , i.e. $d(u, v) < d(x, y)$ or not, where $d(u, v)$ denotes the distance between u and v . Assuming access to a comparison oracle that may answer difficult queries incorrectly, Chapter 5 presents robust techniques that perform k-center clustering with provable theoretical guarantees. It further extends these methods to generate agglomerative hierarchical clusters. These algorithms are evaluated based on the summaries generated over various textual and image datasets.

1.4 Clustering with Generative Models

In many applications like community detection, it is infeasible to gather ground truth data to evaluate the effectiveness of a clustering algorithm. However, these datasets are accompanied by network information where nodes refer to records and edges capture the interaction between them. For example, consider a social network over individuals with the edges capturing friendship information and the goal is to identify communities of individuals based on their political inclination.

Community detection is one of the widely studied applications of clustering. It is particularly used to understand sociological behavior [104, 85], protein-protein interactions [52], gene expressions [65], recommendation systems [139], medical prognosis [178], DNA 3D folding [49], image segmentation [174], natural language processing [31], product-customer segmentation [64] and many more. Lack of ground truth data has been one of the consistent challenges to evaluate the quality of clustering techniques. Different techniques optimize different objectives and generate very different sets of clusters. Therefore, the key question whether the output of a clustering algorithm corresponds to the ground truth set of clusters is unsettled. To benchmark different clustering algorithms, generative models have been proposed to emulate real-world interaction between records. These models make assumptions about interaction between records which help to analyze different heuristics rigorously.

Stochastic Block Model (SBM) is one of the most popular generative models. It assumes that any pair of nodes is connected independently with a probability that depends on whether the considered pair of nodes belong to the same cluster or not. The independence assumption of the generative model does not capture the transitivity property of edge formation, which is commonly observed in social networks. For example, if u, v are friends and v, w are friends, then more likely than not u, w are friends. Chapter 6 presents a geometric block model that captures these properties and validates it over real-world datasets. To recover clusters from these

Table 1.1: Summary of contributions. Our work [91] is an extension of Chapter 6 and is not included in the thesis.

| Facet | Application | Chapter | Citation |
|-------------------|---|-----------|----------|
| Supervision | Entity Resolution: Robustness | Chapter 3 | [87] |
| | Entity Resolution: Scalability | Chapter 4 | [88, 89] |
| | Objective-based Clustering for Data Summarization | Chapter 5 | [16] |
| Generative Models | Generative Model for Community Detection | Chapter 6 | [92] |

datasets, we propose a simple triangle counting-based algorithm, which is proven to be order-optimal and evaluated on various real-world datasets.

1.5 Layout

Table 1.1 presents the papers that contributed to this thesis. The rest of the thesis is organized as follows.

- Chapter 2 discusses the background and related work on the different facets of clustering and their applications to data integration.
- Chapter 3 presents robust techniques to perform entity resolution in the presence of noise in supervision.
- Chapter 4 focuses on scalability of ER pipelines and proposes a new methodology to continuously feedback partial ER results into the blocking component, thereby improving its overall effectiveness and efficiency.
- Chapter 5 studies objective based clustering techniques using supervision in the form of comparison queries.
- Chapter 6 presents a generative model to capture interaction between records and proposes a simple triangle counting-based algorithm to recover the underlying clusters.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter presents the background that is relevant to all the chapters and discusses related work on clustering and relevant topics.

Clustering refers to the task of partitioning a dataset into subsets (or clusters) such that similar points share the same cluster and dissimilar points are separated into different clusters. The identified clusters generally have different interpretations for different applications. For example, a cluster may refer to a unique ground truth entity in entity resolution/de-duplication and it may refer to a group of individuals with similar interests in social network analysis. We now present a primer on relevant aspects of clustering and its applications to data integration.

2.1 Clustering with Supervision

As discussed in Chapter 1, identifying ground truth clusters with automated techniques is challenging due to presence of noise and ambiguous representation across different data sources. There has been a lot of interest in leveraging crowd workers to provide high-quality labels to simple multiple choice questions as supervision and use these answers to guide the clustering algorithm. This mechanism helps to improve the accuracy of clusters but at an additional cost due to human intervention. Additionally, humans can make mistakes while answering difficult questions and crowdsourcing is generally time-consuming as compared to automated techniques. To reduce the cognitive overload of crowd workers, there has been a lot of research on designing intuitive visualizations and identifying the right crowd worker for an input

query based on their expertise [57, 53]. This dissertation abstracts the crowdsourcing platform as an oracle that can employ these techniques at the backend to answer the input query. We now present a brief summary of the prior literature on these oracle abstractions and different forms of interaction with the oracle.

2.1.1 Crowdsourcing and Oracle Abstraction

Many crowdsourcing platforms like Amazon Mechanical Turk [185], Appen [182] and Prolific [166] have made it easier to ask simple questions to individuals. [199] presented methods to estimate the label of a query response by considering expertise of crowd workers along with their biases. [207] recently proposed a game-based crowdsourcing mechanism to accurately label rules. Due to the monetary cost of asking queries to a crowd worker, it is infeasible to ask millions of questions to an oracle. Therefore, active learning-based techniques are used to identify a small set of queries to train a classifier that acts as an oracle. Active learning is very effective in learning a high-quality classifier for balanced datasets [84, 167]. In case of skewed distribution of records, many weak supervision techniques have been proposed to quickly label a large set of data points and train a classifier [90, 168, 186]. Crowdsourcing is a vibrant area of research and there is a lot of interest in understanding the expertise of crowd workers to develop effective techniques to assign questions. We refer the reader to [53, 57] for a detailed survey of the recent techniques to implement the oracle.

We now describe the different types of queries that have been studied to leverage oracle expertise to perform clustering.

2.1.2 Types of Interactions

We categorize the interaction between the clustering algorithm and the crowdsourcing platform into two main categories [29]. First, result initiated interaction refers to the setting where an expert user provides certain input to the clustering algorithm after exploring the clustering output or the data points. Second, algorithm

initiated interaction allows the clustering algorithm to ask specific questions to the user, which are used to generate, verify or refine accurate clusters.

Result initiated interaction includes techniques where the oracle is expected to provide information about the desired output either by exploring certain data points or arbitrarily generating constraints [62, 27, 93]. Such mechanisms were initially proposed to capture domain knowledge from the expert who generates the initial set of clusters using an automated strategy and then fixes mistakes to improve the overall result. Even though such interactions have been well studied, these techniques rely on the ability of domain expert to identify mistakes. [152] partially addresses this challenge by providing hints to the user for improved selection of data points. Recently, many studies have also considered input constraints for the clustering algorithm that label certain pairs of records as must-link or cannot-link constraints [47, 30]. Additionally, [33, 66] have studied techniques to leverage feedback that is used to move incorrectly clustered points to their correct locations.

Algorithm initiated interaction considers all techniques where the algorithm identifies specific data points for which it needs supervision (often referred to as a query). This mechanism is motivated by active learning techniques for supervised learning, where the algorithm identifies samples for the crowd worker to label. Many types of queries have been studied for clustering. One of the most commonly studied oracle abstractions considers pairwise queries [144, 145, 194, 196, 83] where the oracle considers a pair of records as input and outputs if the two records belong to the same cluster or not. We formally define this oracle in Section 2.1.3. Other extensions of this interaction model query more than two input records and ask the oracle to generate a clustering [195, 193]. This interaction paradigm generally assumes that the query generated by the algorithm can be answered by the oracle without the context of other data points.

In this dissertation, we focus on algorithm initiated interactions and present two specific types of querying formats assumed for different types of clustering techniques.

2.1.3 Binary Oracle for Entity Resolution

A binary oracle considers queries of the form ‘do records u and v belong to the same optimal cluster?’. Such queries are easy to answer in settings where ground truth clusters refer to entities, often referred to as entity resolution. Entity Resolution was first proposed in the seminal work of Fellegi and Sunter in 1969 [82]. We formally define the entity resolution task and then provide a brief overview of the related work.

Problem 1. *Given a set of data sources \mathcal{S} , generating a collection of records $V = \{v_1, \dots, v_n\}$, identify a partitioning $\mathcal{C} = \{C_i : C_i \cap C_j = \phi, \forall i \neq j\}$ where $\cup_{C_i \in \mathcal{C}} C_i = V$ such that each partition $C_i \in \mathcal{C}$ refers to a distinct entity and no pair of partitions refer to same entity.*

Traditional ER architectures consist of three components (a) Blocking (b) Pair Matching and (c) Clustering.

Pair Matching and Clustering. Pair matching component of ER considers pairs of records and performs local decisions of whether they correspond to the same entity. The output of pair matching can have inconsistencies. For example, a pair matching algorithm may output that records 1 and 9 refer to the same entity, and records 1 and 6 refer to the same entity but records 6 and 9 refer to different entities. Clustering processes the output of pair matching to construct disjoint clusters of records, each referring to a distinct entity.

Pair Matching. Fellegi and Sunter [82] proposed classification-based approach to identify matching record pairs. The initial approaches that learned classifier required large training datasets which was later improved upon by active learning-based techniques [172]. In addition to classifier-based techniques, there have been studies on using rules [118, 80] and distance metrics [78] to speed up the resolution procedure.

One of the major drawbacks of these techniques has been the sensitivity to parameters. [74] discusses the different aspects of data integration and provides a detailed discussion about the general paradigm of entity resolution. In Example 1, error in similarity calculation shows that automated strategies can fail in many cases. Recently, the focus has shifted towards oracle-based techniques that leverage a human in the loop to provide labels to questions like ‘does u and v refer to the same entity?’ [194, 197, 83]. Oracle can be considered as an abstraction of the crowd platform or a classifier trained using active learning-based techniques [172]. Traditional strategies considered ER to be an offline task that needs to be completed before results can be used, which can be expensive in resolving billions of records. To address this concern, recent techniques propose to identify duplicate records early in the resolution process by leveraging prior similarities to consider a suitable prioritization of queries to the oracle [83]. Such online strategies have empirically shown to enable higher recall (i.e. more complete results) if terminated early or if there is limited resolution time available.

Clustering. In the context of ER, clustering is performed to handle inconsistencies of pair matcher. We refer the reader to [74] for a detailed discussion of clustering techniques for traditional ER architecture. Recent oracle-based techniques [194, 197, 83] leveraged transitivity to form clusters: when records u and v refer to the same entity and records v and w refer to the same entity, it can be inferred that u and w also refer to the same entity. This approach to construct transitive closure, assumed that the answers returned by the oracle are always correct. It can lead to poor resolution in the presence of oracle error. As an example, two different clusters (C_1 and C_2) can be incorrectly merged due to noise in a single query between nodes of C_1 and C_2 . Chapter 3 discusses the impact of noise on prior techniques and presents techniques to generate precise clusters.

Blocking. To scale entity resolution, blocking is traditionally performed as a pre-processing step [39]. Blocking identifies a small set of candidate record pairs that are considered for subsequent steps of matching and clustering. Intuitively, blocking considers different groupings of similar records into blocks and then selects record pairs from the blocks that are expected to be clean i.e. those with fewer non-matching pairs for further comparisons. One of the most common techniques to construct blocks is ‘Standard blocking’ [60], where each token in the dictionary corresponds to a block and any record that contains the token, belongs to the block. A blocking technique is considered effective if it can prune out the majority of the non-matches without losing any matching pair. [159] presents a comprehensive survey of various blocking techniques. We present a detailed discussion on blocking techniques in Chapter 4.

2.1.4 Comparison Oracle for Objective-based Clustering

In many applications, the relative distance between records is compared to generate clusters. For example, data summarization techniques identify a small representative subset of the data where each representative summarizes a group of similar records in the dataset. Popular metric based clustering algorithms such as k-center clustering and hierarchical clustering are often used for data summarization.

Definition 1 (k-center clustering). *Given a collection of n records $V = \{v_1, \dots, v_n\}$, identify k centers (say $S \subseteq V$) and a mapping of records to corresponding centers, $\pi : V \rightarrow S$, such that the maximum distance of any record from its center, i.e., $\max_{v_i \in V} d(v_i, \pi(v_i))$ is minimized.*

We now formalize a quadruplet comparison oracle that answers Yes/No queries about the relative distance between two pairs of records.

Definition 2 (Quadruplet comparison oracle). *An oracle is a function $\mathcal{O}_c : V \times V \times V \times V \rightarrow \{\text{Yes}, \text{No}\}$. Each oracle query considers two pairs of records as input and outputs $\mathcal{O}_c(v_1, v_2, v_3, v_4) = \text{Yes}$ if $d(v_1, v_2) \leq d(v_3, v_4)$ and **No** otherwise.*

Such oracle queries are easy to crowdsource as it does not need the crowdworker to calculate the distance accurately but needs them to compare them. Distance-based comparison oracles have been used to study a wide range of problems and we list a few of them – learning fairness metrics [122], top-down hierarchical clustering with a different objective [79, 51, 99], correlation clustering [181] and classification [180, 120], identify maximum [111, 189], top- k elements [129, 164, 63, 70, 133, 76], information retrieval [126], skyline computation [190]. However, there is no prior work that considers *quadruplet* comparison oracle queries to perform k -center clustering and single/complete linkage based hierarchical clustering.

2.2 Clustering with Generative Models for Community Detection

In applications like community detection and social network analysis, all records can be represented by a graph capturing interaction between them. There are a plethora of graph clustering algorithms that have been proposed to identify communities. For example, k -center, k -means, k -median, linkage based agglomerative algorithms are some of the popular clustering algorithms.

However, due to lack of ground truth data, it is impossible to benchmark the effectiveness of these techniques. To circumvent this challenge, generative models have been proposed to model interaction between nodes, which is then used to evaluate the effectiveness of a clustering algorithm. Stochastic block model (SBM) or the *planted-partition* model is one of the random graph models for community detection that generalizes the well-known Erdős-Renyi graphs [12, 13, 56, 71, 77, 115, 119, 150].

Definition 3 (Stochastic Block Model). *Consider a graph $G(V, E)$, where $V = C_1 \sqcup C_2 \sqcup \dots \sqcup C_k$ is a disjoint union of k clusters denoted by C_1, \dots, C_k . The edges of the graph are drawn randomly: there is an edge between $u \in C_i$ and $v \in C_j$ with probability $q_{i,j}$, $1 \leq i, j \leq k$.*

This model has been incredibly popular both in theoretical and practical domains of community detection, and the aforementioned references are just a small sample. One aspect that the SBM does not account for is a “transitivity rule” (“friends having common friends”) inherent to many social and other community structures. To be precise, consider any three vertices x, y and z . If x and y are friends (connected by an edge), and y and z are friends (connected by an edge), then it is more likely than not that x and z are also friends (connected by an edge). This phenomenon can be seen in many network structures - predominantly in social networks, blog-networks and advertising. SBM, primarily a generalization of Erdős-Renyi random graph, does not consider this characteristic, and in particular, the event that an edge exists between x and z is independent of the events that there exist edges between x and y , and y and z . We refer the reader to [11] for a detailed survey on stochastic block model and known algorithms for cluster recovery. Chapter 6 analyzes real-world social networks to propose a geometric block model, which is motivated by random geometric graphs [162].

2.3 Notation

Table 2.1 summarizes the notation used in the thesis.

Table 2.1: Table of Notation.

| Symbol | Definition |
|--|---|
| $V = \{v_1, \dots, v_n\}$ | Set of records |
| $H = (V, A, p_m)$ | Graph with initial matching probabilities. $p_m : A \rightarrow [0, 1]$ is a partial function returning the probability that u and v are matching, for all $(u, v) \in E$. |
| $\mathcal{C}^* = (V, E^+) = \{C_1^*, \dots, C_k^*\}$ | Real-world entities, in non-increasing order of size. $C^*(u) \in \mathcal{C}^*$ denotes the entity of u in \mathcal{C}^* . |
| $Q = Q_+ \cup Q_-$ | Oracle responses: Q_+ (resp. Q_-) contains Yes (resp. No) answers. |
| $Q[S]$ | Subgraph of Q induced by nodes in $S \subseteq V$ (oracle responses about S). |
| $\mathcal{C} = (V, E'^+) = \{C_1, \dots, C_l\}$ | Inferred version of \mathcal{C} by the oracle strategy in non-increasing order of size. $C(u) \in \mathcal{C}$ denotes the entity of u in \mathcal{C} . |
| $p_e(u, v)$ | Error probability $p_e : Q \rightarrow [0, 0.5]$ of a specific answer, conditioned on the answer value (YES or NO). |
| B | Block: A subset of records, $B \subseteq V$ |
| $P = (V, A)$ | Blocking graph, $A \subset V \times V$ |
| ϕ | Feedback frequency |
| $p(B)$ | Probability score of a block B |
| $u(B)$ | Uniformity score of block B |
| $H(B)$ | Entropy of block B |
| \mathcal{H} | Block Hierarchy |
| G_t | Random Geometric graph |
| γ | Fraction of nodes used for scoring blocks |
| μ_g | Expected similarity of a matching edge |
| μ_r | Expected similarity of a non-matching edge |

PART I: CLUSTERING WITH SUPERVISION

CHAPTER 3

ENTITY RESOLUTION WITH SUPERVISION: ROBUSTNESS TO NOISE

In this chapter, we study the impact of noise in binary oracle answers for settings where ground truth clusters refer to entities and develop an error correction toolkit that guarantees high quality with minimum effort. The chapter focuses on scenarios where data is collected from various sources leading to a skewed cluster size distribution. Section 3.1 discusses a high-level motivation of the problem and Section 3.2 formalizes the notion of a noisy oracle and defines the problem statement. In Section 3.3, we describe a random graph-based error correction layer and prove its theoretical guarantees. Section 3.4 builds ER pipelines by leveraging the error correction layer with different parameter settings. Section 3.5 presents the empirical evaluation of the techniques described in the chapter and Section 3.6 presents the related work.

3.1 Introduction

Entity resolution (ER) seeks to identify clusters of records that refer to the same underlying real-world entity. As discussed in Chapter 2, ER is an intricate problem that can leverage humans to match pairs of records based on domain knowledge, but would be challenging for automated strategies. For these reasons, many frameworks have been developed to leverage humans (abstracted as an oracle) for performing entity resolution tasks [194, 196, 83]. These studies introduced the notion of a binary *oracle* that *correctly* answers questions of the form “do records u and v refer to

the same entity?” and proposed techniques to leverage machine-generated pairwise matching probabilities to reduce oracle queries. However, certain questions can be difficult to answer correctly even for humans. Prior techniques can generate arbitrarily poor quality clusters in the presence of noise. To improve the robustness, we propose a cost-effective approach that can be added as an extra-layer to any oracle-based strategy, helping to preserve their performance guarantees while maintaining high precision. The error correction layer can be tuned (or even turned off) trading off query budget for accuracy, thereby providing flexibility to adapt to different ER applications.

Example 1. *In Figure 3.1 we show nine pictures along with textual descriptions taken from web pages of different trip planning websites. The records 1, 6, 8 and 9 refer to Eiffel Tower in Paris, 3, 4, 7 refer to the replica of Eiffel Tower at Las Vegas, 2 is Coliseum in Rome and 5 is the Leaning Tower of Pisa. We used Google Vision API to generate textual features of all the pictures and used the generated features to estimate record pair similarity values. Pairs of pictures that got matching similarity more than 0.5 are connected by a green edge. Pairs of pictures that got scores below or equal than 0.5 are connected by a red edge. Some edges are removed to ensure clarity. Table 3.1 shows the title, description and source corresponding to each image.*

Table 3.1: Text data associated with pictures in Figure 3.1.

| | Entity | Name | Source |
|---|--------|---|----------------------------|
| 1 | EF | Eiffel Tower Recommended Sightseeing Time: 1-3 hours | viator.com |
| 2 | CO | Colosseum | trip.com |
| 3 | LV | Eiffel Tower Viewing Deck at Paris Las Vegas | tripadvisor.in |
| 4 | LV | Eiffel Tower Experience and Dinner at Paris Las Vegas | viator.com |
| 5 | LP | Leaning Tower of Pisa | trip.com |
| 6 | EF | Eiffel Tower: explore the top | tou Eiffel.paris |
| 7 | LV | Eiffel Tower Viewing Deck | trip.com |
| 8 | EF | Champagne bar at the Eiffel Tower viewing deck | restaurants-tou Eiffel.com |
| 9 | EF | Eiffel Tower Tours and Activities, skip the line | trip.com |

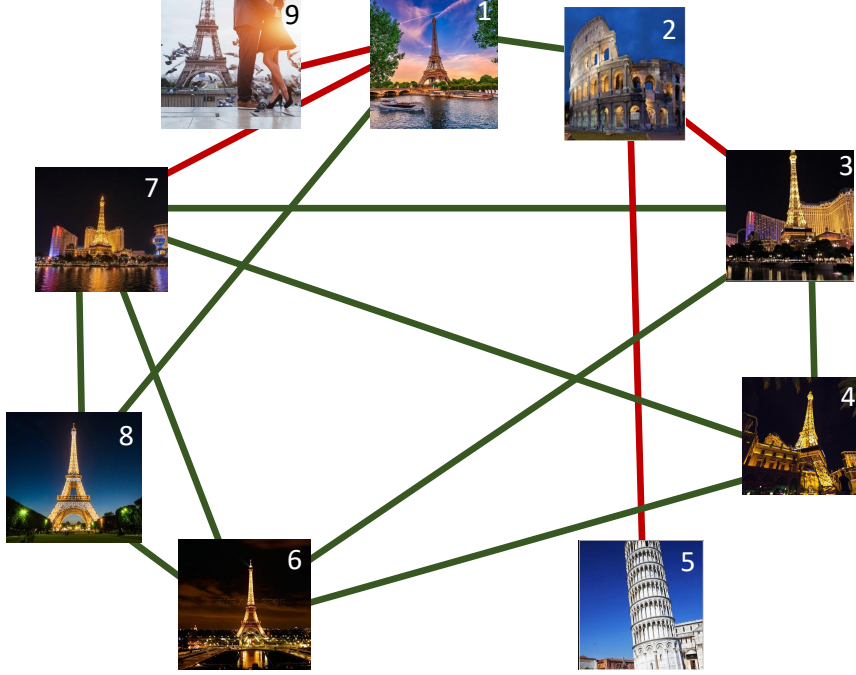


Figure 3.1: Example pictures from different travel websites. Edges represent matching probabilities (edges are colored green if probability is more than 0.50 and red otherwise).

For this chapter, we think of records as nodes in a graph (as shown in Figure 3.1) where edge weight captures the probability that its endpoints refer to same entity. This graph corresponds to the output of the blocking phase.

3.2 Preliminaries

Consider a collection of records V and a graph $\mathcal{C}^* = (V, E^+)$, where E^+ is a subset of $V \times V$ such that $(u, v) \in E^+$ represents that u and v refer to the same entity. \mathcal{C}^* is transitively closed, that is, it partitions V into cliques representing distinct entities. We call the nodes in each clique a *cluster* of V , and we refer to the clustering \mathcal{C}^* as the *ground truth* for the ER problem. We refer to the cluster containing a given node u , as $C^*(u) \in \mathcal{C}^*$.

Consider a graph $H = (V, A, p_m)$, $A \subseteq V \times V$, with pairwise machine-generated matching probabilities $p_m : A \rightarrow [0, 1]$ over a collection of n records $V = \{v_1, \dots, v_n\}$.

Noisy Oracle abstraction. Consider a black box which can answer questions of the form “do u and v represent the same entity?”. Edges between any pair of records can be either asked the black box or inferred with the help of previous answers. If the black box is guaranteed to answer all queries correctly, we can reconstruct \mathcal{C}^* exactly with a reasonable number of queries [83, 194, 197]. However, in most applications of crowdsourcing and supervised learning, some answers can be erroneous and we can only build a noisy version of \mathcal{C}^* , which we refer to as \mathcal{C} . $C(u)$ refers to the cluster in \mathcal{C} that contains the node u . For the sake of simplicity, we refer to this realistic oracle as “noisy oracle”.

Definition 1. *A noisy pairwise oracle for \mathcal{C}^* is a function $\mathcal{O}_b : V \times V \rightarrow \{\text{Yes}, \text{No}\} \times [0, 0.5]$. If $\mathcal{O}_b(u, v) = (a, e)$ with $a \in \{\text{Yes}, \text{No}\}$ and $e \in [0, 0.5]$, then $\Pr[(u, v) \in E^+] = 1 - e$ if $a = \text{Yes}$, and e otherwise.*

For instance, if $\mathcal{O}_b(u, v) = (\text{Yes}, 0.15)$, then $(u, v) \in E^+$ with probability 0.85 and if $\mathcal{O}_b(u, v) = (\text{No}, 0.21)$, then probability of $(u, v) \in E^+$ is 0.21. We refer to the probability of a specific answer for the pair (u, v) being erroneous, conditioned on the answer being Yes or No, as its *error probability* $p_e(u, v)$. Let $Q = Q_+ \cup Q_-$ be a graph containing all the edges that have been queried at a given moment, along with the noisy oracle answers, we state $p_e : Q \rightarrow [0, 0.5]$. In the ideal case, when $p_e = 0$ for any pair (u, v) , the noisy oracle reduces to the perfect oracle [83, 194, 197]. An ER strategy s takes as input matching probability graph H and grows a clustering \mathcal{C} by asking edges as queries to the noisy oracle. We call *inference* the process of building a clustering \mathcal{C} from Q . \mathcal{C} initially consists of singleton clusters: s can either merge existing clusters into larger clusters or split an already established cluster. Note that the subgraph of Q_- induced by $C(u)$ (that is, $Q_- \cap C(u)$) can be non-empty, because of wrong answers. We refer to such a subgraph as $Q_-[C(u)]$.

Evaluation Metrics. We evaluate the quality of clusters generated by a strategy by comparing F-measure of the identified pairs of records that refer to the same entity. Let \mathcal{C}_t be the clustering returned by the strategy, after $t = |Q|$ questions. \mathcal{C}_t can be considered as a transitively closed set of edges $E_t'^+$. Recall and precision of the identified clusters after t questions are defined as follows.

$$\text{recall}(t) = \frac{|E_t'^+ \cap E^+|}{|E^+|}, \text{precision}(t) = \frac{|E_t'^+ \cap E^+|}{|E_t'^+|}, \text{ and}$$

$$\text{fmeasure}(t) = 2 \frac{\text{recall}(t) \cdot \text{precision}(t)}{\text{recall}(t) + \text{precision}(t)}.$$

Classic F-measure cannot distinguish whether a strategy achieves high F-measure only at a given value of t or earlier in the ER process. Therefore, similar to [83], we use a *progressive F-measure* function, denoting the *area* under the **fmeasure**-questions curve. We omit the index t from \mathcal{C}_t and $E_t'^+$, when we do not refer to any particular value of t . Now, we are ready to define our problem.

Problem 2 (Noisy Oracle Strategy). *Given a set of records V , a noisy oracle access to \mathcal{C}^* and a matching probability function p_m (possibly defined on a subset of $V \times V$), find the strategy that maximizes the progressive F-measure of the generated clusters \mathcal{C} .*

Matching probabilities. There are many ways of estimating the matching probability function p_m . For instance, similarity calculation techniques can provide pairwise similarities which can be mapped to matching probabilities, as in Section 3.1 of [201].

Error probabilities. There are many ways of accessing error probabilities. For instance, the crowd platform could return a confidence score associated with each answer. Another option is to learn a function mapping similarity scores to error probabilities, akin to matching probabilities [201]. See discussion in Section 3.5.

3.3 Random Graph Toolkit

If the oracle answers every question correctly, constructing a spanning forest over the set of matching edges is sufficient to recover the ground truth set of clusters. Any

other matching edge is inferred by using transitivity over the spanning forest. A single mistake in labelling can affect the precision/recall of the constructed clusters. For example, two different clusters C_1 and C_2 could be incorrectly merged if any of the edges between C_1 and C_2 is labelled as matching. The effect of error can be reduced if we query more than one edge between them and merge only if the majority of those edges are matching.

Analogously, the precision of the queried graph can be improved by strengthening the min-cuts of the graph formed on the queried edges. Therefore, we exploit the concept of *expander graphs* [21], which are sparse graphs with strong connectivity properties. Connectivity strength can be controlled with a unique parameter tuned by the user (which we refer to as β). When the user increases β , the precision of the clustering also increases, at a small price. When the user decreases β , performance becomes similar to the oracle strategy alone, trading off budget for quality of solution. Eventually, $\beta = 0$ is equivalent to using the oracle strategy alone. The difference between spanning forests and a random graph with expansion property is illustrated in Figure 3.2.

Definition 2 (γ -expansion and weighted γ -expansion). *A graph $G = (V, E)$ satisfies γ -expansion if for every subset of nodes $V' \subseteq V$, $|V'| \leq |V|/2$, the total number of edges crossing the cut $(V', V \setminus V')$ is at least $\gamma|V'|$. Analogously, if edges can have weights, G satisfies weighted γ -expansion if the total weight of edges crossing the cut $(V', V \setminus V')$ is at least $\gamma|V'|$.*

Our toolkit has the invariant that every cluster $C \in \mathcal{C}$ – more precisely, every subgraph $Q_+[C]$ of positive answers¹ – has good expansion properties. Specifically, every time an oracle strategy asks a query involving two nodes – say (x, y) – we ask other queries incident to the corresponding clusters $C(x)$ and $C(y)$, such as (x_1, y_1)

¹Recall that we use the notation $Q_-[C(u)]$ for referring to the subgraph Q_- induced by $C(u)$. Similarly, $Q_+[C]$ is the subgraph of Q_+ induced by a cluster C .

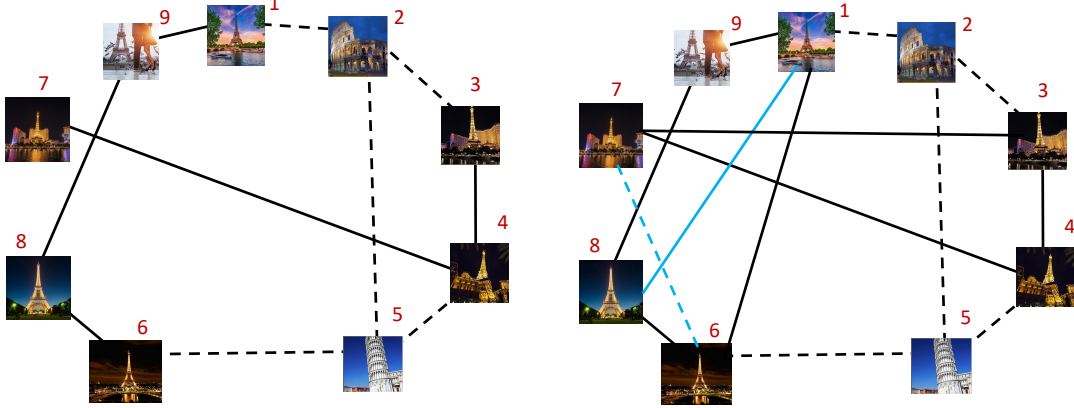


Figure 3.2: Comparison of tree-based clusters in the perfect oracle setting (left) and robust clusters with a noisy oracle (right). Positive (negative) answers are represented with solid (dashed) edges. Wrong answers are shown in blue.

and (x_2, y_2) , with $x_1, x_2 \in C(x)$ and $y_1, y_2 \in C(y)$. Note that extra queries are *not* replicas of (x, y) , and they are all distinct. If the answers are positive then the *degree* of nodes within $C(x) \cup C(y)$ increases, and if the degree gets high enough then we merge $C(x)$ and $C(y)$ together. We know that indeed random *regular* graphs have good expansion properties [21], and hence choosing the queries randomly from $C(x) \times C(y)$ helps us to maintain the required expansion. Note that, since every answer comes with a different error probability (see Section 3.2), we are interested in the joint error probability of each cut, rather than its cardinality. Under independent error in answers, that is when answers to different queries are not correlated, this is equivalent to considering *weighted* degree regularity, where weight is given by the function $w(x, y) = -\log p_e(x, y)$, where $(x, y) \in Q_+$.

Expansion properties of clusters translate into two desiderata:

- **Robustness:** since the joint error probability of each cut is small, all the subsets of nodes are likely referring to the same entity.
- **Cost-effectiveness:** the total number of queries is small, as edge density of the graph is small.

Algorithm 1 `query_multiple_edges($C(x), C(y), \beta$)` method for deciding when two clusters (or any two sets of nodes) are matching.

```

1:  $S \leftarrow C(x) \times C(y)$ 
2:  $p \leftarrow 1 \cdot \prod_{(u,v) \in S \cap Q_+} p_e(u, v)$  ▷ joint error probability of answers
3: for  $(u, v)$  in shuffle( $S \setminus Q$ ) do ▷ random queries
4:   if  $p \leq 1/(e(|C(x)| + |C(y)|))^{\beta(\min(|C(x)|, |C(y)|))}$  then
5:     return true
6:   end if
7:   if  $P[C(x) \text{ non matching } C(y)|Q] \geq 0.95$  then
8:     return false
9:   end if
10:  if  $\mathcal{O}_b(u, v)$  then
11:     $p \leftarrow p \cdot (p_e(u, v))$ 
12:  end if
13: end for
14: if  $|S \cap Q_+| = |S|$  then return true else return false

```

Example 2. In Figure 3.2, we show an example queried graph having expansion properties ($\gamma = 1$) for data in Figure 3.1, compared with spanning trees. Both connected components of the left figure (i.e., trees in the spanning forest) and graphs of right figure yield the same, correct, clustering $\{1, 6, 8, 9\}, \{3, 4, 7\}, \{2\}, \{5\}$. While connected components only work in the absence of errors, graphs with expansion property produce the correct clustering also in the presence of plausible human errors such as $(6, 7)$ (false positives) and $(1, 8)$ (false negative).

The rest of this section is organized as follows. We first describe methods to construct graphs with good expansion (Section 3.3.1). We refer to it as **random expansion** as indeed we are building a random graph inside each cluster that is known to have good expansion property. Then we describe how to plug such methods in previous perfect oracle strategies, while achieving a target level of robustness (Section 3.3.1).

3.3.1 Algorithms

The random expansion toolkit consists of Algorithms 1 and 2. The input includes the noisy oracle answers Q , the matching probability function p_m , and the error probabilities p_e , as in definition of Problem 2. In addition, the input includes (i) the

pairs of nodes (x, y) that the oracle strategy is planning to ask and (ii) two parameters β and τ . The toolkit also uses the following shared data and methods.

- The method `query_edge(u, v)` $\equiv \mathcal{O}_b(u, v)$, which returns a $\{\text{Yes}, \text{No}\}$ noisy oracle answer for (u, v) , and updates Q accordingly.
- The partition \mathcal{C} , which can be updated via union, split and find operations. In our notation, $C(u) \in \mathcal{C}$ is the cluster of u .

Parameters. The algorithms 1 and 2 use two parameters, β and τ .

- β trades-off queries for precision. Smaller values of β correspond to sparser clusters, and therefore to fewer queries. Greater values of β correspond to denser clusters and to higher precision.
- τ is used for optimization purposes. Specifically, it is used for limiting the absolute number of queries where needed.

Methods. The random expansion toolkit consists of two subroutines.

- Algorithm 1 is meant to be called in place of the perfect oracle with the purpose of growing clusters with good expansion properties. Given a query (x, y) selected by an oracle strategy (represented with the two corresponding clusters $C(x)$ and $C(y)$), Algorithm 1 provides an intermediate layer between the ER logic and the `query_edge(u, v)` method: instead of asking the selected query (x, y) as the oracle strategy would do, Algorithm 1 selects a bunch of random queries between $C(x)$ and $C(y)$, and returns a **Yes** answer only if the joint error probability of the cut is small.
- Algorithm 2 provides functionalities for growing edge cuts that were not considered during the execution of the ER process, and adjusting spurious weak cuts. Running Algorithm 2 at a later phase of the ER process can fix premature decisions.

Algorithm 1. Our method `query_multiple_edges()` is used for comparing two sets of nodes $C(x), C(y)$ and determining if the two clusters $C(x)$ and $C(y)$ refer to the

same entity or not. Upon a positive outcome, the two clusters (or the cluster and the singleton node) should be merged into one. The method asks random “inter-cluster” queries (lines 3–13), and it has positive outcome (line 5) when the joint error probability of all the previously collected and the new positive answers between $C(x)$ and $C(y)$ become smaller than a threshold (line 4). The early termination option (lines 7–9) prevents from iterating through all the inter-cluster queries in case of negative outcomes.

The threshold for positive outcome (line 4) depends on the size of $C(x)$ and $C(y)$, because of both practical and technical reasons. Intuitively: (i) in large clusters we can ask more independent queries than small clusters, and aim for smaller error; (ii) errors in large clusters have a bigger impact on F-score than errors in small clusters². We use a reasonable constant at line 7 for avoiding unnecessary complexity. Indeed (i) variations in the range $(0.9, 1.0)$ do not have significant impact on progressive F-score in our experiments because most error probability values we work with are at least 0.1, and (ii) changes at line 7 do not affect our theory (later in this section).

Algorithm 2. This method helps to correct the mistakes in the clusters by either merging smaller clusters or by removing erroneous nodes from the clusters. It tries to maintain the regularity of subgraphs by either adding new edges or splitting the clusters in regular subgraphs. The major advantage of this subroutine is that it can be applied for any collection of clusters consisting of few queried edges and it returns various regular subgraphs which would represent true clusters with high probability. The algorithm works in two phases. In the *merging* phase, it examines all the cluster pairs and whenever `query_multiple_edges()` returns **Yes**, it merges them (line 5). The `query_multiple_edges()` method here ignores the past set of oracle answers Q

²Theoretical guarantees may not extend to small sized clusters. However, in practice, we observed that even in tiny clusters – with 2 or 3 nodes each – the error probability is small, thanks to high-quality answers with low p_e . This happens especially in datasets where the negative-to-positive error is negligible or zero.

Algorithm 2 `boost_fscore`(τ, β) method.

```
1:  $(C_i, C_j) \leftarrow \text{next\_edge\_cbn}()$  ▷ Merge phase
2:  $numc \leftarrow 0$ 
3: while next\_edge\_cbn()  $\vee$   $numc \leq \binom{\tau}{2}$  do
4:   if query\_multiple\_edges( $C_i, C_j, \beta$ ) then
5:      $C_i.\text{union}(C_j)$ 
6:   end if
7:    $numc \leftarrow numc + 1$ 
8:    $(C_i, C_j) \leftarrow \text{next\_edge\_cbn}()$ 
9: end while
10: for  $i \in [1, |\mathcal{C}|]$  do ▷ Split phase
11:   for  $u \leftarrow \text{next\_node\_cp}(\mathcal{C}[i])$  do
12:     if  $\neg \text{query\_multiple\_edges}(C(u) \setminus \{u\}, \{u\}, \beta)$  then
13:        $C(u).\text{split}(\{u\})$ 
14:     end if
15:   end for
16: end for
```

that were queried in other phases of the algorithm, for probability estimations. This helps to get an unbiased, uncorrelated judgment of merging the two clusters. Note that cluster pairs are processed in non-increasing order of *cluster benefit*, defined as $\text{cbn}(C_i, C_j) = \sum_{u,v \in C_i \times C_j} p_m(u, v)$ (line 3). The merging phase terminates after at most $\binom{\tau}{2}$ iterations. By changing τ (default $\log n$) two extreme behaviors are possible:

- if $\tau = n$ then we try to merge all the cluster pairs grown so far;
- if $\tau = 0$, we skip the merge phase and go straight to the split.

In the *splitting* phase, it examines all the pairs of nodes and clusters, and whenever `query_multiple_edges()` returns No, it pulls out the node of its cluster (line 12). We give a detailed description of the auxiliary methods used by Algorithm 2 in the following.

- `next_edge_cbn()` iterates over the cluster pairs in \mathcal{C} and returns the next cluster pair, in non-increasing order of benefit (Formally, it starts with $\arg \max_{C_i, C_j \in \mathcal{C}} \text{cbn}(C_i, C_j)$). It has a corresponding “`has_next`” method returning `true` or `false`.
- `next_node_cp`(C) iterates over the nodes in the cluster C and returns the next node u in non-increasing order of product of error probabilities of queried match-

ing edges incident on u i.e. $\prod_{v \in C, (u,v) \in Q_+} p_e(u, v)$. Note that this method can return the same node multiple times till it gets a **Yes** in line 12 of `boost_fscore()`.

The advantage of `boost_fscore()` is twofold. (i) Quality of clustering is improved by correcting early false-positive and negative errors (improving precision and recall). (ii) It comes at a small cost because `query_multiple_edges()` asks new queries only if needed, i.e., if none of the two conditions (lines 4 and 7) is already met.

Analysis. By making use of our toolkit, a perfect oracle strategy can grow clusters with good expansion properties, rather than just spanning forests. Expansion properties of clusters guarantee that the precision of the solution \mathcal{C} is close to 1 with high probability, and low density guarantees that the number of queries is small. Formally, we prove the following theorems, which show that the probability that any cut of a given cluster $C \in \mathcal{C}$ is “wrong” (that is, all the positive answers crossing the cut are wrong) decreases exponentially with γ , provided C satisfies γ -expansion.

Theorem 1. *Let $G = Q_+[C] = (C, E)$, $C \in \mathcal{C}$. The expected number of wrong cuts is smaller or equal than $\frac{2}{(e|C|)^{\beta-1}}$, if C satisfies weighted γ -expansion with $\gamma \geq \beta \log(e|C|)$ and $\beta \geq 1$.*

Proof. For a given cut, we calculate the probability that all the **Yes** answers across the cut are wrong and use it to estimate the expected number of such wrong cuts. It can be seen that the probability of a cut $(A, C \setminus A)$ being wrong is $\prod_{u \in A, v \in C \setminus A, (u,v) \in Q_+} p_e(u, v)$. The expected number of wrong cuts is bounded by the following:

$$E[\text{\#Wrong Cuts}] = \sum_{A \subseteq C, 1 \leq |A| \leq |C|/2} \prod_{u \in A, v \in C \setminus A, (u,v) \in Q_+} p_e(u, v)$$

Since C satisfies weighted γ -expansion, sum of edge weights i.e.

$$\sum_{u \in A, v \in C \setminus A, (u,v) \in Q_+} -\log(p_e(u, v)) \geq \gamma|A|, \text{ where } \gamma = \beta \log(e|C|). \text{ Hence,}$$

$$\begin{aligned}
E[\# \text{Wrong Cuts}] &\leq \sum_{A \subseteq C, |A| \leq |C|/2} \left(\frac{1}{e|C|} \right)^{\beta|A|} = \sum_{r=1}^{|C|/2} \binom{|C|}{r} \left(\frac{1}{e|C|} \right)^{\beta r} \\
&\leq \sum_{r=1}^{|C|/2} \left(\frac{|C|e}{r} \right)^r \left(\frac{1}{e|C|} \right)^{\beta r} \leq (|C|e)^{1-\beta} \sum_{r=1}^{|C|/2} 1/r^2 \\
&\leq \frac{\pi^2}{6(e|C|)^{\beta-1}}
\end{aligned}$$

Hence,

$$E[\# \text{Wrong Cuts}] = \sum_{r=1}^{|C|/2} \binom{|C|}{r} p_e^{\gamma r} < \frac{2}{(e|C|)^{\beta-1}}$$

□

We now show that the probability of a wrong cluster (that is, clusters with at least a wrong cut) is negligible, for certain values of β . Here n is the total number of entities.

Theorem 2. *If C satisfies weighted γ -expansion $\forall C \in \mathcal{C}$, where $\gamma \geq 3 \log(en)$, then the probability that there exists a cluster in \mathcal{C} with a wrong cut is at most $\frac{1}{n}$.*

Proof. With $\gamma \geq 3 \log(en)$ and following the proof of Theorem 1, the expected number of erroneously inferred cuts in a single cluster is atmost $\frac{1}{en^2}$ (denoted by p_{er}). Using union bound, the total number of erroneously inferred cuts in the set of clusters $\mathcal{C} = \{C_1, \dots, C_l\}$ is atmost $\sum_{C_i \in \mathcal{C}} p_{er}$. So the expected number of incorrect partitions in the set of clusters is calculated as :

$$\sum_{C \in \mathcal{C}} p_{er} \leq np_{er} \leq \frac{1}{en}$$

□

We finally show that a given cluster has high expected precision.

Theorem 3. *Let $G = Q_+[C] = (C, E)$, $C \in \mathcal{C}$. The expected precision of C is at least $1 - O\left(\frac{1}{n^\beta}\right)$, if C satisfies weighted γ -expansion with $\gamma \geq \beta \log(en)$ and $\beta \geq 1$.*

Proof. Referring to Theorem 1, the probability of getting a wrong cut, $A, C \setminus A$ is evaluated as $\prod_{u \in A, v \in C \setminus A, (u,v) \in Q_+} p_e(u, v) \leq \left(\frac{1}{en}\right)^{\beta|A|}$. Given a wrong cut $A, C \setminus A$, the maximum number of edges, incorrectly labeled as positive is $|A| * |C \setminus A|$. So, the expected number of false positives (fp) within the cluster C can be calculated as follows

$$\begin{aligned}
fp &= \sum_{r=1}^{|C|/2} r(|C| - r) \binom{|C|}{r} \left(\frac{1}{en}\right)^{\beta r} \\
&= |C|(|C| - 1) \left(\frac{1}{en}\right)^\beta + |C|(|C| - 1) \sum_{r=2}^{|C|/2} \binom{|C|-2}{r-1} \left(\frac{1}{en}\right)^{\beta r} \\
&\leq |C|(|C| - 1) \left(\frac{1}{en}\right)^\beta + |C|(|C| - 1) \sum_{r=2}^{|C|/2} \left(\frac{(|C|-2)e}{r-1}\right)^{r-1} \left(\frac{1}{en}\right)^{\beta r} \\
&< 3|C|(|C| - 1) \left(\frac{1}{en}\right)^\beta
\end{aligned}$$

The total number of edges in C labeled as positive is $|C|(|C| - 1)/2$. Hence, the precision $\geq 1 - O\left(\frac{1}{n^\beta}\right)$ \square

Corollary 1. *If $\beta = 1$, the expected precision of $C \in \mathcal{C}$ is $1 - O\left(\frac{1}{n}\right)$.*

Intuitively, we estimate the probability of an incorrect cut in the queried graph to figure out the false positive error and union bound over all cuts to get the worst-case error bound. The independence assumption of each error helps us easily estimate the error of incorrectly merging two different clusters (or the error of an incorrect cut). Using this worst-case bound, we achieve a bound on the number of queries required for nearly accurate clustering.

Answer error. Our analysis provides performance guarantees under the assumption of independent errors in noisy oracle answers, i.e., when answers to different queries

are not correlated. We are aware that some specific records (e.g. number 9 in Figure 3.1) can be more difficult to match. In Section 3.5, we provide experimental results for both datasets with independent error (we refer to these as *synthetic* error datasets), and datasets with a significant amount of correlated error, due to real crowd answers. Results of the former type confirm our theoretical analysis, and results of the latter demonstrate that the toolkit is robust to correlated error in practice.

Application to perfect oracle strategies. Given a perfect oracle strategy, `query_multiple_edges()` can be used as a substitute of plain connectivity for deciding if two clusters refer to the same entity or not. In case of *node ordering* based strategies, such as [194], a singleton node is added to a cluster by querying a single edge with it. We can replace that querying step with `query_multiple_edges()` which will try to check if the singleton cluster containing the node refers to the same entity as the one it was supposed to be queried with. The *edge ordering* based strategies, such as [197], query the edges in decreasing order of probabilities to decide if the two endpoint clusters ought to be merged or not. Similarly, we can make the same decision by replacing this querying step with `query_multiple_edges()` of the two endpoint clusters. We note that the strategy in [83] combines node and edge ordering, and can be modified to apply random expansion similarly. Along with the incorporation of `query_multiple_edges()`, any strategy can call the `boost_fscore()` procedure towards the end for correcting the errors made in earlier stages. This shows that our toolkit is independent, self-contained and easy to use.

3.4 Pipelines and Trade offs

In the previous section, we have described a simple way to merge the random graph toolkit with the perfect oracle strategies to minimize the effect of error in the noisy setting. One of the main advantages of the above described methods is that they maintain high precision throughout the resolution phase. This high precision is

Algorithm 3 `lazy(τ, θ, β)` algorithm.

```
1:  $\mathcal{C} \leftarrow \{\{u\} : u \in V\}$ 
2: node( $\tau, \theta$ )
3: edge(0.0,  $\beta$ )
4: boost_fscore( $\tau, \beta$ )
```

Algorithm 4 `node(τ, θ)` algorithm.

```
1:  $P \leftarrow \text{max\_node\_ecs}()$ 
2: while  $\frac{|Q+|}{|Q|} \geq \theta \vee |Q| \leq 5$  do
3:    $v \leftarrow \text{max\_node\_ecs}()$ 
4:    $(i, C, b) \leftarrow \text{next\_cluster\_cbn}(v, P)$   $\triangleright b = \text{cbn}(v, C)$ 
5:   while  $i \leq \tau \wedge \text{cbn}(v, C) > \theta$  do  $\triangleright \theta$  is a lower-bound for benefit
6:      $u \leftarrow C.\text{pick\_random}()$ 
7:     if  $\mathcal{O}_b(v, u)$  then  $\triangleright$  possibly wrong answer
8:        $c.\text{union}(\{v\})$ 
9:       break
10:    end if
11:     $(i, C, b) \leftarrow \text{next\_cluster\_cbn}(v, P)$ 
12:  end while
13:   $P \leftarrow P \cup \{v\}$ 
14: end while
```

achieved at the cost of lower progressive F-score. A close look at the expansion toolkit shows that we can possibly improve this shortcoming as well. We now describe three different pipelines which are devised by building on the ideas of expansion toolkit and Hybrid [83], the state-of-the-art resolution algorithm for perfect oracle setting.

3.4.1 Progressive F-score: Lazy Pipeline

Our basic pipeline (called `lazy()`) is focused towards optimizing the progressive F-score at the cost of lower precision in the start. It does so by following a mix of perfect oracle strategies `node()` from [194] and `edge()` from [197] as in [83] in the beginning to avoid asking extra queries as required by the error correction toolkit. However, at

Algorithm 5 `edge(θ, β)` algorithm.

```
1:  $(u, v, p) \leftarrow \text{next\_edge\_amp}()$ 
2: while  $p > \theta$  do  $\triangleright \theta$  is a lower-bound for avg. matching prob.
3:   if  $\mathcal{O}_b(u, v)$  then
4:      $C(u).\text{union}(C(v))$ 
5:   end if
6:    $(u, v, p) \leftarrow \text{next\_edge\_amp}()$ 
7: end while
```

the end, it uses our expansion toolkit through `boost_fscore()` to correct errors, and achieve higher final F-score as well. It runs in three phases (see Algorithm 3). The first phase uses `node()` which has great performance in practice for resolving dataset with few large clusters, but has much worse performance than [197] if there are many small clusters. Therefore, we end the first phase when large clusters (if any) have been resolved, possibly leaving some clusters ungrown. The second phase examines inter-cluster edges as in [197] and possibly ask them to the oracle. If the answer is positive, we merge the clusters, aiming at growing all the clusters left ungrown in the first phase. In the second phase we prioritize cluster pairs with high matching probabilities (irrespective of their size). Since, the first two phases only maintain a forest, they do not provide any guarantees when answers can be erroneous, but give the best performances in the perfect oracle setting for different cluster size distributions. At the third phase, `lazy()` uses `boost_fscore()` for error correction—this is where it uses the expansion property.

Parameter setting. The τ and θ parameters control the “duration” of `node()` and `edge()`. The parameters used for the toolkit are β , and τ . We set default values for θ and τ to 0.3 and $\log n$ respectively (as in [83]), and for β to 1 (as in Corollary 1).

Auxiliary methods. `lazy()` pipeline makes use of the following auxiliary methods which are useful to understand the pseudocode provided in Algorithms 3, 4 and 5.

- `next_cluster_cbn(v, P)` iterates over the clusters in \mathcal{C} and returns the next cluster C in P , in non-increasing order of cluster benefit $\text{cbn}(\{v\}, C)$. It also returns the benefit and a count (from the cluster with maximum benefit to v , which defaults to 0).
- `max_node_ecs()` returns the next node v , in non-increasing order of expected cluster size $\text{ecs}(v)$ i.e. $\sum_{u,v \in V} p_m(u, v)$.
- `next_edge_amp()` iterates over the cluster pairs in \mathcal{C} and returns an arbitrary inter-cluster edge of the next cluster pair, in non-increasing order of average

matching probability. Formally, it starts with $\arg \max_{C_i, C_j \in \mathcal{C}} \text{avg}_{u \in C_i, v \in C_j} p_m(u, v)$.

Together with the edge, it returns the average probability.

Node phase. The `node()` method in Algorithm 4 processes nodes in non-increasing order of maximum cluster benefit. It keeps a list of nodes that are processed (lines 1, 13). The list is initialized to `max_node_ecs()` (line 1) because expanding the largest cluster first enables high recall early on. The next node processed is most likely to belong to the largest cluster (line 4), until such cluster has been fully grown. Then, the next node processed will seed the second-largest cluster, and so on. Every processed node is compared to clusters in P (lines 4–12) until either a (possibly wrong) answer is positive (line 7), or the benefit drops under a threshold θ (line 5). In the first case, the node is added to the cluster. We limit at most τ comparisons for each node, which defaults to $\log n$, before a positive answer is obtained. However, if $\mathcal{O}_b(u, v)$ mainly collects negative answers, then the benefit will drop and the loop is terminated. For the sake of simplicity, we use the same threshold θ for benefit and the fraction of positive answers (line 2). Also, [83] proposes to use the notion of window w to provide a handle over the use of high benefit nodes. Since it doesn’t provide much gain practically [83], we choose $w = 1$ for simplicity. We can incorporate the same by modifying line 3 in Algorithm 4 accordingly.

Edge phase. Recall of `node()` algorithm can be smaller than 1 for two reasons: (1) positive-to-negative errors in pairwise queries, and (2) positive questions “deferred” for small benefit. The `edge()` method in Algorithm 5 compares cluster pairs, in non-increasing order of average matching probability (lines 1, 6). Every comparison is done by querying a random intra-cluster edge. If the answer is positive (line 3), then the two clusters are merged. `edge()` terminates whenever the matching probability drops under θ (line 2).

Discussion. The `lazy()` pipeline can be useful in applications with reasonably few erroneous answers. In our experiments we show that, in such scenarios `lazy()` can

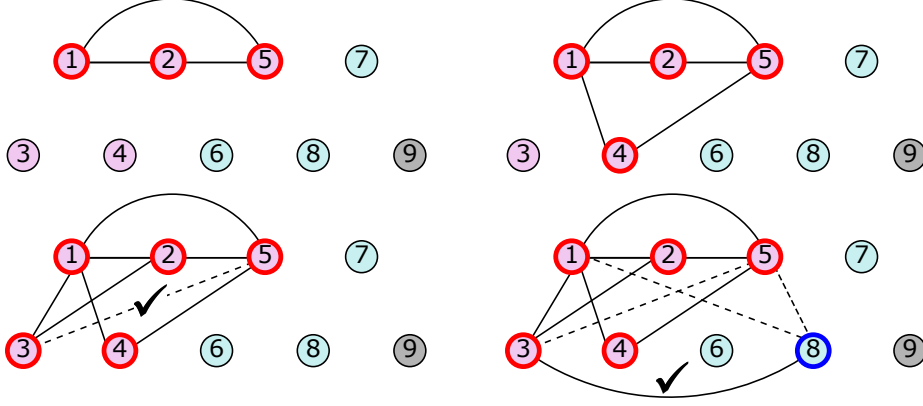


Figure 3.3: Sample execution of `node()`. Positive (negative) answers are represented with solid (dashed) edges. Node fill and stroke colors are representative of true and computed clusters, respectively. Thick edges highlight corrected errors.

provide high F-score at the end of `boost_fscore()` computation. Nevertheless, errors made in the first two phases can strongly affect progressive F-score.

Example 3 (Figure 3.3). Assume that nodes in Figure 3.3 are processed by `node()` (Algorithm 4) in the following order 1, 2, 5, 4, 3, 8, 7, 6, 9, and that every edge has the same error probability. Assume also that `query_multiple_edges()` is set for returning **Yes** only upon two positive answers between the input clusters $C(x)$ and $C(y)$. Note that the β parameter is not meant for controlling the number of positive answers required for merging clusters, but rather for ensuring that nodes in every cluster C have the right weighted degree with respect to the cluster sizes for achieving good expansion properties. For size 5 cluster, this corresponds approximately to $\beta = 0.5$. Finally, assume that the noisy oracle answers **Yes/No** as in Figure 3.1 and they can correctly classify 4. Initially, 1, 2, 5 get three positive answers and a size 3 cluster C is created. When node 4 is processed, `node()` asks at least 2 questions for adding 4 to C , whereas in the perfect oracle strategy only 1 would be sufficient. If both are positive, then 4 is added to C and the next node 3 is pulled from the queue and compared with C . Suppose that 3 gets 1 positive and 1 negative answer, that are (3, 2) and (3, 5) respectively. A new query between 3 and C (for instance (3, 1)) is asked, and if the

Algorithm 6 $\text{eager}(\tau, \theta, \beta)$ algorithm. Default values for parameters are the same as Algorithm 3.

```

1:  $\mathcal{C} \leftarrow \{\{u\} : u \in V\}$ 
2:  $\text{node\_exp}(\tau, \theta)$ 
3:  $\text{edge\_exp}(0.0, \beta)$ 
4:  $\text{boost\_fscore}(\tau, \beta)$ 

```

answer is positive, then 3 is added to C , correcting the false negative answer for $(3, 5)$. Similarly, after the first cluster $C = 1, 2, 3, 4, 5$ is formed, $\text{node}()$ carefully compares the next node 8 with C with at least 2 queries, for instance $(8, 3)$ and $(8, 1)$. Even though we get a wrong answer for $(8, 3)$, we can still make a correct decision by asking $(8, 5)$. This resulted in the creation of a new cluster with the green records.

Example 4. In the same setting, assume that the first 5 edges in Figure 3.2 are processed by $\text{edge}()$ (Algorithm 5) in the following order $(1, 2)$, $(2, 5)$, $(5, 1)$, $(3, 4)$, $(3, 1)$. The first 4 edges get positive answers and two clusters $C_1 = \{1, 2, 5\}$ and $C_2 = \{3, 4\}$ are created. When edge $(3, 1)$ is processed, $\text{edge}()$ asks 3 questions for merging C_1 and C_2 , namely $(3, 1)$ itself, and for instance $(4, 2)$ and $(4, 5)$. Nodes that queries $(4, 2)$ and $(4, 5)$ would never be asked in the perfect oracle setting, and that $(4, 5)$ may be never asked by $\text{node}()$. Note also that $\text{node}()$ tries to grow one cluster at a time, while $\text{edge}()$ grows multiple sub-cluster in parallel and then merges them. This is also true for their corresponding perfect oracle versions in [194, 197] and for their progressive variation in [83].

3.4.2 Final F-score: Eager Pipeline

The pipeline $\text{eager}()$ in Algorithm 6 maintains high precision at the cost of low progressive F-score which is orthogonal to $\text{lazy}()$. The method $\text{node_exp}()$ is a variant of Algorithm 4, where the basic $\mathcal{O}_b(u, v)$ method is replaced by the expansion method $\text{query_multiple_edges}(C(u), C(v), \beta)$ (line 7) for comparing clusters to sin-

Algorithm 7 `adaptive`(τ, θ, β) algorithm. Default values for parameters are the same as Algorithm 6.

```

1:  $\mathcal{C} \leftarrow \{\{u\} : u \in V\}$ 
2: node_exp_adp( $\tau, \theta$ )
3: boost_fscore( $\tau, \beta$ )
4: edge_exp_adp(0.0,  $\beta$ )
5: boost_fscore( $\tau, \beta$ )

```

gleton nodes. The method `edge_exp`() is an analogous variant of Algorithm 5³. The method `boost_fscore`() is called at the end (line 4). The parameter β controls how strong is the intra-cluster connectivity, and trades off precision for cost. High values of β correspond to high precision and cost. Small values yield low precision; specifically, $\beta = 0$ asks a single positive question between a node and a cluster, yielding a result similar to the `lazy`() pipeline. We observed in our experiments that $\beta = 1$ achieves the best progressive F-measure, and we set this as our default value.

Discussion. The product of false positive probabilities for every node in every cluster of `eager`() is below a threshold $\beta \log(e|C|)$. This corresponds to random expansion with positive edges $(u, v) \in Q_+$ having weight of $-\log p_e(u, v)$. For the special case of constant error probability p^E , every node v of a cluster ends up with the same number of incident positive answers $\beta \log_{1/p^E}(e|C(v)|)$. In practice, maintaining degree regularity throughout the execution can be overkill and may result in a slow growth of the F-measure.

3.4.3 Adaptive Pipeline

The pipeline `adaptive`() in Algorithm 7 achieves the best of `eager`() and `lazy`(). It provides the same final F-score of `eager`() earlier in the querying procedure, along with the high progressive F-measure of `lazy`(). The intuition is to switch between a single \mathcal{O}_b query and `query_multiple_edges`() depending on the current answer. We compare clusters with \mathcal{O}_b as in `lazy`(), but we use our robust comparison tool

³that is, replacing \mathcal{O}_b with `query_multiple_edges`()

Algorithm 8 Adaptive `edge_exp_adp`(θ, β) algorithm.

```
1:  $(u, v, p) \leftarrow \text{next\_edge\_amp}()$ 
2: while  $p > \theta$  do
3:    $q \leftarrow \mathcal{O}_b(u, v)$ 
4:   if  $q \wedge (p < 0.5) \vee \neg q \wedge (p \geq 0.5)$  then
5:     if query_multiple_edges( $C(u), C(v), \beta$ ) then
6:        $C(u).union(C(v))$ 
7:     end if
8:   else
9:     if  $q$  then
10:       $C(u).union(C(v))$ 
11:    end if
12:  end if
13:   $(u, v, p) \leftarrow \text{next\_edge\_amp}()$ 
14: end while
```

`query_multiple_edges()` if the result is in “disagreement” with matching probabilities. Formally: (i) a positive answer in case of low average matching probability (< 0.5); (ii) a negative answer in case of high average matching probability (≥ 0.5). We call respectively `node_exp_adp()` and `edge_exp_adp()` the adaptive versions of `node_exp()` and `edge_exp()`. The method `edge_exp_adp()` is shown in Algorithm 8 (the pseudo-code of the method `node_exp_adp()` is analogous). Finally, we add an extra execution of `boost_fscore()` (line 3), with respect to `eager()`, for correcting early errors due to the adaptive nature of `node_exp_adp()`.

Discussion. The clustering maintained by `adaptive()` temporarily allows clusters with low expansion in the node phase, if the answer from the oracle is in agreement with matching probabilities, irrespective of error probability of the answer. As a consequence, some *cuts* of a cluster can have small weight, thus temporarily violating the expansion property (see Definition 2). We observe in practice that such violations in the early phase of `adaptive()` can provide high gains in progressive F-score without losing precision.

3.4.4 Application Scenarios.

There are different scenarios for our toolkit strategies, depending on matching and error probabilities (that is, depending on how accurate machine-based methods and human workers can be on the specific application).

- (HC) The error rate of answers is high and matching probabilities are correlated with the ground truth, that is, truly positive edges have high probability and truly negative edges have low probability. In this scenario, we expect `eager()` to perform better than `lazy()`, and `adaptive()` to perform like `eager()`.
- (LC) The error is low and matching probabilities are correlated with the ground truth. In this scenario, we expect `lazy()` to be better than `eager()`, and `adaptive()` to be like `lazy()`.
- (LU) The error is low and matching probabilities are uncorrelated with the ground truth. In this scenario, we still expect `lazy()` to be better, but we expect `adaptive()` to be like `eager()`.
- (HU) The error is high and matching probabilities are uncorrelated with the ground truth. Here, we expect `eager()` to be better than `lazy()`, and `adaptive()` to be like `eager()`.

Finally, there can be mixed cases (MIX) of reasonable error rate and matching probability noise. We expect the different strategies to have similar progressive F-score in such cases. In Section 3.5 we provide experiments covering all these cases, except HU.

3.5 Empirical Evaluation

Datasets. Some datasets have real attribute values, and come with a cache of answers from the AMT crowd. We refer to such datasets as “Real values/Real oracle” (RR).

Similarly, we refer to real-world datasets with synthetic noisy oracle answers, as “Real values/Synthetic oracle” (RS). Other datasets have synthetic attribute values and synthetic noisy oracle answers (“Synthetic values/Synthetic oracle”, SS). We have access to the gold standard of each dataset except for `dblp`. We use the same method as [161, 83] for computing a labeling of `dblp` to be used in place of the gold standard for the purposes of comparing oracle strategies. We refer to the computed labeling as “silver” standard. Properties of the datasets are given in Table 3.2.

- `cora`: Title, author, venue, and date of scientific papers.
- `skew`: Simulated hospital patients data, including name, phone and address, produced using Febrl [59].
- `sqrtn`: Same as `skew`, with different cluster sizes.
- `captcha`: CAPTCHA images, each showing a four-digit number. The number of records per entity follows a power-law distribution with an exponent of -2.5.
- `gym`: Images of gymnastics athletes, where it is very difficult to distinguish the face of the athlete, e.g. when the athlete is upside-down on the uneven bars.
- `allsports`: Images of athletes from ten different sports such as Tennis, Soccer, Gymnastics. The pairs of images across sports are easy to distinguish but the images within the same category of sport are quite difficult due to various angles of the body, face and uniform.
- `dblp`: All the computer science articles in the homonym bibliographic index, up to August, 13th 2015.

Matching probabilities. Matching probabilities of text datasets (`cora`, `skew`, `sqrtn` and `dblp`) are computed using string similarity, such as Jaro [205] and Q-grams. Specifically, similarity scores are mapped to probabilities using buckets as in Section

Table 3.2: Number of nodes n (i.e., records), number of clusters k (i.e., entities), size of the largest cluster $|C_1|$, reference to the paper where they appeared first, and origin (real or synthetic). The scenario column matches the dataset with scenarios of Section 3.4.3. We provide experiments for both low and high error rate for **cora** and **dblp**, thus we match them with both LC and HC.

| dataset | n | k | $ C_1 $ | $ E^+ $ | ref. | origin | scenario |
|------------------|------|------|---------|---------|-------|--------|----------|
| cora | 1.9K | 191 | 236 | 62.9K | [146] | RS | LC, HC |
| skew | 900 | 93 | 50 | 8.2K | [83] | SS | LC |
| sqrtn | 900 | 30 | 30 | 13.1K | [83] | SS | LC |
| captcha | 244 | 69 | 8 | 386 | [191] | RR | MIX |
| gym | 94 | 12 | 15 | 449 | [191] | RR | MIX |
| allsports | 200 | 64 | 5 | 227 | [192] | RR | LU |
| dblp | 3.1M | 3.0M | 159 | 299.7K | [83] | RS | LC, HC |

3.1 of [201]. Record pairs are evenly divided into buckets of equal size according to their similarity. Probability for each bucket is computed using the gold standard (for **cora**, **skew** and **sqrtn**) and the silver standard (for **dblp**). Computing all pairwise matching probabilities for **dblp** is prohibitively expensive, therefore obviously non-matching pairs have been removed by a machine-based preprocessing technique (i.e., *blocking*). Matching probabilities for remaining pairs are randomly drawn from overlapping exponential distributions: $e^{-\lambda x}$ for negative edges and $1 - e^{-\lambda x}$ for positives ($\lambda = 10$). Matching probabilities for **captcha** and **gym** are the same in [191].

Error probabilities. We consider two different models of error in the oracle answers. In our first model, the oracle error is constant irrespective of the pair being queried. This constant-error model has been widely used in the entity resolution literature [191, 110]. In our second model, edges may have different probability of errors. It is reasonable to believe that if workers are not malicious, then false positives and negatives are more common in high-similarity and low-similarity pairs, respectively. We validate this observation by our experiments on multiple real datasets, e.g., **gym** and **captchas**. We construct 10 equi-width buckets of matching probabilities and measure the oracle error for edges in each bucket treating error rate in each bucket to be constant. This model is similar to a situation where the oracle asks to a fixed

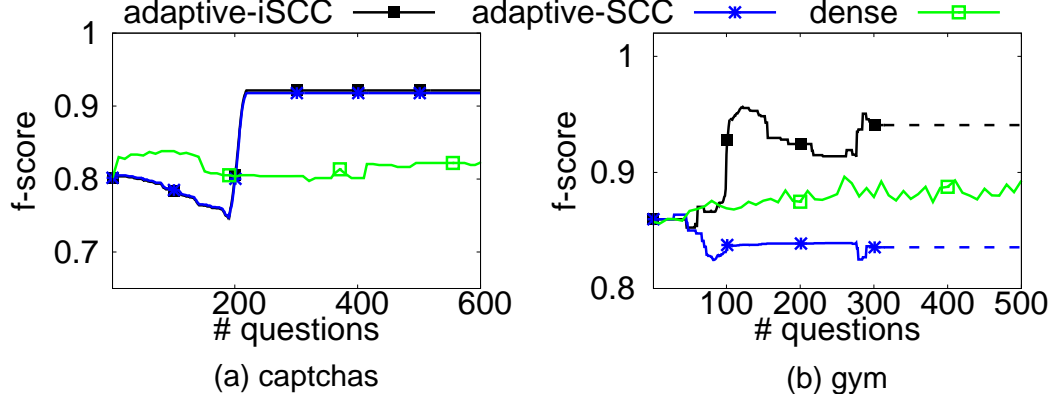


Figure 3.4: Comparison of SCC-based `adaptive()` variations and the original implementation on `dense()`.

number of crowd workers. In such a scenario, each answers' accuracy is different as it depends upon the question's difficulty and the expertise of the worker.

For the synthetic oracle, we generate erroneous answers by flipping the correct answer with given error probabilities p^+ , p^- .

3.5.1 Comparison with Previous Strategies

In this section, we evaluate our pipelines along with previously proposed algorithms for noisy setting [110], [191], [192], dubbed `votes()`, `dense()`, and `waldo()`. We plot F-score vs number of queries, as progressive F-score can be quantified as the area under this curve. We refer to the ideal curve as in [83], as `ideal()`.

For a fair comparison, we consider the following settings.

1. `dense()` implementation in [191] returns, at each step, the result of a clustering algorithm over both high matching probabilities and oracle answers. Therefore, the F-score is always in a high range even in absence of answers. This is different from our algorithms (and also from `votes()`) that only consider query answers for clustering, yielding a gradual growth of the F-score.
2. both `dense()` and `votes()` can ask multiple times the same query. However, our algorithms ask each query once.

3. `waldo()` employs a “batch” cost model for multi-wise queries.

Clustering. To compare our approach with the original implementation of `dense()` in [191], we implement two variants of `adaptive()` that use the same clustering algorithm as `dense()` – the so-called “Spectral Connected Components” (SCC) algorithm – dubbed `adaptive-SCC()` and `adaptive-iSCC()`. The two variants differ in whether SCC computes clusters from scratch (i.e. only based on collected oracle answers) or from already established clusters by the original `adaptive()` implementation. The latter variant `adaptive-iSCC()` allows only for clusters that are either same or contain the clusters inferred by `adaptive()` (the “i” stands for “incremental”) and thus (i) its recall is lower-bounded by the original `adaptive()` recall, and (ii) its F-score converges to the original `adaptive()` F-score. Figure 3.4 compares the above variants of `adaptive()` algorithm with the original `dense()` implementation in [191]. The plots show that `adaptive-iSCC()` gets the best progressive F-score. Initially, when the number of questions is 0, the clustering returned is purely SCC over input matching probabilities. The initial F-score of `dense()` is higher than both variants of `adaptive()`, especially for `captchas` for which machine-generated matching probabilities are less noisy than `gym`. As new queries are made, the clustering produced by the two variants of `adaptive()` become better, achieving higher final and progressive F-score. In the remainder of the chapter, we use a variant of `dense()` that does not use SCC. In this way it starts from 0 F-score like `votes()`, `waldo()` and the original `adaptive()` algorithm, making the comparison fair.

Multiple queries. In the table below, we show the distribution of queries for `votes()`. Such a strategy can ask the same question up to 8 times for our RR datasets with the goal of aggregating answers of single crowd workers as part of the ER problem [110]. `dense()` rarely asks more than 5 times the same query in all our datasets.

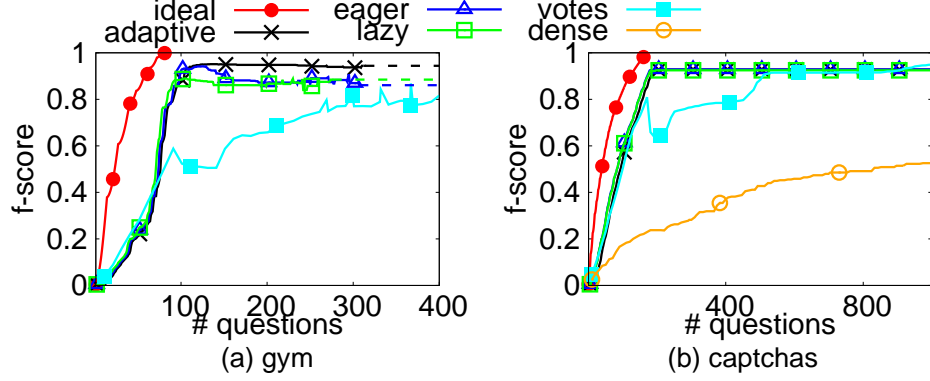


Figure 3.5: Comparison of strategies over RR datasets.

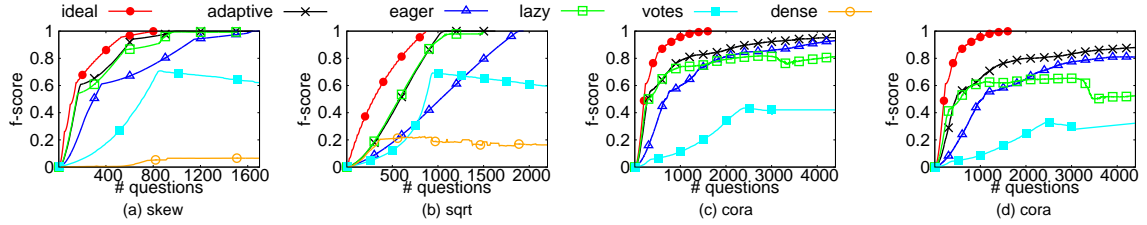


Figure 3.6: Comparison of strategies over RS and SS datasets, with constant error probabilities (a-c have $p^+ = p^- = 0.1$ and d has $p^+ = p^- = 0.2$), both for generation of answers and for input to our algorithms.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---------|----|----|------|----|-----|----|----|---|
| captcha | 99 | 57 | 3330 | 39 | 65 | 39 | 38 | 0 |
| gym | 13 | 22 | 39 | 76 | 106 | 46 | 40 | 3 |

In the following, we count the number of distinct queries if not specified otherwise (see our cost model in Section 3.2).

Real error. In Figure 3.5, we compare the progressive F-score of our and previous strategies over RR and RS datasets. We set the x axis to the number of *distinct* queries in all the datasets. The plots show that our `eager()` and `adaptive()` pipelines achieve more than 90% F-score with less than 400 queries in case of `gym`. Similarly for `captchas`, the F-score achieved is more than 90% even with the simplest `lazy()` pipeline. This is due to the zero false positives in the oracle answers for `captchas`. In such a scenario, `lazy()` pipeline has 100% precision. Benefits of using our pipelines

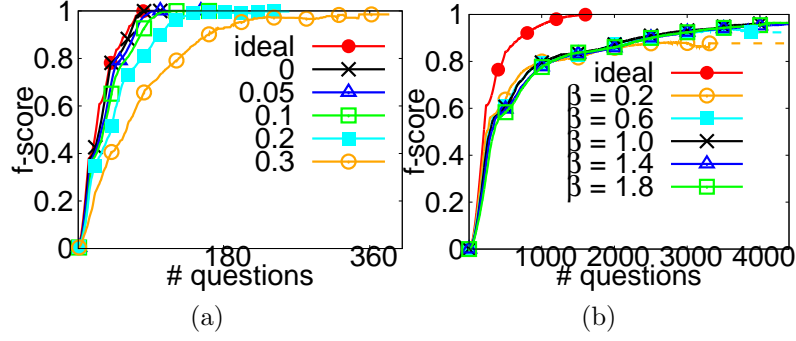


Figure 3.7: (a) F-score of `adaptive()` for different error rates in crowd answers (`gym`). (b) Different settings of β for `adaptive()` over `cora` dataset with $p^+ = p^- = 0.1$.

become evident with more challenging datasets, such as `gym`. The results of Figure 3.5 confirms our theory for `eager()`, and shows the effectiveness of `adaptive()` in practice. In order to investigate further the effect of oracle answers on these datasets, Figure 3.7a shows the `adaptive()` pipeline over the `gym` dataset with *perfect* matching probability function. That is, we set $p_m(u, v) = 1$ iff $(u, v) \in E^+$, and 0 otherwise. For this experiment, we generate synthetic erroneous answers and plot the results of `adaptive()` as the error rate for both false positives and negatives varies in the range $[0, 0.3]$. In this setting, performance of `adaptive()` is almost ideal till error rate is less than 0.2.

Tuning of β . Figure 3.7b demonstrates the effect of changing the tuning parameter β on the performance of `adaptive()` (multiple queries are counted separately). It is evident that the progressive F-score improves as the β value is reduced. The downside of this is that it tries to grow a sparser graph, which has lower precision and the algorithm plateau's out at a lower final F-score. On the other hand, higher values of β make the approach conservative to ask more queries and reduces the progressive F-score. The above described behavior is consistent with our theoretical bounds proven in Section 3.3. Note that β has no explicit control on the *number* of positive answers asked by `adaptive()` because the actual amount of positive answers depends on the

specific error probabilities (see discussion on Example 3). Nonetheless, varying β has predictable effects.

- If the error probability for positive answers is a constant p_- , the number of positive answers per node is bounded by $O(\beta \log_{1/p_-} n)$.
- If $\beta = 0$, the threshold at line 4 of Algorithm 1 is 1, thus the number of positive answers per node is bounded by 1, yielding overall $n - |C|$ positive queries, akin to perfect oracle strategies [194].
- For a given dataset, with arbitrary error probabilities, increasing β yields a non-decreasing amount of positive queries.

Synthetic error. In Figure 3.6 we compare the progressive F-score of our and previous strategies with constant error probabilities over SS datasets. We set the x axis to the total number of queries. The plots show that all our pipelines achieve more than 90% F-score in most cases, even with constant error probability. One exception is `cora` with error rate $p^+ = p^- = 0.2$, where final F-score is slightly above 85%. It is interesting to observe that the `adaptive()` pipeline is quite close to the `ideal()` algorithm even for the noisy oracle setting. Also, the overall overhead to reach perfect F-score is only 200 for `sqrt` and `skew` datasets. Expansion based approach provides smaller benefits when the clusters being resolved are small. This is evident from the benefit of using our toolkit in Figures 3.7.

Waldo. `waldo()` makes a collection of pairwise and multi-item queries (k-node query), in order to optimize the total cost of resolution of clusters. The major benefit of asking a multi-item query ($k=6$ in `waldo()`) is that the cost of a query is much less than $\binom{k}{2}$. Hence, we adopt the cost model from [192] to evaluate the expense of resolution of records for different F-score values. We consider the cost of a pairwise query to be 2 cents and a multi-item query to be 5 cents (as opposed to $\binom{6}{2} = 15$). Given the sensitivity of error estimates for multi-item queries with respect to datasets and crowd workers, we use datasets (`allSports` and `cora`) from the `waldo()` [192]

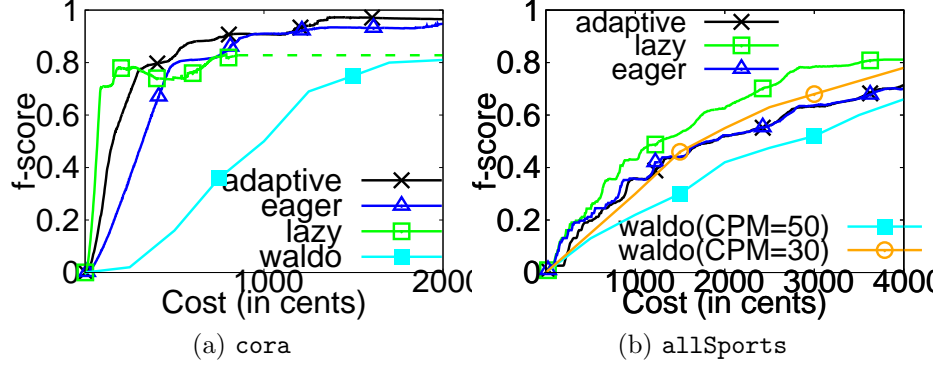


Figure 3.8: Comparison of `waldo()` and `adaptive()`

paper. In `allSports` we experiment with the same CPM settings in the `waldo()` paper [192]; CPM describes the confusing pairs fraction with higher error rate. (We refer the interested reader to [192] for more details about CPM.) Figure 3.8 shows that for `cora`, `adaptive()` is at least three times cheaper than `waldo()` to achieve 0.8 F-score. In the case of `allSports`, the `waldo()` approach is comparable with `adaptive()`. Our default `adaptive()` setting is slightly worse than `waldo()` 30 CPM setting and consistently better than `waldo()` 50 CPM setting. Among our strategies, `lazy()` is much better than both `waldo()` and `adaptive()`. This is consistent with the LU application scenarios in Section 3.4.3. We recall that `allSports` is a RR dataset and thus we use real crowd answers and corresponding error probabilities for expansion.

Running time. `adaptive()` provides a better query complexity as compared to `dense()`, `votes()` and `waldo()`. This gain in query complexity reflects directly in the time taken to resolve the clusters by each of the techniques. Additionally, for `gym` the average time taken to issue one query by `adaptive()` is roughly 2.67 milliseconds for the first 100 queries. This further reduces to 1.45 as more clusters are resolved. For `votes()`, the time taken is roughly 8.89 milliseconds for the first 100 queries which further increases with increase in queries because it tends to ask the same queries more frequently which are not counted as new queries by our query model. On

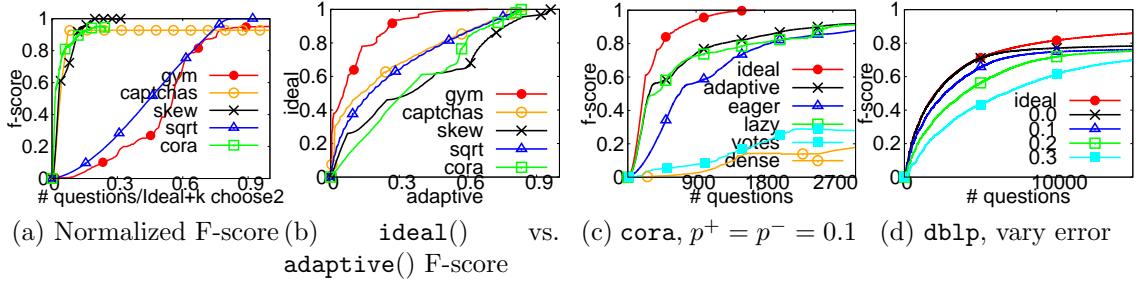


Figure 3.9: (a-b) `adaptive()` strategy on different datasets. (c-d) Comparison of our strategies over our bibliographic dataset.

the other hand, `dense()` algorithm takes 64.9 milliseconds roughly to issue a query, irrespective of it being already queried to the oracle or not. This clearly shows that `adaptive()` is at least 3 times faster than `votes()` and roughly 24 times faster than `dense()` algorithm. We observe similar behavior for other datasets too. `adaptive()` not only provides better query complexity, but the time taken to issue a single query is also less than `votes()` and `dense()` algorithms.

Comparison of datasets. In Figures 3.9a and 3.9b, we compare the progressive F-score of `adaptive()` over different datasets. In Figure 3.9a, the x -axis is the fraction of the minimum number of queries for complete resolution, that is $n - |C| + \binom{k}{2}$ (see theoretical analysis in [194]). Please note that the final fraction at the end of the ER process can be higher than 1. Larger area means that `adaptive()` has a comparatively higher progressive F-score on the given dataset. This does not mean that `adaptive()` is closer to `ideal()` on the given dataset. Figure 3.9b shows the fraction of `ideal()` F-score that is reached by `adaptive()` at different points in the ER process (number of queries is expressed as a fraction of total `adaptive()` queries). Within this framework, best cases for `adaptive()` are the two RR datasets, dubbed, `gym` and `captchas`. Nevertheless, `gym` has comparatively smaller progressive F-score than all other datasets, as it constitutes a “hard” case for `ideal()` in the first place (in the sense that the best possible progressive F-score is comparatively low) due to

uniform cluster size distribution (akin to `sqrt`).

Alternative forms of expansion. Although random expansion is analytically proven to require less *queries* than deterministic expansion [21], empirically one can make use of alternative forms of expansions for selecting “control queries”. The table below compares the number of queries asked by `adaptive()` on `cora` with different expansion strategies, dubbed *high*, *low*, and *uncertain*, selecting queries with the closest matching probability to 1, 0, and 0.5 respectively. Random expansion gets to the same F-score of deterministic alternatives with less or comparable number of queries.

| F-score | p^+, p^- | random | high p_m | low p_m | uncertain p_m |
|---------|------------|--------|------------|-----------|-----------------|
| 0.9 | 0.1 | 2420 | 2481 | 2355 | 2417 |
| 0.85 | 0.2 | 3041 | 4683 | 4028 | 3744 |

We observed similar results for all the considered datasets. Even though `cora` seems to select *uncertain* as the best deterministic alternative, a deeper look at the experimental data reveals that the 0.5 matching probability bucket is the one containing most edges in `cora`, and drawing edges from this bucket closely resembles random selection. This confirms the effectiveness of our strategy.

3.5.2 Sparse Graphs

For sake of comparison, the experiments so far report performance of our algorithms and previous strategies in a crowd-only setting. In practice, adopting a crowd-only approach is prohibitively expensive, and people often remove obvious non-matching pairs during a pre-processing phase. Then, they ask the crowd to examine the remaining pairs, which lead to a relatively sparse graph. We note that even in the crowd-only setting, our strategies are far from asking the crowd the complete graph. For instance, Figure 3.5b shows that `adaptive()` reaches its maximum f-score for `captchas` after less than 200 queries, which is less than 0.6% of the size of the complete graph (see Table 3.2). Anyway, sparsification is useful in many ER applications, and necessary when computing all pairwise matching probabilities is

prohibitively expensive (e.g. `dblp` dataset with more than 3M records). In this section, we evaluate our strategy in different sparse graph experimental settings. We show that:

- our algorithms outperform previous strategies also when obviously non-matching pairs are removed (we use `cora`);
- our techniques can scale over a large number of records, where building a sparse graph is the only viable option (`dblp`).

In this framework, `cora` can be thought of as a portion of `dblp` based on a small set of authors. We consider two sparsity models, that we refer to as *node* partition and *edge* filtering.

- In the node partition model, used by `waldo()`, obviously non-matching nodes (e.g., watches and dishwashers in an e-commerce dataset) are put in separate “domains” and cross-domain edges are consistently removed. The result is a collection of disconnected complete subgraphs that can be resolved independently.
- In the edge filtering model, used by `dense()` and `votes()`, obviously non-matching edges are removed either by a matching probability threshold or other cheap procedures such as (overlapping) *blocking*. The result is a sparse graph, possibly consisting of several connected components (not necessarily cliques).

Node partition. Figure 3.8a shows that our strategies outperform `waldo()` on `cora`, which is akin to a complete subgraph of `dblp`. Since the `cora` domain does not have special characteristics with respect to the others, we expect `adaptive()` to outperform `waldo()` on the whole `dblp`. We note that time complexity of `waldo()` is high [192] and it has been experimentally tested over small datasets. Running `waldo()` on `dblp` does not appear to be feasible.

Edge filtering. Figure 3.9c compares our strategies with `dense()` and `votes()`, with a matching probability threshold set to 0.4. We remove edges below 0.4 and consider

them as given to be negative. Sparsification removes 96% of the edges generating a sparse graph with 309 different components. `adaptive()` is robust to sparsity: its F-score is close to complete `cora` experiments, such as Figure 3.6c. In the complete graph setting, indeed, most low probability edges won't be queried anyway (only a few of them for cluster splits).

Scalability. Figure 3.9d shows the progressive F-score of `adaptive()` over the large `dblp` dataset, as the error probability in synthetic oracle answers increases. We do not show `dense()` and `votes()` because of their time and memory requirements (as a frame of comparison, the computation of `dense()` took 4 days on `cora`). Initial F-score growth of `adaptive()` closely follows `ideal()` in the perfect oracle setting, then flattens out after the initial growth because of the final stage of the node phase, and eventually after edge phase begins, F-score converges to 100% F-score.

3.6 Related Work

We discussed the background of ER in Chapter 2. This section summarizes the prior work from the lens of robustness towards oracle error.

Querying individual crowd workers. The strategies in [110] query individual crowd workers, rather than making use of a crowd abstraction, and every answer represents a *vote*. The goal is to achieve a clear majority of **Yes** or **No** answers for some pairs (u, v) , and use those high confidence pairs for building clusters. Also, the strategies described in [191] query individual crowd workers, but assume that they have a known *error rate* p^E . In this setting, each **Yes/No** answer for a given pair of items can be interpreted as a matching probability: $1 - p^E$ if the pair is supposed to be matching, and p^E otherwise. (An information theoretic perspective of it is provided in [143].) Such strategies start from an initial clustering based on prior probabilities and then refine the solution by asking the crowd. Finally, the work in [192] uses the same idea of fixed crowd-worker error rate than [191] and considers, in addition, a

combination of k -node queries involving multiple items (for instance, with $k = 3$, “are u , v and z the same entity?”) and classical pairwise queries. The k -node queries can be asked to a real crowd-sourcing platform with less cost (bit higher error) than the corresponding sum of $\binom{k}{2}$ pairwise queries, therefore, the cost of replicated queries for a single pair can be amortized. All the above works use various robust graph clustering mechanisms before returning the solution to a user. Our work considers the oracle as an abstraction instead of dealing with individual crowdworkers. The major benefit behind our approach is that the oracle can leverage answer-quality mechanisms [42, 198] to better aggregate the responses of different crowd workers. Additionally, we propose a generic error correction layer that can be applied on top of other oracle strategies to make them robust.

Other Related Works. Wang et al. [196] describe a hybrid human-machine framework CrowdER, that automatically detects pairs that have a high likelihood of matching, which are then verified by humans. Gokhale et al. [103] propose a hybrid approach for the end-to-end workflow, making effective use of active learning via human labeling. Records are partitioned into entities using the rules learned from classifiers. Chai et al. [50] provide a grouping mechanism based on similarity for reducing the number of questions (instead of all pairs in each group, they ask only one pair). In [50] the authors devise a partial ordering-based technique to resolve entities, which applies for records of text attributes. Progressive F-score has been discussed in [202, 161, 109]. Finally, estimating or improving crowd accuracy in general [68, 98, 125, 28, 42] is outside the scope of this work.

3.7 Summary and Future Work

Most of the prior work assumed perfect oracle answers to perform entity resolution. However, these techniques are sensitive to noise and can generate arbitrarily worse clusters. This chapter formalizes the problem of generating clusters using noisy

binary oracle queries and proposes a cost-effective error correction layer which can be applied on perfect oracle-based strategies to make them robust. Additionally, we present three pipelines `adaptive()`, `eager()` and `lazy()` which use the ideas of Hybrid [83] along with a random graph toolkit to provide high progressive F-score with provable guarantees. The experimental evaluation confirms our theory and superiority of proposed techniques over the prior literature.

The key takeaways from the chapter are as follows.

- The random expansion toolkit helps to achieve high precision with a low overhead in the number of oracle queries.
- The redundancy in oracle queries can be tuned based on noise in oracle answers and the toolkit can be applied on top of any perfect-oracle strategy.
- Among the different pipelines, `adaptive()` achieves the best progressive F-score and generates the most accurate clusters.
- The error correction layer is not helpful to correct mistakes in small clusters with high noise.

In our future work, we plan to apply our results to temporal record linkage, where the user is given multiple versions of the same entity, coming from real-world at different *times*. In this setting, error probability is expected to be smaller if two records are farther in time, and higher for similar representations of the same entity.

CHAPTER 4

ENTITY RESOLUTION WITH SUPERVISION: SCALABILITY

Chapter 3 presented the effect of error on the quality of generated clusters. It assumed access to an input graph H with edges $A \subseteq V \times V$, which is the output of the blocking phase of the ER pipeline. In the absence of blocking $A = V \times V$ contains all pairs of records, which cannot be enumerated for million-scale datasets. This chapter is devoted to discuss some common challenges with respect to large scale datasets and present a progressive blocking approach to improve scalability.

Section 4.1 discusses the limitations of prior work and highlights high-level contributions of the chapter with an example. Sections 4.2 and 4.3 provide preliminary definitions and a high-level description of the proposed approach. Sections 4.4 and 4.5 explain details of the block intersection and block scoring methods, respectively. Section 4.6 provides theoretical analysis of its effectiveness and Section 4.7 presents the empirical evaluation on large-scale datasets. Section 4.8 discusses the related work specific to blocking and we conclude in Section 4.9.

4.1 Introduction

As discussed in Chapter 2, blocking constitutes the first step of ER that selects sub-quadratic number of record pairs to compare in the subsequent steps. Blocking groups similar records into *blocks* and then selects pairs from the “cleanest” blocks – i.e., those with fewer non-matching pairs – for further comparisons. The literature is rich with methods for building and processing blocks [159], but depending on the data

Table 4.1: Sample records (we omit schema information) referring to 4 distinct entities. r_i^e represents the i -th record referring to entity e . Records in the first two rows refer to a Chevrolet Corvette C6 ($c6$) and a Z6 ($z6$). Records in the last two rows to a Chevrolet Malibu (ma) and a Citroën C6 (ci) (same model name as Corvette C6 but different car).

| | |
|--|---|
| r_1^{c6} : 'chevy corvette c6' | r_2^{c6} : 'chevy corvette c6 navigation' |
| r_3^{c6} : 'chevrolet corvette c6' | r_1^{z6} : 'corvette z6 navigation' |
| r_1^{ma} : 'chevy malibu navigation' | r_2^{ma} : 'chevrolet chevy malibu' |
| r_3^{ma} : 'chevrolet malibu' | r_1^{ci} : 'citroen c6 navigation' |

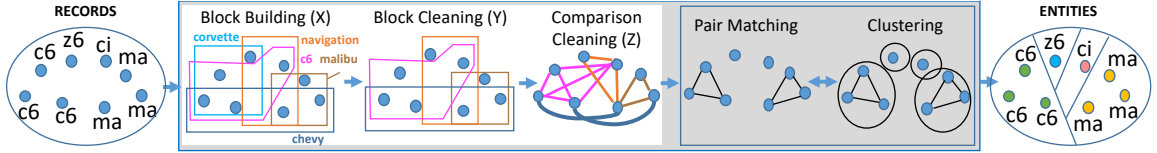


Figure 4.1: Illustration of a standard blocking pipeline. Block building, block cleaning and comparison cleaning sub-tasks are highlighted in white. The downstream ER algorithm is shown in gray. Description of each record is reported in Table 4.1.

set at hand, different techniques can either leave too many matching pairs outside, leading to incomplete ER results and low effectiveness, or include too many non-matching pairs, leading to low efficiency.

This chapter presents a new *progressive* blocking technique that overcomes the above limitations by short-circuiting the two operations – blocking and pair comparisons (pair matching and clustering) – that are traditionally solved sequentially. The method starts with an aggressive blocking step, which is efficient but not very effective. Then, it computes a limited amount of ER results on a subset of pairs selected by the aggressive blocking, and sends these partial (matching and non-matching) results from the ER phase back to the blocking phase, creating a “loop”, to improve blocking effectiveness. In this way, blocking can progressively self-regulate and adapt to the properties of each dataset, with no configuration effort. We illustrate the shortcomings of prior approaches and our blocking method, that we call **pBlocking**, in the following example.

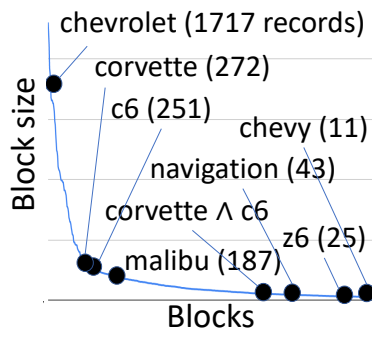


Figure 4.2: Block size distribution (standard blocking) for the real `cars` dataset used in our experiments.

Example 2. Consider the records in Table 4.1 from the `cars` dataset used in our experiments, and a standard schema-agnostic blocking strategy \mathcal{S} such as [155]. As shown in Figure 4.1, we consider three blocking sub-tasks [159]. First, during block building, \mathcal{S} creates a separate block for each text token (we only show the blocks ‘corvette’, ‘navigation’, ‘malibu’, ‘c6’ and ‘chevy’). Then, during block cleaning, \mathcal{S} uses a threshold to prune out all the blocks of large size. Depending on the threshold value (using the block sizes in the entire `cars` dataset, shown in Figure 4.2), we can have any of the following extreme behaviors. (Note that no intermediate setting of the threshold can yield a sparse set of candidates that is at the same time complete.)

- Aggressive blocking: \mathcal{S} prunes every block except the smallest one (‘chevy’) and returns (r_1^{c6}, r_2^{c6}) , (r_1^{c6}, r_1^{ma}) , (r_2^{c6}, r_1^{ma}) and (r_1^{ma}, r_2^{ma}) , missing r_3^{c6} and r_3^{ma} .
- Permissive blocking: \mathcal{S} prunes only the largest block (‘chevrolet’) and returns many non-matching pairs.

Finally, during comparison cleaning, \mathcal{S} can use another threshold to further prune out pairs sharing few blocks, e.g. by using meta-blocking [154]. As in block cleaning, different threshold values can yield aggressive or permissive behaviours. Note that matching pairs such as (r_2^{c6}, r_3^{c6}) share the same number of blocks (‘corvette’ and ‘c6’) as non-matching pairs such as (r_2^{c6}, r_1^{z6}) (‘corvette’ and ‘navigation’). (Even worse, ‘c6’ is larger than ‘navigation’.)

pBlocking can solve these problems in a few rounds: the first round does aggressive blocking, the second round does more effective blocking by making targeted updates according to partial ER results, and so on. Examples of such updates to the blocking result are discussed below.

1. Creation of new blocks that help *inclusion* of $(r_1^{c6}, r_3^{c6}), (r_2^{c6}, r_3^{c6})$: **pBlocking** creates a *new* block ‘corvette \wedge c6’ with records present in both blocks ‘corvette’ and ‘c6’. This block is much smaller than its two constituents and has only Corvette C6 cars.
2. Adaptive cleaning to help *inclusion* of $(r_1^{ma}, r_3^{ma}), (r_2^{ma}, r_3^{ma})$: **pBlocking** can discourage pruning of block ‘malibu’ that contains Chevrolet Malibu cars, even if it is a large block;
3. Adaptive cleaning to help *exclusion* of non-matching pairs: **pBlocking** can encourage pruning of block ‘navigation’ that contains no matching pairs, even if it is a small block.

After a few rounds of updates like the above, **pBlocking** returns all the matching pairs with very few non-matching pairs. Note that after the last round, the ER output can be computed on the resulting pairs as in the traditional setting. Updates of type (1) are performed via a new *block intersection* algorithm, while (2) and (3) are performed by a new *block scoring* method. By construction, when the blocking scores converge, the entire blocking result also converges.

The main contribution of this chapter is a new blocking methodology with both high efficiency and effectiveness in a variety of application scenarios. Since **pBlocking** can in principle start off using any blocking strategy, it represents not only a new approach but also a way to “boost” traditional ones. **pBlocking** works seamlessly across different entity cluster size distributions such as:

- *small entity clusters*, where, using block intersection, **pBlocking** can recover entities such as Corvette C6 consisting of few records sharing large and dirty blocks.
- *large entity clusters*, where, using block scoring, **pBlocking** can recover entities such as Chevrolet Malibu consisting of many records sharing large and clean blocks.

We prove theoretically and show empirically that, with a few rounds and a limited amount of partial ER results, our progressive blocking method can provide a significant boost in blocking effectiveness without penalizing efficiency. Specifically, we (i) demonstrate fast convergence and low space and time complexity ($O(n \log^2 n)$, where n is the number of records) of **pBlocking**; (ii) report experiments achieving up to 60% increase in recall when compared to state-of-the-art blocking [67], and up to 5x boost in efficiency. Finally, we observe that **pBlocking** can yield up to 70% increase on the F-score of the final ER result, thus confirming the substantial benefits of our approach.

4.2 Blocking Preliminaries

Let V be the input set of records, with $|V| = n$. Consider an (unknown) graph $\mathcal{C}^* = (V, E^+)$, where $(v_i, v_j) \in E^+$ means that v_i and v_j represent the same entity. \mathcal{C}^* is transitively closed, that is, each of its connected components $C^* \subseteq V$ is a clique representing a distinct entity. We call each clique a *cluster* of V , and refer to the partition induced by \mathcal{C}^* as the ER *ground truth*.

Definition 3 (Pair Recall). *Given a set of matching record pairs $A \subseteq V \times V$, Pair Recall is the fraction of pairs $(u, v) \in E^+$ that can be either (i) matched directly, because $(u, v) \in A$, or (ii) indirectly inferred from other pairs $(u, w_0), (w_0, w_1), \dots, (w_c, v) \in A$ by connectivity.*

A formal definition of the blocking task follows.

Problem 3 (Blocking Task). *Given a set of records V , group records into possibly overlapping blocks $\mathcal{B} \equiv \{B_1, B_2, \dots\}$, $B_i \subseteq V$ and compute a graph $P = (V, A)$, where $A \subseteq \{(u, v) : \exists B_i \in \mathcal{B} \text{ s.t. } u \in B_i \wedge v \in B_i\}$, such that A is sparse ($|A| \ll \binom{n}{2}$) and A has high Pair Recall. We refer to P as the blocking graph.*

The blocking graph P is the final product of blocking and contains all the pairs that can be considered for pair matching. The efficiency and effectiveness of the blocking method is measured as Pair Recall (PR) of (the set of edges in) P and the number of edges in it for a certain PR, respectively. Blocking methods consist of three sub-tasks as defined by [159]: block building, block cleaning and comparison cleaning. In the following, we describe each of these steps and the corresponding methods in the literature.

Block building (\mathcal{BB}) takes as input V and returns a block collection \mathcal{B} , by assigning each record in V to possibly multiple blocks. The popular *standard blocking* [155] strategy creates a separate block B_t for each token t in the records and assigns to B_t all the records that contain the token t . To tolerate spelling errors, *q-grams blocking* [106] considers character-level q-grams instead of entire tokens. Other strategies include *canopy clustering* [147] and *sorted neighborhood* [117]. Canopy clustering iteratively selects a random seed record r , and creates a new block B_r (or a canopy) with all the records that have a high similarity with r with respect to a given similarity function (e.g., using a subset of features [147]). We can use different similarity functions to build different sets of canopies. Sorted neighborhood sorts all the records according to multiple sort orders (e.g., each according to a different attribute [117]) and then it slides a window w of tokens over each ordering, every time creating a new block B_w . Blocks have the same number of distinct tokens but the number of records in a block can vary significantly. Multiple block building strategies can be employed at the same time to generate the collection of blocks \mathcal{B} .

Block cleaning (\mathcal{BC}) takes as input the block collection \mathcal{B} and returns a subset $\mathcal{B}' \subseteq \mathcal{B}$ by pruning blocks that may contain too many non-matching record pairs. Block cleaning is typically performed by assigning each block a *score* : $\mathcal{B} \rightarrow \mathbb{R}$ with a block scoring procedure and then pruning blocks with low score. Traditional scoring strategies include functions of block sizes such as TF-IDF [78, 156].

Comparison cleaning (\mathcal{CC}) takes as input the set X of all the intra-block record pairs in the block collection \mathcal{B}' (which is a subset of the intra-block record pairs in \mathcal{B}) and returns a graph $P = (V, A)$, with $A \subseteq X$, by pruning pairs that are likely to be non-matching. Comparison cleaning is typically performed by assigning each pair a *weight* : $X \rightarrow \mathbb{R}$ and then pruning pairs with low weight. Weighting strategies include *meta-blocking* [154] possibly with active learning [176, 67]. In classic meta-blocking, $weight(u, v)$ corresponds to the number of blocks in which u and v co-occur, based on the assumption that that more blocks a record pair shares, the more likely it is to be matching.¹ The recent BLOSS strategy [67] employs active learning on top of the pairs generated by meta-blocking, and learns a classifier using features extracted from the blocking graph for further pruning.

We denote with $\mathcal{B}(X, Y, Z)$ a blocking strategy that uses the methods X , Y , and Z , respectively for block building, block cleaning and comparison cleaning. The strategy used in our **cars** example can be thus denoted as $\mathcal{B}(\textit{standard blocking}, \textit{TF-IDF}, \textit{meta-blocking})$.

After blocking. Typical ER algorithms include *pair matching* and entity *clustering* operations. Such operations label as “matching” the pairs referring to the same entity and “non-matching” otherwise, and typically require the use of a classifier [151] or a

¹This assumption holds for block building methods such as standard blocking, q-grams blocking and sorted neighborhood with multiple orderings [117], and extends naturally to canopy clustering by using multiple similarity functions.

crowd [197]. Clustering consists of building a possibly noisy clustering \mathcal{C}' according to labels, and has been discussed in Chapter 3.

4.3 Overview of pBlocking

Analogous to traditional blocking methods, **pBlocking** takes as input a collection V of records and returns a blocking graph P . A high-level view of the methods introduced in **pBlocking**, for each of the main blocking sub-tasks of Section 4.2, is provided below. Such methods, unlike previous ones, can leverage feedback of partial ER results.

Block building in **pBlocking** constructs new blocks arranged in the form of a *hierarchy*. First level blocks are initialized with blocks generated by a traditional method (e.g., standard blocking, sorted neighborhood, canopy clustering or q-gram blocking). Subsequent levels contain intersections of the blocks in the previous levels. **pBlocking** can use feedback from the partial ER output to build intersections such as ‘corvette \wedge c6’ that can lead to new, cleaner blocks, and avoid bad intersections such as ‘corvette \wedge chevrolet’ that would not improve the fraction of matching pairs in P (Chevrolet Corvette C6 and Z6 are different entities). We discuss block intersection in Section 4.4.

Block cleaning in **pBlocking** prunes dirty blocks based on feedback-based scores. First round scores are initialized with a traditional method (e.g. TF-IDF). Then, scores are refined based on feedback by combining two quantities: the fraction $p(B)$ of matching pairs in a block B , and the block uniformity $u(B)$, which captures the distribution of entities within the block ($u(B)$ is the inverse of *perplexity* [141]). Since the goal of blocking phase is to identify blocks that have a higher fraction of matching pairs and fewer entity clusters, we combine the above values as $score(B) = p(B) \cdot u(B)$. **pBlocking** can use feedback from the partial ER output to estimate $p(B)$ and $u(B)$, yielding high scores for clean blocks such as ‘malibu’ (high $p(B)$ and high $u(B)$) and

Algorithm 9 Our blocking method pBlocking

Require: Records V , methods X , Y , and Z for each blocking step. Default: $X=token\ blocking$, $Y=TF-IDF$ and $Z=meta-blocking$.

Ensure: Blocking graph P

```
1:  $\mathcal{C} \leftarrow \emptyset$ 
2:  $B \leftarrow$  build the first level of block hierarchy with method  $X$ 
3:  $scores \leftarrow$  initialize block scores using method  $Y$ 
4:  $P \leftarrow$  block cleaning and comparison cleaning with method  $Z$ 
5:  $P_{new} \leftarrow \emptyset$ 
6: for round=2; round  $\leq 1/\phi \wedge P \neq P_{new}$ ; round++ do
7:   while ER progress is less than  $\phi$  do
8:      $\mathcal{C} \leftarrow$  Execute an incremental step of method  $W$  for pair matching and
       clustering on  $P$ 
9:   end while
10:   $score \leftarrow$  update the block scores according to  $\mathcal{C}'$  //Feedback
11:   $B \leftarrow$  update the block hierarchy based on  $score$ 
12:   $P \leftarrow P_{new}$ 
13:   $P_{new} \leftarrow$  block cleaning and comparison cleaning with  $Z$ 
14: end for return  $H$ 
```

low scores for dirtier blocks such as ‘navigation’ (low $p(B)$ and low $u(B)$), and ‘c6’ (low $u(B)$). We discuss block scoring in Section 4.5.

Finally, *comparison cleaning* in pBlocking is implemented with a traditional method such as meta-blocking.

Workflow. Algorithm 9 describes the pBlocking workflow and how the introduced blocking methods can be used. We denote with $\text{pBlocking}(X, Y, Z)$ a progressive blocking strategy that uses the methods X , Y and Z , respectively for building the first level of the block hierarchy, initializing the block scores, and performing comparison cleaning as described in Algorithm 9. In our **cars** example, we have $\text{pBlocking}(\text{standard blocking}, TF-IDF, \text{meta-blocking})$.

We first initialize the set of clusters \mathcal{C} , the block hierarchy and the block scores (**lines 1–3**). The next step (**line 4**) consists of computing the first version of the blocking graph P according to the selected method for comparison cleaning (e.g., meta-blocking). The graph P is then progressively updated, round after round (**lines 6–12**). In order to activate the feedback mechanism, pBlocking needs to interact

with an ER algorithm W for pair matching and clustering operations (**line 7–8**). Algorithm W is executed over P until it makes a *progress* of ϕ with $\phi \in [0, 1]$, that is, until $\phi \cdot n \log^2 n$ record pairs have been processed since the previous round.² At that point, the algorithm W is interrupted, \mathcal{C} is updated (**line 8**) and sent as feedback to all of **pBlocking**’s components. Based on such feedback, we update the function $score(B) = p(B) \cdot u(B)$ (**line 9**) and construct new blocks in the form of a hierarchy (**line 10**). Higher score blocks are used to enumerate the most promising record pairs and generate the updated blocking graph P_{new} (**lines 11-12**). When either the maximum number of rounds $\frac{1}{\phi}$ has been reached (setting $\phi = 1$ is the same as switching off the feedback) or the blocking result converges ($P = P_{new}$), **pBlocking** terminates by returning P .

We present a formal analysis of the effectiveness of **pBlocking** in Section 4.6. We refer to Section 4.7 for experiments. Due to its robustness to different choices of the pair matching algorithm W , we do not include W in **pBlocking**’s parameters (differently from X, Y, Z). Natural choices for W include *progressive* ER strategies that can process P in an online fashion and compute \mathcal{C} incrementally [192, 194, 151]. However, traditional algorithms, such as [78] can be used as well by adding *incremental ER* techniques [109, 200] on top.

4.3.1 Computational Complexity

For efficiency, it is crucial to ensure that the total time and space taken to compute P is close to linear in n . Since every round of **pBlocking** comes with its time and space overhead, we first describe how to bound the complexity of every round and then discuss how to set the parameter ϕ in Algorithm 9 (and thus the maximum number of rounds) to bound the complexity of the entire workflow.

²For algorithms such as [194], progress can be defined as a fraction $\phi \cdot n$ of processed *records* since the previous round.

Round Complexity. `pBlocking` implements the following strategies to decrease overhead of each round.

Efficient block cleaning. We compute the block scores by sampling $\Theta(\log n)$ records from each of the top $O(n)$ high-score blocks computed in the previous round.

Efficient comparison cleaning. For simplicity, we build P by enumerating at most $\Theta(n \log^2 n)$ intra-block pairs by processing blocks in non-increasing block score.

Based on the above discussion, we have Lemma 1.

Lemma 1. *A single round of `pBlocking`(X, Y, Z), such as `pBlocking`(standard blocking, TF-IDF, meta-blocking) has $O(n \log^2 n)$ space and time complexity.*

Workflow Complexity. As discussed in Section 4.6, ϕ can be set to a small constant fraction. Thus, along with Lemma 1, this guarantees an $O(n \log^2 n)$ complexity for the entire workflow. Experimentally a smaller ϕ value yields higher final recall, thus as a default we set $\phi = 0.01$, yielding a maximum of 100 rounds. Although such a ϕ value gets the best trade-off between effectiveness and efficiency in our experiments, we also observe that slight variations of its setting do not affect the performance much (Section 4.7), demonstrating the robustness of `pBlocking`.

4.4 Block Building

One of the major challenges of block building (\mathcal{BB}) is that when generating candidate pairs that capture matches it can also generate a number of non-matching pairs. This phenomenon is highly prevalent in datasets with very few matching pairs. To overcome this challenge, our *block building by intersection* algorithm takes a collection of blocks B_1, \dots, B_m built by a traditional method for \mathcal{BB} and creates smaller clean blocks out of large dirty ones, thus contributing to the recall of the blocking graph without adding extra non-matching pairs. An *intersection block hierarchy* \mathcal{H} is constructed as follows. Let the first layer be B_1, \dots, B_m . Then blocks in layer L consist of the intersection of L distinct blocks in the first layer.

Example 3. Consider our *cars* example in Section 4.1, and the blocks corresponding to tokens ‘corvette’ and ‘c6’, namely B_{corvette} , and B_{c6} . A sample block in the second level of \mathcal{H} is $B_{\text{corvette}, \text{c6}} = B_{\text{corvette}} \cap B_{\text{c6}}$. When we build the new block, we only include records containing the two tokens ‘corvette’ and ‘c6’ (possibly non consecutively), thus obtaining a cleaner block than the original ones.

Refined blocks. We refer to the newly created block as a *refined* block, and to the intersecting blocks as *parent* blocks. Not all the refined blocks are useful. We need one of the following correlation-based conditions to hold to decide if a refined block $B_{i,j}$ must be kept in \mathcal{H} .

- $\text{score}(B_{i,j}) > \text{score}(B_i) \cdot \text{score}(B_j)$, that is, the score of the refined block is higher than the combined score of the parent blocks.
- The existence of a randomly chosen record r in blocks B_i and B_j is positively correlated, i.e. $\Pr[r \in B_{i,j}] = |B_{i,j}|/n > \Pr(r \in B_i) \cdot \Pr(r \in B_j)$, which simplifies to $|B_{i,j}| > \frac{|B_i||B_j|}{n}$. For example, the number of common records in blocks corresponding to tokens ‘c6’ and ‘corvette’ is much higher than the common records in blocks corresponding to ‘navigation’ and ‘c6’.

Suppose the maximum depth of the hierarchy is d which is a constant. The construction of refined blocks can take $O(n^d)$ time if the number of blocks considered in the first layer is $O(n)$. For efficiency, we iterate over the records (linear scan) and for each record r , we consider all pairs of blocks that contain r as candidates to generate blocks in the different levels of the hierarchy. The following lemma bounds the total number of refined blocks across the hierarchy.

Lemma 2. *The number of blocks present in \mathcal{H} is $O(n)$ if each record r is present in a constant number of blocks.*

Proof. Our algorithm considers each record $u \in V$ and generates intersection blocks by performing conjunction of blocks that contain the record u . Suppose the record u

Algorithm 10 Block Layers Creation

Require: Set of records V , depth d

Ensure: Layer set $\{L_1, \dots, L_d\}$

```
1: for  $i = 1; i \leq d; i++$  do
2:    $L_i \leftarrow \phi$ 
3: end for
4:  $\text{processed} \leftarrow \phi$ 
5: for  $v \in V$  do
6:    $\text{blockLst} \leftarrow \text{getBlocks}(v)$ 
7:   for  $i = 2; i < d; i++$  do
8:     for  $\mathcal{B} = \{B_j : B_j \in \text{blockLst}\}, |\mathcal{B}| = i$  do
9:        $B' = \cap_{B_j \in \mathcal{B}} B_j$ 
10:      if  $B' \notin \text{processed}$  then
11:         $L_i.\text{append}(B')$ 
12:         $\text{processed.append}(B')$ 
13:      end if
14:    end for
15:     $\text{blockLst} \leftarrow L_i$ 
16:  end for
17: end for
```

is present in γ_u blocks at the first layer. Then the maximum number of blocks present in \mathcal{H} that contain u is $\sum_{i=1}^d \binom{\gamma_u}{i}$. Assuming γ_u is a constant, the maximum number of blocks in the hierarchy is $n \sum_{i=1}^d \binom{\gamma_u}{i} = O(n)$. \square

Refinement algorithm. We are now ready to describe pBlocking's intersection method for building the block hierarchy. Our method has two steps:

- (Algorithm 10) The first step creates all possible blocks considering the intersection search space.
- (Algorithm 11) The cleaning phase removes the blocks that do not satisfy the correlation criterion described above.

Algorithm 10 describes the creation step, which iterates over all the records in the corpus and creates all possible blocks per record. The list of all blocks to which a record belongs is constructed (denoted by blockLst) and the new blocks are added in different layers. The layer of the new block depends on the number of intersecting

Algorithm 11 Layer Cleaning

Require: Layer set $\{L_1, \dots, L_d\}$ **Ensure:** Cleaned Layer set $\{L_1, \dots, L_d\}$

```
1: for  $i = 2; i < d; i++$  do
2:   for  $\text{block} \in L_i$  do
3:      $\text{parentLst} \leftarrow \text{getParents}(\text{block})$ 
4:     if  $\prod_{p \in \text{parentLst}} \text{score}(p) < \text{score}(\text{block})$  or  $\prod_{p \in \text{parentLst}} \frac{|L_{i-1}[p]|}{n} < \frac{|L_i[\text{block}]|}{n}$ 
5:       then
6:         continue
7:       else
8:          $L_i.\text{remove}(\text{block})$ 
9:       end if
10:    end for
11: end for
```

blocks that constitute the new block. Then, the cleaning step in Algorithm 11 iterates over the different layers and keeps only the blocks that satisfy the score or size requirements. For a block in layer q , `getParents()` identifies the two blocks which are in layer $(q - 1)$ whose conjunction generates the block being considered. If these parents have been removed during the cleaning phase, then their parents are considered and the process is continued recursively until we end up at the ancestors present in the list of blocks.

Block Layers Creation (Alg. 10) constructs all blocks in the form of a hierarchy and Layer Cleaning (Alg. 11) deactivates the blocks that do not satisfy the correlation requirements. Since the result of Block Layers Creation does not change in different `pBlocking` iterations, decoupling the creation component from the cleaning component (which changes dynamically) allows for more efficient computation.

Time complexity. Assuming the depth of the hierarchy is a constant, Algorithms 10 and 11 operate in time linear in the number of records n . Block refinement takes 3 minutes for a dataset with one million records in our experiments.

4.5 Block Cleaning

Let $A \subset V \times V$ be the pairs selected by blocking phase at a given point (we recall that A is the edge set of the blocking graph $P = (V, A)$) and each considered pair $(u, v) \in A$ has a similarity value denoted by $p_m(u, v)$. A block $B \subseteq V$ refers to a subset of records. Using this notation, we discuss the different methods for scoring blocks and how the scores converge with feedback for effective ER performance.

Block scoring. Block scoring helps to distinguish informative blocks based on their ability to capture records from a single cluster. By selecting pairs within informative blocks, down-stream ER operations can focus on records pairs that have a high probability of being a match. The most common mechanism used in the literature is TF-IDF and it assigns block scores inversely proportional to the block size prioritizing smaller blocks over larger ones. If the dataset has small clusters, such a simple method can work well. However, if the dataset has a skewed cluster size distribution, some large blocks are just uninformative (and are rightfully less preferred by TF-IDF), but others can represent a large cluster and thus should stand out in the scoring. Distinguishing these blocks before pair matching can be difficult, but **pBlocking** provides a way to leverage the feedback.

Specifically, the scoring algorithm of **pBlocking** prioritizes blocks having (a) high fraction of matching pairs measured as matching probability within a block and (b) fewer number of clusters (especially larger clusters) measured as uniformity (a function of entropy of the cluster distribution within a given block B). Lower entropy and hence lower diversity values indicate the representativeness of B towards a particular cluster as opposed to higher entropy values which refer to the presence of many fragmented clusters.

More formally, the matching probability score identifies the probability that a randomly chosen pair $(u, v) \mid u, v \in B$ refers to the same entity and is defined as follows.

Definition 4 (Matching Probability score $p(B)$). *The value $p(B)$ is defined as the fraction of matching pairs within a block B .*

The block uniformity, $u(B)$ captures perplexity of cluster distribution within B measured in terms of its entropy.

Definition 5 (Cluster Entropy $H(B)$). *The cluster entropy of a block, $H(B)$ refers to the entropy of the cluster distribution when restricted to the records present in block B . Mathematically, $H(B) = -\sum_{C \in \mathcal{C}} p_C \log p_C$, where $p_C = |C \cap B|/|B|$ refers to the probability that a randomly chosen node from B belongs to cluster C .*

Using $H(B)$, block uniformity score is defined as follows.

Definition 6 (Block Uniformity $u(B)$). *The block uniformity $u(B) = e^{-H(B)}$ is the inverse of perplexity [141] of the cluster distribution within the block where perplexity refers to the exponential of cluster distribution entropy.*

Example 4. *Suppose that we know that a block B contains records of two clusters C_1 and C_2 and thus we can compute the uniformity of B exactly. If the two clusters are perfectly balanced in B , i.e., $|C_1 \cap B| = 0.5 \cdot |B|$ and $|C_2 \cap B| = 0.5 \cdot |B|$, the entropy is $H(B) = -0.5 \log 0.5 - 0.5 \log 0.5 \approx 0.69$ and thus $u(B) = e^{-H(B)} = 0.5$. If there is some skew, e.g. $|C_1 \cap B| = 0.7 \cdot |B|$ and $|C_2 \cap B| = 0.3 \cdot |B|$, then the entropy is lower $H(B) = -0.7 \log 0.7 - 0.3 \log 0.3 \approx 0.61$ and the uniformity is higher $u(B) \approx 0.54$. In the extreme case where $C_1 \cap B = B$ and $C_2 \cap B = \emptyset$, $H(B) = 0$ and $u(B) = 1$.*

Note that when resolving two duplicate-free datasets where all clusters are of size 2 (also known as Record Linkage) the entropy increases with block size, thus block uniformity yields comparable results to traditional TF-IDF.

Since the goal of block scoring is to identify blocks that have high matching probability and high uniformity, we multiply the two values to get a final estimate of the block score.

Definition 7 (Block Score, $score(B)$). *The score of a block B , $score(B)$, is defined as the product of matching probability score and uniformity score of B . That is, $score(B) = p(B)u(B)$.*

Next, we describe the algorithm to estimate these components of block score. The exact value of matching probability and block uniformity requires complete ER results. However, **pBlocking** estimates these scores initially with the similarity estimates of every pair of records and refines these scores with additional feedback from partial ER results.

Matching probability score. The matching probability score is estimated as the average matching similarity of pairs of records within the block, i.e.:

$$p(B) = \frac{\sum_{u,v \in B} p_m(u, v)}{\binom{|B|}{2}}$$

where $p_m(u, v)$ is estimated as follows:

- for pairs declared as matches, we set $p_m(u, v) = 1$;
- for pairs declared as non-matches, we set $p_m(u, v) = 0$;
- for unlabelled pairs, we use the p_m values computed by common similarity metrics (e.g. via jaccard similarity or the similarity-to-probability mapping as in [161]).

Block uniformity estimation. Estimating uniformity score requires the cluster size distribution in B , which is harder to infer from the prior similarity values. We next describe a mechanism to estimate entropy $H(B)$ needed to compute the uniformity score. We consider each record $u \in B$, and consider the cluster $C(u)$ that contains u . We are interested in computing $\frac{|C(u) \cap B|}{|B|}$ in order to compute entropy $H(B)$. Instead, we compute the expected size of $|C(u) \cap B|$ as $E_u = E[|C(u) \cap B|] = \sum_{v \in B} p_m(u, v)$ based on p_m values of edges incident on u . We compute the expected cluster size for every record $u \in B$ and sort them in non-increasing order. Let L be

the sorted list. Let the first record in the sorted list L , that is, the node with highest expected cluster size in B be u . On expectation u has E_u records in B that belong to $C(u)$. All these records must have similar expected cluster sizes as well. We put u and the next $\lfloor E_u \rfloor$ records from L to a set S_U , assuming that they belong to the same cluster $C(u)$. We recurse on $L \setminus S_U$ until a partition $\{S_U, S_V, \dots\}$ of the block is generated. The size of each partition can be thought of as a rough estimate of the true cluster distribution in B and is used to calculate the entropy.

Example 5. Consider a block B , with $|B| = 10$. Let $[u_1, u_2 \dots u_{10}]$ be the corresponding list L of records sorted in non-increasing E_{u_i} values. If $E_{u_1} = \sum_{i \in 2 \dots 10} p_m(u_1, u_i) = 6.6$ we set $S_{U1} = \{u_1 \dots u_{1+\lfloor E_{u_1} \rfloor}\} = \{u_1 \dots u_7\}$ and then consider the next node in L which is u_8 . If $E_{u_8} = \sum_{i \in 9, 10} p_m(u_8, u_i) = 2$ we set $S_{U8} = \{u_8 \dots u_{8+\lfloor E_{u_8} \rfloor}\} = \{u_8 \dots u_{10}\}$ and then finish. As $|S_{U1}| = 0.7 \cdot |B|$ and $|S_{U8}| = 0.3 \cdot |B|$ we estimate $u(B) = e^{-0.7 \log 0.7 - 0.3 \log 0.3} \approx 0.54$.

The value returned by this mechanism is generally an under-estimate of the true entropy $H(B)$ but in practice it can approach $H(B)$ quickly with increasing feedback data and turns out to be very efficient. Section 4.6.2 discusses this convergence rate in different application scenarios.

Efficient block cleaning. Traditional scoring strategies such as TF-IDF are based on block size computation and thus operate in linear time. Computing our $score(B)$ values requires instead to process intra-block pairs and thus yields potentially quadratic computation. Hence, we sample $\Theta(\log n)$ records from each block for its score computation. This strategy operates in $\Theta(\log^2 n)$ time and takes less than 1 minute for a data set with $1M$ records in our experiments. Our sampling strategy can give a constant approximation of the matching probability scores estimated using all the records within each block.

4.6 Analysis of pBlocking

In this section we present a theoretical analysis of the effectiveness of pBlocking. We first analyze the pair recall of blocking in the absence of feedback by considering a natural generative model for block creation. Next we analyze the effect of feedback on block scoring and the final recall.

4.6.1 Pair Recall without Feedback

We start by giving the following basic lemma below.

Lemma 3. *The blocking graph $P = (V, A)$ contains a spanning tree for each clique C^* of $\mathcal{C}^* = (V, E^+)$ iff the Pair Recall is 1.*

Proof. If A contains a spanning tree for each clique C^* , then any pair $(u, v) \in A \cap E^+$ contributes directly to the recall. All pairs of records (u, v) that refer to the same entity, $(u, v) \in E^+$ and are not present in A , $(u, v) \notin A$ can be inferred from the edges in the spanning tree using transitivity, ensuring PR=1. For the converse, let us assume that $\exists C^* \in \mathcal{C}^*$ such that A does not contain any spanning tree over the matching edges. This implies that C^* is split into multiple components (say C_1, C_2) when restricted to $A \cap E^+$ edges. In this case, the collection of matching edges joining these components, $\{(x, y), \forall x \in C_1, y \in C_2\}$ cannot be inferred as none of these edges are processed by the mentioned ER operations, yielding pair recall of P less than 1. \square

Our probabilistic model for block creation is motivated by the standard blocking [155], sorted neighborhood [117] and canopy clustering [147] algorithms which aim to generate blocks that capture high similarity candidate pairs. This model of block generation is closely related to Random Geometric Graphs [162] which were proposed by Gilbert in 1961 and have been used widely to analyze spatial graphs.

Definition 8 (Random Geometric Graphs). *Let S^t refer to the surface of a t -dimensional unit sphere, $S^t \equiv \{x \in \mathbb{R}^{t+1} \mid \|x\|_2 = 1\}$. A random geometric graph $G_t(V, E)$ of n*

vertices V , has parameters $t \in \mathbb{Z}^+$ and a real number $r \in [0, 2]$. It assigns each vertex $v_i \in V$ to a point chosen independently and uniformly at random within S^t and any pair of vertices $v_i, v_j \in V$ are connected if the distance between their respective points is less than r .

Now, we define the probabilistic block generation model.

Definition 9 (Probabilistic Block Generation). *The block generation model places the records $u \in V$ independently and uniformly at random within S^t . Every record u constructs a ball of volume $(\alpha \log n/n)$ with u as the center, where α is a given parameter and all points within the ball are referred to as block B_u .*

The set of points present within a ball B_u can be seen as high similarity points that would have been chosen as blocking candidates in the absence of feedback. Our probabilistic block generation model constructs n blocks, one for each node and every pair of records that co-occur in a block $B_u, u \in V$, has an edge in the blocking graph $P_g(V, E)$ (subscript g to emphasize generative model). Next we analyze pair recall of $P_g(V, E)$.

Notation. Let $d(u, v)$ refer to the distance between records u and v and r_ϵ refer to the radius of an ϵ -volume ball³ in t dimensions. Under these assumptions we first show that the expected number of edges in the blocking graph P_g is at least $\frac{\alpha(n-1)\log n}{2}$ and then that $P_g(V, E)$ has recall $<< 1$.

Lemma 4. *The blocking graph $P_g(V, E)$ contains at least $\alpha \frac{(n-1)\log n}{2}$ candidate pairs on expectation.*

Proof. Each record $u \in V$, constructs a spherical ball of volume $\alpha \log n/n$, with u as the center and all points within the ball are added as neighbors of u in the blocking graph. Hence, the number of expected neighbors of u within the ball is

³ $\epsilon = O(r_\epsilon^t)$.

$\alpha(n-1)\log n/n$. There are a total of n such blocks (one ball per record) and each of the candidate pairs (u, v) is counted twice (once for the block B_u and once for the block B_v). Hence there are a total of $\frac{\alpha(n-1)\log n}{2}$ such candidate pairs. Notice that this analysis ignores the candidate pairs (u, v) which are more than $r_{\alpha \log n/n}$ from each other but are connected in the blocking graph. This would happen if they are present together in another block centered at $w \in V \setminus \{u, v\}$, that is $\exists w \mid d(u, w) \leq r_{\alpha \log n/n}$ and $d(v, w) \leq r_{\alpha \log n/n}$. This shows that the total number of candidate pairs in the blocking graph is at least $\frac{\alpha(n-1)\log n}{2}$. \square

Additionally, $P_g(V, E)$ has the following property:

Lemma 5. *A blocking graph P_g is a subgraph of a random geometric graph G_t with $r = 2r_{\alpha \log n/n}$*

Proof. Following the construction of blocking graph, if the distance between any pair of vertices $u, v \in V$ is less than or equal to $r_{\alpha \log n/n}$, then $(u, v) \in E$. Similarly, any pair of nodes $u, v \in V$ such that $d(u, v) > 2r_{\alpha \log n/n}$, then $(u, v) \notin E$. However, if $r_{\alpha \log n/n} < d(u, v) \leq 2r_{\alpha \log n/n}$, the pair $(u, v) \in H_g$ only if $\exists w \in V$ such that $d(u, w) \leq r_{\alpha \log n/n}$ and $d(v, w) \leq r_{\alpha \log n/n}$. This shows that the blocking graph H_g is a subgraph of a random geometric graph where a pair of vertices (u, v) is connected only if the distance $d(u, v) \leq 2r_{\alpha \log n/n}$ is connected. \square

This means that if G_t has suboptimal recall then P_g also has poor recall and hence, we analyze the recall of G_t with $r = 2r_{\alpha \log n/n}$. Lemma 3 shows that the blocking graph will achieve recall = 1 only if it contains a spanning tree of each cluster. Hence, we analyze the formation of spanning trees in $G'_t = G_t(V, E \cap E^+)$ that refers to G_t restricted to matching edges. We show the following result,

Lemma 6. *The graph G_t restricted to matching edges in the ground truth, E^+ splits a cluster C , where $|C| < n/\alpha$ into multiple components.*

Proof. Using the connectivity result from [162], a random geometric graph G_t of n nodes is disconnected if the expected degree of the nodes is $< \log n$. Additionally, it splits the graph G_t into many smaller clusters. Therefore, a cluster $C \in V$ is disconnected in $G'_t = G_t(V, E \cap E^+)$ if the degree of each vertex is $< \log |C|$.

The expected degree of a record $u \in C$, restricted to G'_t is $O(|C|(\frac{\alpha \log n}{n})) < \log n$ if $|C| < \frac{n}{\alpha}$. Hence, the expected degree of each node within a cluster C is $o(\log |C|)$, leading to formation of disconnected components within C . \square

Theorem 4. *A blocking graph $P_g(V, E)$, generated according to the probabilistic block model has recall < 1 unless all clusters have size $\Theta(n)$ assuming α is a constant.*

Proof. Lemma 6 shows that the cluster C of size $< n/\alpha$ is split into various disconnected components when restricted to matching edges. Hence, the blocking graph P_g does not form a spanning tree of C and will have recall less than 1. Since the cluster C is broken into many small clusters, the drop in recall is also significant. \square

This analysis exposes the lack of robustness of performing blocking without feedback.

4.6.2 Pair Recall with Feedback

This section analyzes the pair recall of blocking when employed with **pBlocking**. For this analysis, we consider the noisy edge similarity model $p_m(u, v)$ that builds on the edge noise model studied in prior work on ER [83].

Definition 10 (Noisy edge model). *Noisy edge model defines the similarity of a pair of records with parameters $\theta \in (0, 1)$, $\beta = \Theta(\log n)$ and $\beta' = \Theta(\log n)$. A matching edge $(u, v) \in E^+$ has a similarity distributed uniformly at random within $[\theta, 1]$ with probability $1 - \frac{\beta}{n}$ and remaining edges are distributed uniformly within $[0, \theta)$. A non-matching edge has similar distribution on similarity values with β' instead of β .*

When $\beta \ll \beta'$, the matching probability score of a block with a higher fraction of matching edges is much higher than the one with fewer matching edges and **pBlocking** algorithm will consider blocks in the correct ordering even in the absence of feedback. However, it is most challenging when non-matching edges are generated with a distribution similar to matching edges, that is β and β' are close. We define a random variable $X(u, v)$ to refer to the edge similarity distributed according to the noisy edge model. Following this notion, let μ_g and μ_r denote the expected similarity of a matching and non-matching edge respectively.

$$\mu_g = (1 - \beta/n) \frac{1 + \theta}{2} + \frac{\beta \theta}{n 2}$$

and μ_r has the same value with β' instead of β .

We show that the feedback-based block score initialized with TF-IDF weights is able to achieve perfect recall with feedback of $\Theta(n \log^2 n)$ pairs assuming that the ER phase makes no mistakes on the pairs that it processes, helping to ensure the correctness of partially inferred entities. Additionally, the feedback from the ER phase is distributed randomly across edges within a block. We also discuss the extension when feedback is biased towards pairs from large entity clusters and high similarity pairs. In those scenarios, **pBlocking**'s scoring mechanism converges quicker leveraging the larger feedback due to transitivity.

To prove the convergence, we first estimate the lower and upper bound of matching probability scores of a block B in the presence of feedback and show that feedback of $\Theta(\log^2 n)$ is enough to rank blocks with larger fraction of matching pairs higher than the blocks with fewer matching pairs. Our analysis first considers the blocks containing more than $\gamma \log n$ records (where γ is a large constant say $72/\theta$) and we analyze the smaller blocks separately.

Convergence for large blocks. First, we evaluate the converged block scores with feedback F and evaluate the condition that the block scores are in the correct order.

Lemma 7. *For all blocks B , with more than $\gamma \log n$ records, the matching probability score of B , $p(B)$ after feedback of F randomly chosen pairs is at most $(1 - \alpha)|F|/\binom{\gamma \log n}{2} + 1.5p'(1 - |F|/\binom{\gamma \log n}{2})$ with a probability of $1 - 1/n^3$, where α is the fraction of non-matching pairs in B , γ is a constant and $p' = \mu_g(1 - \alpha) + \mu_r\alpha$.*

Similarly, we prove a lower bound on block score.

Lemma 8. *For all blocks B with $|B| \geq \gamma \log n$, the matching probability score after a feedback $F \leq \binom{\gamma \log n}{2}$ record pairs in B is at least $(1 - \alpha)|F|/\binom{\gamma \log n}{2} + 0.5p'(1 - |F|/\binom{\gamma \log n}{2})$ with a probability of $1 - 1/n^3$, where $p' = \mu_g(1 - \alpha) + \mu_r\alpha$ and γ is a constant.*

Now, we analyze different scenarios of edge noise to understand the trade-off between required feedback and noise.

Lemma 9. *For every pair of blocks, B_c, B_d with more than $\gamma \log n$ records, the matching probability score estimate of B_c with $1 - \alpha$ fraction of matching edges is greater than the score of B_d with $1 - \beta$ (with $\alpha < \beta$) fraction of matching edges with a probability of $1 - \frac{2}{n}$ if $((1 - \alpha)\mu_g + \alpha\mu_r) > 3((1 - \beta)\mu_g + \beta\mu_r)$ even in the absence of feedback.*

Proof. Using Lemma 7 and 8, we can evaluate the condition that $\text{score}(B_c) > \text{score}(B_d)$ with a probability of $1 - \frac{2}{n^3}$, in the absence of feedback. In order to guarantee this for all blocks, we perform a union bound over $\Theta(n^2)$ pairs of blocks, guaranteeing the success rate to $1 - o(1)$. \square

The previous lemma shows a scenario where the noise is not high and the prior based estimation of matching probability scores give a correct ordering of blocks. Now,

we consider the more challenging noisy scenario and show that $\Theta(\log^2 n)$ feedback per block is enough for correct ordering.

Lemma 10. *For every pairs of blocks, B_c, B_d with more than $\gamma \log n$ records, the matching probability score estimate of B_c with $1 - \alpha$ fraction of matching edges is greater than the score of B_d with $1 - \beta$ (where $\alpha < \beta$) fraction of matching edges with a probability of $1 - \frac{2}{n}$ whenever the ER phase provides overall feedback of $\Theta(n \log^2 n)$ randomly chosen edges.*

Proof. Using Lemma 8, $\text{score}(B_c) \geq |F|/(\gamma \log n)(1 - \alpha) + 0.5(\mu_g(1 - \alpha) + \alpha\mu_r)(1 - |F|/(\gamma \log n))$ and using Lemma 7, $\text{score}(B_d) \leq |F|/(\gamma \log n)(1 - \beta) + 1.5(\mu_g(1 - \beta) + \beta\mu_r)(1 - |F|/(\gamma \log n))$ with a probability of $1 - \frac{2}{n^3}$. Hence, $\text{score}(B_c) > \text{score}(B_d)$ holds if $F = c \log^2 n$, where c is a large constant. With a union bound over $\binom{n}{2}$ pairs of blocks, the score of any block B_c (with higher fraction of matches) is higher than that of any block B_d (with lower fraction of matches) with a probability of $1 - \frac{2}{n}$. The total feedback to ensure $\Theta(\log^2 n)$ feedback on each block is $\Theta(n \log^2 n)$ as we consider $\Theta(n)$ blocks for scoring. \square

Similar lemmas hold for the uniformity score calculation.

Convergence for small blocks. The above analysis does not extend to blocks of size less than $\gamma \log n$. However, all these blocks are ranked higher than the large blocks by TF-IDF. Hence, when **pBlocking** is initialized, the initial set of candidates generated will consider all these blocks before any of the larger blocks. In the worst case, there can be δn such blocks, for some constant δ because our approach constructs a constant number of blocks per record (say δ). Thus, the maximum number of candidates considered from small blocks is $\delta n \binom{\gamma \log n}{2}$ and all these candidates are considered in the first iteration of **pBlocking**. Following the discussion on small and large blocks, we prove the main result of the convergence of **pBlocking**.

Theorem 5. *pBlocking pipeline achieves perfect recall with a feedback of $O(n \log^2 n)$ spread randomly across blocks.*

Proof. For blocks with more than $\gamma \log n$ records, Lemmas 9 and 10 show that a block with higher fraction of matching pairs is ranked higher than a block with fewer matching pairs, if provided with a feedback of $\Theta(n \log^2 n)$. Blocks with less than $\gamma \log n$ records have not been considered above but in the worst case, these blocks generate $O(n \log^2 n)$ candidates as the maximum number of blocks considered is $\Theta(n)$. This ensures that a feedback of $\Theta(n \log^2 n)$ is sufficient to ensure the stated result. \square

Table 4.2: Number of nodes n (i.e., records), number of clusters k (i.e., entities), size of the largest cluster $|C_1|$, the total number of matches in the data set $|E^+|$ and the reference to the paper where they appeared first.

| dataset | n | | k | $ C_1 $ | $ E^+ $ | ref. | description |
|------------------|-------|------|-------|---------|---------|-------|---|
| songs | 1M | 1M | 0.99M | 2 | 146K | [69] | Self-join of songs with very few matches. |
| citations | 1.8M | 2.5M | 3.8M | 2 | 558K | [69] | Bibliographic records from DBLP and CiteSeer. |
| products | 2554 | 22K | 23.5K | 2 | 1154 | [103] | A collection of products from retail companies website. |
| cora | 1.9K | | 191 | 236 | 62.9K | [146] | Title, author, venue, and date of scientific papers. |
| cars | 16.5K | | 48 | 1799 | 5.9M | [134] | Descriptions of cars with make and model. |
| camera | 29.7K | | 26K | 91 | 102K | [1] | A collection of cameras from over 25 retail companies. |
| febr11 | 100M | | 99.5M | 2 | 500K | [61] | A collection of hospital patients data, including name, address and phone number, that we produced using the dataset generator of the Febrl system. |
| febr12 | 100M | | 50M | 100 | 2500M | [61] | |

Discussion. Lemma 10 considers the convergence of block scores when the feedback is provided randomly over $\Theta(\log^2 n)$ edges within a block. If the feedback is biased towards $\Theta(\log^2 n)$ non-matching edges, the scores of noisier blocks will drop quicker and pBlocking will converge faster. Similarly, if the ER algorithm queries pairs with higher similarity (e.g. edge ordering [197]) or grows clusters by processing nodes (e.g. node ordering [194]), providing larger feedback due to transitivity, this will only facilitate the growth (reduction) in score of blocks with higher (lower) fraction of matching pairs leading to faster convergence.

Finally, for the presented analysis, we assumed that oracle answers are correct. Nonetheless, (i) for small amount of oracle errors ($\sim 5\%$), we can leverage methods such as [87, 191] to correct them, and (ii) in more challenging applications with up

to 20% erroneous answers, we show experimentally (see Section 4.7) that **pBlocking** keeps converging, only at a slightly slower rate and demonstrates robustness.

4.7 Experiments

This section empirically evaluates the ability of **pBlocking** to boost the efficiency and effectiveness of blocking and thus to improve the performance of ER. We also demonstrate the fast convergence of **pBlocking** thus confirming our theoretical analysis in Section 4.6, and the robustness of **pBlocking** in different scenarios, including errors in ER results. This section is structured as follows.

- *Section 4.7.1.* We compare the efficiency and effectiveness of **pBlocking** to prior work showing higher pair recall and faster running time in all the data sets.
- *Section 4.7.2.* We analyze **pBlocking** when used in conjunction with different ER methods showing higher *F-score* (up to 60%) irrespective of the method of choice.
- *Section 4.7.3.* We study the dynamic performance of **pBlocking** and show its ability to converge monotonically to high effectiveness without compromising on efficiency in different scenarios including errors in ER results.

Before showing results we describe our experimental setup and the methods considered in our experiments.

Experimental set-up. We implemented the algorithms in Java and machine learning tools in Python. The code was run on a server with 500GB RAM and 64 cores. We consider six real-world data sets (see Table 4.2) of various sizes and diverse cluster distributions. All the datasets are publicly available and come with their own manually curated ground truth. We use publicly available pre-trained deep learning

models⁴ to generate text descriptions of the image data (`cars`). `febr11` and `febr12` were constructed with uniform and zipfian distributions of cluster sizes. For more details about these parameters, please refer to [61]. For implementing the hierarchy we observed that we can trim at a depth of 10 without any significant drop in the performance. The implementation of blocking strategies is adapted from [159]⁵.

Blocking methods. We consider 10 strategies for the blocking sub-tasks described in Section 4.2 and combine such strategies into 20 different pipelines. We study such pipelines with and without our `pBlocking` approach on top.

BB) We consider five methods for Block Building (*BB*) and follow the suggestions of [160] for their configuration. Standard blocking [155] (*StB1*) generates a new block for each text token in the dataset. Q-grams blocking [106] (*QGBL*) generates a new block for each 3-gram of characters. Sorted neighborhood [117] (*SoNE*) sorts the tokens for each attribute and generates a new block for every sliding window of size 3 over these sort orders. Dynamic Blocking [148] (*DyB1*) generates a new block for each token and constructs a hierarchy containing intersections of these large blocks. All blocks of size more than 20 are considered for hierarchy construction⁶ Canopy clustering [147] (*CaC1*) generates a new block for each cluster of high similarity records (calculated as unweighted Jaccard similarity). We construct multiple instances of canopies (blocks), one based on the similarity of each attribute of record pairs and one based on all attributes together.

BC) We consider 2 traditional block scoring methods for Block Cleaning (*BC*), dubbed *TF-IDF* [173] and uniform scoring (*Unif*). For comparison purposes,

⁴<https://cloud.google.com/vision,>
visual-recognition/

<https://www.ibm.com/watson/services/>

⁵<http://sourceforge.net/projects/erframework/>

⁶This threshold on block size was shown to have best blocking quality in [148].

we process blocks in non-increasing score order until the number of intra-block pairs equals a parameter M and then prune the remaining blocks. We set default M to 500 million for `febrl` and 10 million for all other datasets⁷.

CC) We consider 2 popular methods for Comparison Cleaning (**CC**), dubbed meta-blocking [154] (**MB**) and BLOSS [67], and follow the suggestions of [154] for their configuration. Weights of record pairs are set to their Jaccard similarity weighted with the block scores from the **BC** sub-task. We consider the top 100 high-weight pairs for each record and prune the remaining record pairs.

We recall that variants of our approach are denoted as **pBlocking**(,,) while traditional blocking pipelines without feedback are denoted as **B**(,,) where the parameters correspond to techniques for **BB**, **BC** and **CC** sub-tasks, respectively. Default methods are **StB1** for **BB**, **TF-IDF** for **BC** and **MB** for **CC**. Default ϕ for **pBlocking** is 0.01.

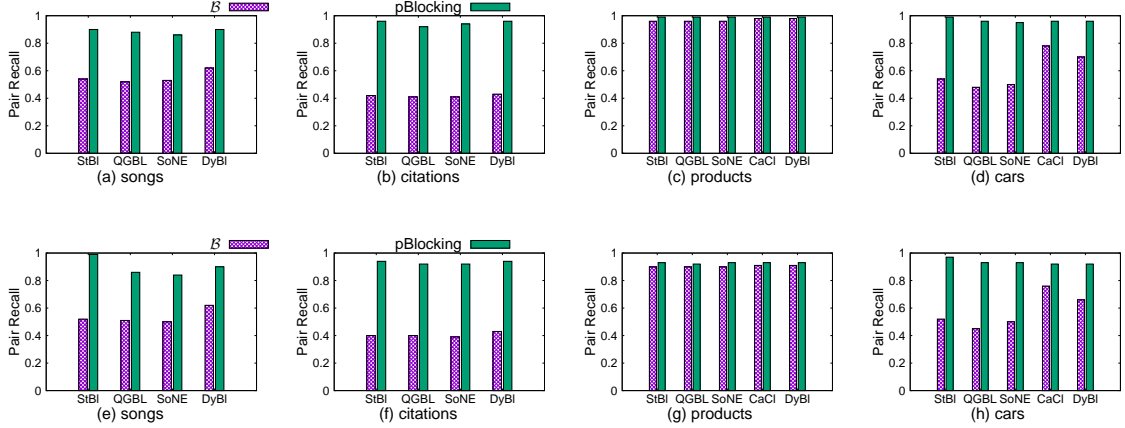


Figure 4.3: Pair recall of **B** and **pBlocking** with TF-IDF for **BC** and varying **BB** and **CC**. (a-d) use **MB** and (e-h) use **BLOSS** for **CC**. **CaCl** did not finish within 24 hrs on **songs** and **citations** data set.

Pair matching and Clustering methods. We consider the following 3 strategies that leverage the notion of an *oracle* to answer pairwise queries of the form “does u

⁷We note that setting a score threshold rather than a limit on the number of pairs would not take into account different scores distributions fairly.

match with $v?$ ” (a) **Edge** [197] with default parameter setting. (b) **Eager** [87], the state-of-the-art technique to solve ER in the presence of erroneous oracle answers. (c) **Node** is the ER mechanism derived from [194] and was proposed as an improvement over **Edge**. The **Eager** algorithm handles noise for data sets with matching pairs much larger than n and performs similar to **Edge** for data sets that have fewer matching pairs [83], so we use it as default. Each of these techniques recalculate the prioritization of the updated set of blocked pairs in each feedback round. We implement the abstract oracle tool with a classifier using scikit learn⁸ in Python. We consider two variants, Random forests (default) and a Neural Network. The random forest classifier is trained with default settings of scikit learn. The neural network is implemented with a 3-layer convolutional neural network followed by two fully connected layers. We used word2vec word-embeddings for each token in the records. In structured data sets, we extract similarity features for each attribute as in [69]. For **cars** we use the text descriptions to calculate text-based features along with image-based features. Given the unstructured nature of text descriptions for some data sets we extracted POS tags using Spacy⁹. All the considered classifiers are trained offline with less than 1,000 labelled pairs, containing a similar amount of matching and non-matching pairs. These labelled record pairs are the ones provided by the respective source for **citations**, **songs**, **products** and **camera** (the papers mentioned in Table 4.2, column “ref.”). For **cars** and **cora** we perform active learning (following the guidelines of [69]) to identify a small set of labelled examples for training, which are excluded from the evaluation of blocking quality.

⁸<https://scikit-learn.org/stable/>

⁹<https://spacy.io/>

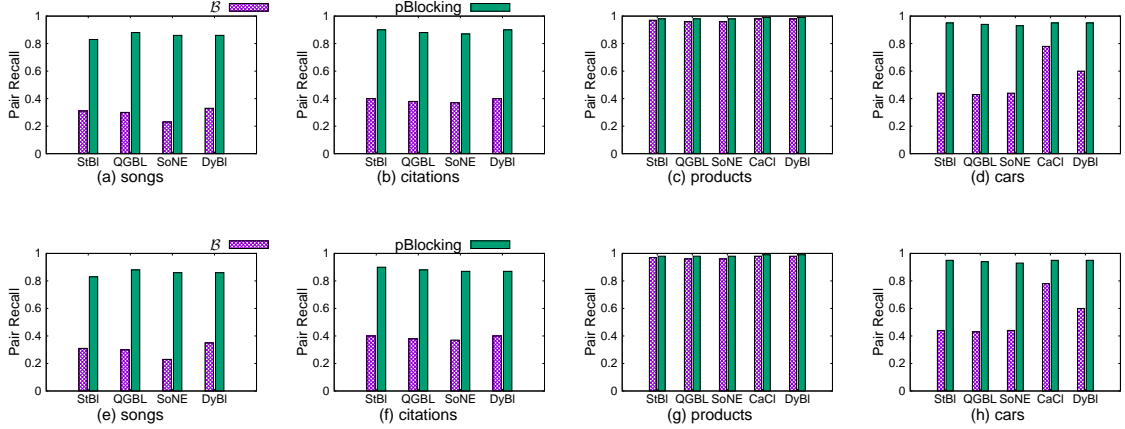


Figure 4.4: Pair recall of \mathcal{B} and pBlocking with Unif for \mathcal{BC} and varying \mathcal{BB} and \mathcal{CC} . (a-d) use MB and (e-h) use BLOSS for \mathcal{CC} . CaCl did not finish within 24 hrs on songs and citations data set.

4.7.1 Benefits of Progressive Blocking

In this experiment we evaluate the empirical benefit of pBlocking compared to previous blocking strategies.

Blocking effectiveness. Figures 4.3 and 4.5 compare the Pair Recall (PR) of pBlocking and of a traditional blocking pipeline \mathcal{B} for different choices of the block building and comparison cleaning techniques. We use default block cleaning technique with TF-IDF and default M value. pBlocking achieves more than 0.90 recall for all data sets and with all the block building strategies, demonstrating its robustness to different cluster distributions and properties of the data. Conversely, most of the considered block building strategies (StBl, QGBL, SoNE and DyBl) have significantly lower recall even when used together with BLOSS for selecting the pairs wisely. QGBL and SoNE help to improve recall in data sets with spelling errors but due to very few spelling mistakes in our data sets, StBl has slightly higher recall. DyBl creates blocks of moderate size that are expected to capture matching pairs. This technique performs better than StBl but the constructed smaller blocks contain a lot of non-matching pairs that affect pair recall.

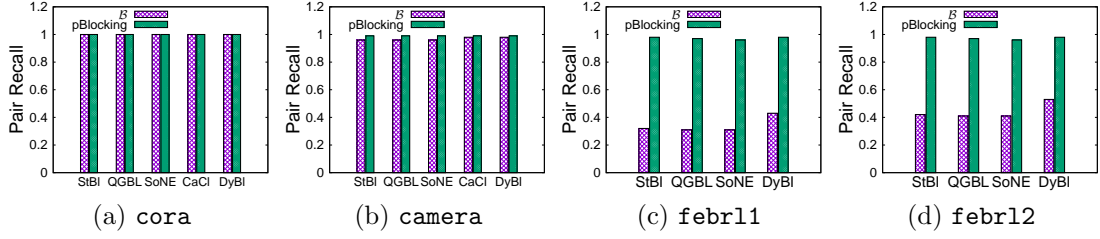


Figure 4.5: Pair recall of \mathcal{B} and pBlocking with varying \mathcal{BB} , TF-IDF for \mathcal{BC} and MB for \mathcal{CC} . CaCI did not finish within 24 hrs on febr1 datasets. We observed similar results with BLOSS for \mathcal{CC} .

In terms of the data sets, the no-feedback blocking approach \mathcal{B} has varied behavior. **products** and **camera** yield the best performance due to the presence of relatively cleaner blocks that help to easily identify matching pairs even without feedback. **songs** and **citations** have higher noise in records and **cars** has a skewed distribution of clusters thereby making it harder for previous techniques. Even though **cars** and **febr12** have low noise, large blocks that contain the majority of the records referring to same entity are partitioned by DyBI and ranked lower by TF-IDF weighting of blocks. Across all datasets and blocking strategies, the comparison between pBlocking and \mathcal{B} is statistically significant ($p < 0.01$) using the student’s paired t-test. For this analysis, we do not consider **cora** (the smallest data set) as it has less than 2M pairs and hence, all techniques achieve perfect recall.

Figure 4.4 performs the same comparison with the pipelines initialized using **Unif** weights in place of TF-IDF. Since all blocks are assigned equal weight, we consider the block cleaning threshold of 100 along with default value of M. pBlocking performs substantially better than \mathcal{B} for different settings of block building techniques across various datasets. With comparison to TF-IDF weighting scheme, **Unif** performs slightly worse but the difference is not substantial. The no-feedback pipeline \mathcal{B} has varied performance across different data sets with the best performance on **products**

Table 4.3: Running time comparison of \mathcal{B} and **pBlocking** with **StB1** and **DyB1** for \mathcal{BB} , TF-IDF for \mathcal{BC} and MB for \mathcal{CC} . The ‘blocking’ column denotes the time taken to perform blocking and ‘ER’ denotes the time taken to identify matches over blocked pairs.

| Dataset | StB1 | | | | | | | |
|------------------|---------------------------|--------------|--------------|--------------------------------|---------------|---------------|-------------------|---------------|
| | 0.95 Pair recall | | | | | | Time budget: 1 hr | |
| | pBlocking(StB1,TF-IDF,MB) | | | \mathcal{B} (StB1,TF-IDF,MB) | | | Pair Recall | |
| | Blocking | ER | Total | Blocking | ER | Total | pBlocking | \mathcal{B} |
| songs | 4.5 min | 24.5 min | 29 min | 3min | 180 min | 3hrs 3min | 0.96 | 0.78 |
| citations | 12 min | 43 min | 55 min | Did not finish in 24 hrs | | | 0.97 | 0.64 |
| cars | 3hr 20min | 50 min | 4hr 10min | 25 min | 11hr 30 min | 11hr 55min | 0.78 | 0.54 |
| febr11 | 55 min | 2hr 35 min | 3hr 30min | Did not finish in 24 hrs | | | 0.64 | 0.21 |
| febr12 | 95 min | 4 hr 15 min | 5hr 50min | Did not finish in 24 hrs | | | 0.34 | 0.15 |
| products | 35 sec | 5min 55 sec | 6min 30sec | 27 sec | 5 min 46 sec | 6min 13sec | 0.99 | 0.98 |
| camera | 42 sec | 11min 38 sec | 12min 20sec | 33 sec | 12 min 30 sec | 13min 3sec | 0.97 | 0.96 |
| cora | 30 sec | 4 min 50 sec | 5min 20 sec | 27 sec | 4min 48sec | 5min 15sec | 1 | 1 |
| | DyB1 | | | | | | | |
| | pBlocking(DyB1,TF-IDF,MB) | | | \mathcal{B} (DyB1,TF-IDF,MB) | | | Pair Recall | |
| | Blocking | ER | Total | Blocking | ER | Total | pBlocking | \mathcal{B} |
| | Blocking | ER | Total | Blocking | ER | Total | pBlocking | \mathcal{B} |
| songs | 6.5 min | 24.5 min | 31 min | 5 min | 180 min | 3hrs 5min | 0.96 | 0.84 |
| citations | 15 min | 43 min | 58 min | 15 min | 10 hrs | 10 hrs 15 min | 0.97 | 0.67 |
| cars | 3hr 30min | 50 min | 4hr 20min | 30 min | 11hr 25 min | 11hr 55min | 0.78 | 0.64 |
| febr11 | 58 min | 2hr 35 min | 3hr 33min | 1hr 7min | 15hr | 16hr 7min | 0.64 | 0.21 |
| febr12 | 100 min | 4 hr 15 min | 5hr 55min | 1hr 7min | 15hr 20min | 16 hr 27min | 0.34 | 0.15 |
| products | 38 sec | 5min 55 sec | 6min 33sec | 32 sec | 5 min 46 sec | 6min 18sec | 0.99 | 0.98 |
| camera | 45 sec | 11min 38 sec | 12min 23 sec | 37 sec | 12 min 30 sec | 13min 7 sec | 0.97 | 0.96 |
| cora | 36 sec | 4 min 50 sec | 5min 26 sec | 32 sec | 4min 48sec | 5min 20sec | 1 | 1 |

and **cora** while poorest performance on **citations** and **songs**. We observed similar behavior for **cora**, **camera** and **febr1** datasets.

This experiment demonstrates that **pBlocking** helps to improve the pair recall of all blocking techniques (for the same set of parameters) and datasets. Note that increasing the block cleaning threshold M improves pair recall further but worsens the efficiency of the pipeline. As an example, [157] enumerates more than 10^{10} candidates for million scale datasets (where maximum possible candidates $\approx 10^{12}$), as opposed to 10M candidates in Figure 4.3. As reported in [157], the pipeline with 10^{10} candidates requires more than 14.5 hours per dataset to achieve 0.82. For a fair comparison of blocking efficiency, we compare the pair recall within a time budget of 1hr and time taken to achieve 0.95 pair recall in ‘Blocking efficiency’ paragraph and Table 4.3.

Multiple blocking methods. Figure 4.6 demonstrates the effectiveness of considering feedback in pipelines where multiple block building procedures are used to

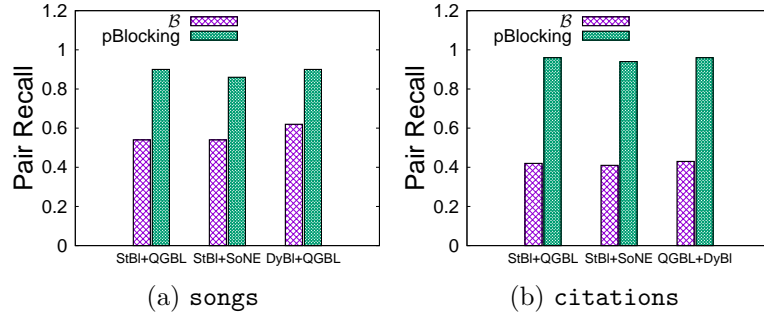


Figure 4.6: Pair recall of \mathcal{B} and pBlocking with combination of two block building strategies and TF-IDF for \mathcal{BC} and MB for \mathcal{CC} .

initialize the pipeline. \mathcal{B} has lower than 0.6 pair recall even when we consider different combinations of block building strategies. Using DyBI along with QGBL achieves the highest pair recall among the considered combinations, due to the ability of DyBI to construct smaller blocks that capture matching record pairs. However, pBlocking achieves more than 0.90 pair recall for all combinations of block building strategies.

Blocking efficiency. In this experiment, we consider two different settings to compare (i) the time required to achieve more than 0.95 pair recall (ii) the pair recall when the pipeline is allowed to run for a fixed amount of time (1 hour). We run each technique for various values of M and choose the best value that satisfies the required constraints. In the case of a fixed budget of running time = 1hour, we run pBlocking’s feedback loop for the most iterations that allow the pipeline to process all records in the required time limit.

Table 4.3 compares the total time required to achieve 0.95 pair recall for each dataset (‘Blocking’ column denotes the time taken to perform blocking and ‘ER’ column denotes the time taken in pair matching and clustering phases of the pipeline). The time taken by the blocking component of the pipeline is higher for pBlocking as compared to \mathcal{B} due to the extra effort spent in incorporating feedback, constructing new blocks and ranking based on their quality. However, pBlocking’s blocking component is highly effective and substantially reduces the time taken to process the

candidates generated by the blocking phase to identify matches. Overall, **pBlocking** provides more than 3 times reduction in running time for most large scale datasets in this setting. In terms of total number of pairs enumerated, **pBlocking** considers around $M=10$ million to achieve 0.95 recall for **citations** as opposed to more than 200 million for **B**. We observed similar results for other block building (**SoNE**, **QGBL**, **CaCl** and **DyBl**) and cleaning strategies with a difference that **DyBl** runs for **febrl** datasets in around 16 hrs.

The last two columns of Table 4.3 compare the pair recall of the generated candidates when the technique is allowed to run for 1 hour. **pBlocking** achieves better pair recall as compared to **B** across all datasets. The gain in recall is higher for larger datasets. The performance of **pBlocking** for **cars** is lower than that of **pBlocking** in Figure 4.3 because the feedback loop does not converge completely in 1hr. The pipeline runs for 8 rounds of feedback in this duration. This is consistent with the performance of **pBlocking** in Figure 4.10a, where the feedback is turned off after 10 iterations. The performance of **pBlocking** and **B** is similar for small datasets of low noise like **products**, **cora** and **camera** as opposed to **songs**, **citations** and **cars**.

Scalability. Figure 4.7 compares the time taken by **pBlocking** on different sub-samples of **febrl** dataset to reach 0.95 pair recall¹⁰. The time taken by **pBlocking** increases linearly with increase in dataset size and the pipeline identifies a majority of the matching records in less than 6 hrs. Since the number of matching pairs in the ground truth increases linearly with dataset size and low noise in records, the size of the blocking graph and the time taken by the pair matching and clustering components scales linearly. The time taken by **BLOSS** is slightly lower than the time taken by **MB** because **BLOSS** processes the meta-blocking graph to further prune out non-matching record pairs. This optimization increases the time taken by the blocking

¹⁰Each sub-sample was generated by using Febrl dataset generator with a smaller value of n , the number of records.

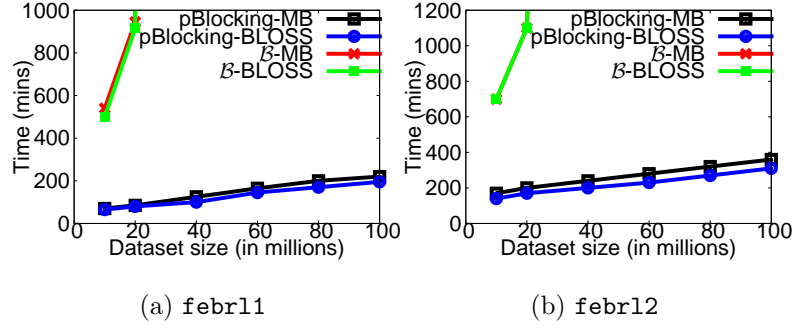


Figure 4.7: Time taken by pBlocking and \mathcal{B} with StB1 for \mathcal{BB} and TF-IDF for \mathcal{BC} for varying dataset size.

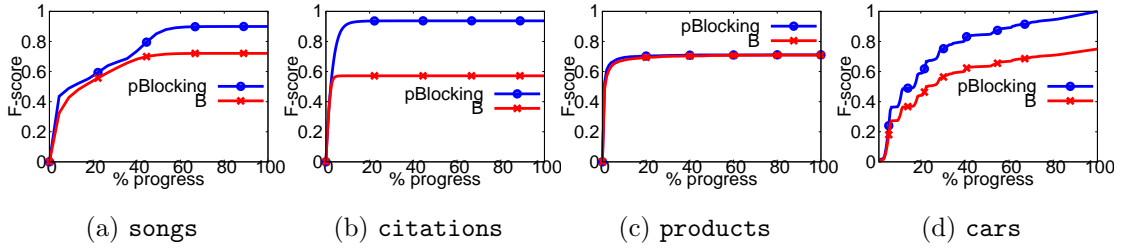


Figure 4.8: Comparison of F-score of $\mathcal{B}(\text{DyB1}, \text{TF-IDF}, \text{MB})$ and pBlocking(DyB1,TF-IDF,MB) with respect to ER progress.

phase of the pipeline but significantly reduces the number of pairs compared by the pair matching phase, thereby improving the overall efficiency. On the other hand, \mathcal{B} does not run for more than 20M records in less than 24 hrs. This experiment demonstrates scalability of pBlocking to achieve high recall over large scale datasets in a reasonable time.

Progressive behavior. Figure 4.8 compares the F-score of different pipelines with respect to the progress of the ER phase. F-score of the entities identified by pBlocking grows faster than \mathcal{B} , demonstrating its effectiveness to maintain better progressive behavior. pBlocking achieves more than 0.9 F-score across all datasets but \mathcal{B} converges at a lower F-score due to the loss in pair recall of the blocking phase. In terms of datasets, pBlocking and \mathcal{B} achieve similar progressive F-score throughout

Table 4.4: (a) Pair recall of **pBlocking** on varying ER strategies. (b) Comparison of the final F-score of the **Eager** method. The blocking graph is computed with **pBlocking**(StB1, TF-IDF, MB) and \mathcal{B} (StB1, TF-IDF, MB) (both with default settings).

| (a) | | | | | (b) | | |
|-----------|---------------|-----------|------|-------|-----------|---------------|-------------|
| Dataset | \mathcal{B} | pBlocking | | | Dataset | \mathcal{B} | pBlocking |
| | | Edge | Node | Eager | | | |
| songs | 0.53 | 0.9 | 0.9 | 0.9 | songs | 0.65 | 0.92 |
| citations | 0.42 | 0.90 | 0.87 | 0.95 | citations | 0.56 | 0.92 |
| cars | 0.54 | 0.98 | 0.99 | 0.98 | cars | 0.64 | 0.94 |
| febr11 | 0.32 | 0.98 | 0.98 | 0.98 | febr11 | 0.48 | 0.98 |
| febr12 | 0.41 | 0.97 | 0.99 | 0.98 | febr12 | 0.58 | 0.98 |
| products | 0.95 | 0.98 | 0.98 | 0.98 | products | 0.71 | 0.72 |
| camera | 0.92 | 0.97 | 0.97 | 0.97 | camera | 0.92 | 0.95 |
| cora | 1 | 1 | 1 | 1 | cora | 0.99 | 0.99 |

the ER progress for **products** dataset. **products** has around 0.72 final F-score due to low precision of pair matching and clustering phase. We observed similar behavior for other blocking pipelines.

4.7.2 Robustness of Progressive Blocking

In this section, we evaluate the performance of **pBlocking** with varying strategies for pair matching and clustering in Algorithm 9 (referred to as W in the pseudo-code). For this analysis, we use the default setting for M as in Figure 4.3.

Varying ER methods. We recall that **pBlocking** can be used in conjunction with a variety of techniques for pair matching and clustering. Table 4.4a compares the Pair Recall of the blocking graph, when using the different progressive ER methods. The final Pair Recall of **pBlocking** is more than 0.90 in all data sets and matching algorithms except **citations** for node ER and more than 0.85 in all cases. This observation confirms our theoretical analysis in Section 6.5, demonstrating that the feedback loop can improve the blocking, irrespective of the ER algorithm under consideration (which is a desirable property for a blocking algorithm). The above comparison of ER performance considers the algorithms with a default choice of Random Forest classifier as the oracle. We observed that the feedback from the ER phase when using a Neural Network classifier contains slightly more errors but the blocking

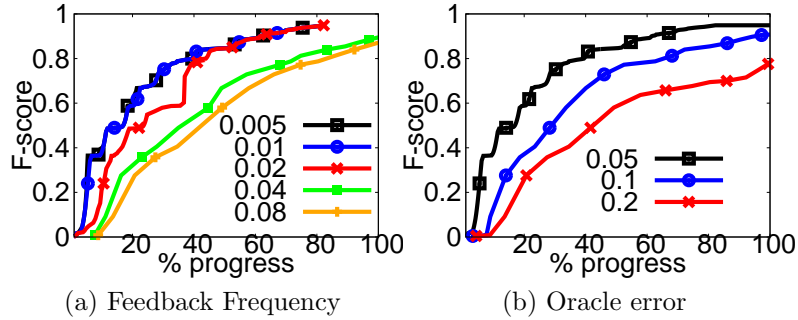


Figure 4.9: Progressive behavior of pBlocking with varying feedback frequency and errors in the feedback (cars).

phase with pBlocking shows similar recall. We provide more discussion on ER errors in Section 4.7.3.

Benefit on the final ER result. Table 4.4b compares the F-score of the final ER results when blocking is performed with and without pBlocking. In this experiment we use the state-of-the-art algorithm, **Eager** as the pair matching algorithm with default parameter values. Final F-score achieved with feedback is more than 0.9 for all data sets except **products**. For **songs**, **citations** and **cars** the F-score of pBlocking is 1.5 times more than that of traditional blocking pipeline without feedback, thus demonstrating the effects of better effectiveness and efficiency of blocking.

4.7.3 Progressive Behavior

This section studies the performance of pBlocking dynamically, in terms of (i) effect of feedback frequency ϕ , (ii) effect of error on convergence, and (iii) convergence of the blocking result in the maximum number of rounds.

Feedback frequency. The ϕ parameter represents the fraction of newly processed record pairs after which feedback is sent from the partial ER results back to the blocking phase. Therefore, the parameter ϕ can control the maximum number of rounds of pBlocking and how often the blocking graph is updated. In order to describe the effect of varying ϕ , Figure 4.9a shows the F-score of ER results as a function

of the percentage of rounds completed, that we refer to as the *blocking progress*. In the figure, different curves correspond to different feedback frequencies, including the default one (in blue). This plot shows that by updating the blocking graph more frequently (and thus increasing the number of rounds), the F-score increases faster when ϕ is reduced from 0.08 to 0.01. The plot also shows that the F-score corresponding to smaller values of ϕ (up to 0.01) is consistently higher or equal as compared to the F-score corresponding to larger values of ϕ . Given that the running time of the pipeline increases with more frequent updates (smaller values of ϕ), there appears to be limited value in decreasing ϕ below 0.01, thus justifying our choice for its default setting.

Effect of ER errors. As in the previous experiment, Figure 4.9b shows the effect of synthetic error in the ER results by varying the fraction of erroneous oracle answers. To this end, we corrupted the oracle answers randomly so as to get the desired amount of noise. We note that even when 1 out of 5 answers are wrong, the final F-score is almost 0.8, growing monotonically from the beginning to the end at the cost of a few extra pairs compared. **pBlocking** converges slower with higher error but the error does not accumulate and it performs much better than any other baseline. Additionally, we observed that even with 20% error, the pair recall of **pBlocking** is as high as 0.98 even though the F-score is close to 0.8 due to mistakes made by pair matching and clustering phase. This confirms that **pBlocking** is robust to errors in ER results and maintains high effectiveness to produce ER results with high F-score.

Score Convergence. Figure 4.10a compares the Pair Recall (PR) of the blocking phase of **pBlocking**(StB1,TF-IDF,MB) after every round of feedback with the recall of \mathcal{B} (StB1,TF-IDF,MB). Both \mathcal{B} and **pBlocking** start with PR value close to 0.52 and **pBlocking** consistently improves with more feedback achieving PR close to 0.9 in less than 18 rounds. This shows the convergence of **pBlocking**'s score assignment strategy to achieve high PR values even with minimal feedback. Figure 4.10b compares the

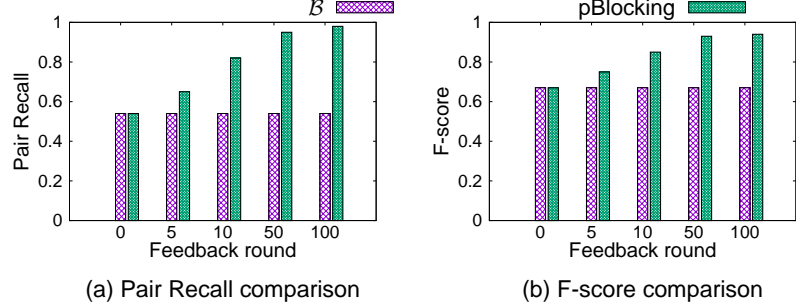


Figure 4.10: Effect of feedback loop in *cars* dataset.

final F-score achieved by our method if the feedback loop is stopped after a few rounds. It shows that pBlocking achieves more than 0.8 F-score even when stopped after 10 rounds of feedback. This experiment validates that the convergence of block scoring leads to the convergence of the entire ER workflow.

4.8 Related Work

We divide the related work into two parts: advanced blocking methods which we improve upon, and progressive ER methods which can be used to generate a limited amount of matching/non-matching pairs to send as a feedback to our blocking computation.

Advanced blocking methods. There are many blocking methods in the literature with different internal functionalities and solving different blocking sub-tasks. In this chapter, we considered four representative block building strategies, namely standard blocking [155], canopy clustering [147], sorted neighborhood [117] and q-grams blocking [106]. It is well-known that such techniques can yield a fairly dense blocking graph when used alone. We refer the reader to [159] for an extensive survey of various blocking techniques and their shortcomings. Such block building strategies can be used as the X method in our Algorithm 9.

One of the prior blocking techniques, Dynamic Blocking [148] considers conjunctions of large blocks to construct a hierarchy of smaller co-occurring sub-blocks. This

approach assumes a priori knowledge of the attributes that are used to whittle down oversized blocks to an acceptable size and was primarily designed for datasets with small clusters (e.g., of size 2), where smaller blocks are correlated with matching pairs. On the other hand, **pBlocking** uses the block scores as a guidance to construct the hierarchy and rank the blocks. Following the score based hierarchy construction procedure, **pBlocking** does not partition large blocks that contain a lot of matching pairs and partitions all blocks that contain fewer matching pairs irrespective of their size.

Recent works have proposed advanced methods that can be used in combination with the mentioned block building techniques by focusing on the comparison cleaning sub-task (thus improving on efficiency). The first technique in this space is *meta-blocking* [154]. Meta-blocking aims at extracting the most similar pairs of records by leveraging block-to-record relationships and can be very efficient in reducing the number of unnecessary pairs produced by traditional blocking techniques, but it is not always easy to configure. To this end, follow-up works such Blast [176] use “loose” schema information to distinguish promising pairs, while [38] and SNB [158] rely on a sample of labeled pairs for learning accurate blocking functions and classification models respectively. Finally, the most recent strategy **BLOSS** [67] uses *active learning* to select such a sample and configure the meta-blocking. Such meta-blocking works compute the blocking graph statically, prior to ER, and thus can be used as the *Z* method in our Algorithm 9. In Figure 4.3 we compare with classic meta-blocking and **BLOSS**, as the latter shows its superiority over Blast and SNB.

Progressive ER. Many applications need to resolve data sets efficiently but do not require the ER result to be complete. Recent literature described methods to compute the best possible partial solution. Such techniques include *pay-as-you-go* ER [202] that use “hints” on records that are likely to refer to the same entity and more generally *progressive* ER such as the schema-agnostic method in [177] and the

strategies in [22, 161] that consider a limit on the execution time. In our discussion, we considered oracle-based techniques, namely **Node** [194], **Edge** [197], and **Eager** [87]. Differently from other progressive techniques, oracle-based methods consider a limit on the number of pairs that are examined by the oracle for matching/non-matching response. Such techniques were originally designed for dealing with the crowd but they can also be used with a variety of classifiers due to their flexibility. All these techniques naturally work in combination with **pBlocking** by sending as feedback their partial results.

Other ER methods. In addition to the above methods, we mention works on ER architectures that can help users to debug and tune parameters for the different components of ER [103, 69, 132, 160]. Specifically, the approaches in [103, 69] show how to leverage the crowd in this setting. All of these techniques are orthogonal to the scope of our work and we do not consider them in our analysis. The previous work in [203] proposes to greedily merge records as they are matched by ER, while processing the blocks one at a time. Each merged record (containing tokens from the component records) is added to the unprocessed blocks, permitting its participation in the subsequent matching and merging by their iterative algorithm. Limitations of processing blocks one at a time has been shown in more recent blocking works [154].

4.9 Summary and Future Work

This chapter presents a new blocking algorithm, **pBlocking** that progressively updates the relative scores of blocks and constructs new blocks by leveraging a novel feedback mechanism from partial ER results. **pBlocking** boosts the effectiveness and efficiency of blocking across all data sets by initializing blocking with any of the standard techniques and then using new feedback-based methods for solving blocking sub-tasks in a data-driven way. To the best of our knowledge, **pBlocking** is the first

framework where blocking and pair matching components of ER help each other and produce high-quality results in a synergy.

The key takeaways from the chapter are summarized below.

- **pBlocking** improves Pair Recall irrespective of the technique used for block building, block cleaning or comparison cleaning, thus demonstrating its flexibility.
- Feedback-based scoring helps in particular to boost blocking efficiency and effectiveness for noisy datasets with many matching pairs (i.e. containing large clusters) such as **cars**, by enabling accurate selection of the cleanest blocks.
- The block intersection algorithm helps in particular with data sets having fewer matching pairs (i.e. with mainly small clusters) such as **citations** and **songs**, by providing a way to build small focused blocks with high fraction of matching pairs. Block intersection can also help in data sets like **products** and **camera** but the benefit is not as high as that in **songs** because many records in such data sets have unique identifiers (e.g. product model IDs) and thus initial blocks are reasonably clean.

Limitations and future work. **pBlocking** assumes an initial set of seed blocks to construct new blocks that prune out non-matching pairs. Any record pair that does not share any of the seed blocks would never be identified as a candidate even after running **pBlocking**. We believe that considering feedback from partial ER results can be helpful to explore other blocking strategies and is an interesting problem for future work.

CHAPTER 5

CLUSTERING WITH COMPARISON ORACLE: DATA SUMMARIZATION

Until now, we have studied techniques to improve robustness and scalability of entity resolution when employed with a binary oracle for supervision. These oracle queries are not helpful in settings where the ground truth clusters may not be known to the crowd worker. This chapter formalizes the notion of a noisy quadruplet oracle to provide supervision for metric based clustering methods like k-center clustering. These methods are particularly useful for data summarization applications.

Section 5.2 presents a formalization of the oracle along with two different noise models and Section 5.3 describes the well-known greedy algorithm for k-center clustering. Section 5.4 develops algorithms for different components of the greedy k-center algorithm, i.e. finding the farthest and nearest neighbor. Section 5.5 and Section 5.6 use these subroutines to solve the k-center and agglomerative clustering. Section 5.4.1.1 presents the theoretical analysis and Section 5.7 evaluates the techniques on various real-world case studies.

5.1 Introduction

Many real-world applications such as data summarization, social network analysis, facility location crucially rely on metric based comparative operations such as finding maximum, nearest neighbor search or ranking. As an example, data summarization aims to identify a small representative subset of the data where each representative is a summary of similar records in the dataset. Popular clustering algorithms such



Figure 5.1: Data summarization example

as k -center clustering and hierarchical clustering are often used for data summarization [131, 100]. In this chapter, we study clustering algorithms such as k -center clustering and agglomerative hierarchical clustering.

Example 6. Consider a data summarization task over a collection of images (shown in Figure 5.1). The goal is to identify k images (say $k = 3$) that summarize the different locations in the dataset. The images 1, 2 refer to Eiffel Tower in Paris, 3 is the Colosseum in Rome, 4 is the replica of Eiffel Tower at Las Vegas, USA, 5 is Venice and 6 is the Leaning Tower of Pisa. The ground truth output in this case would be $\{\{1, 2\}, \{3, 5, 6\}, \{4\}\}$. We calculated pairwise similarity between images using the visual features generated from Google Vision API [6]. The pair (1, 4) exhibits the highest similarity of 0.87, while all other pairs have similarity lower than 0.85. Distance between a pair of images u and v , denoted as $d(u, v)$, is defined as $(1 - \text{similarity between } u \text{ and } v)$. We ran a user experiment by querying crowd workers to answer simple Yes/No questions to help summarize the data (Please refer to Section 5.7.1 for more details).

In this example, we make the following observations.

- **Automated clustering techniques generate noisy clusters.** Consider the greedy approach for k -center clustering [105] which sequentially identifies the farthest record as a new cluster center. In this example, records 1 and 4 are placed in the same cluster by the greedy k -center clustering, thereby leading to poor quality. In general, automated techniques are known to generate erroneous similarity values between records due to missing information or even presence of noise [196, 194, 83].

- **Answering pairwise optimal cluster query is infeasible.** Answering whether 1 and 3 belong to the same optimal cluster when presented in isolation is impossible unless the crowd worker is aware of other records present in the dataset, and the granularity of the optimum clusters. Using the pair-wise Yes/No answers obtained from the crowd workers for the $\binom{6}{2}$ pairs in this example, the identified clusters achieved 0.40 F-score for $k = 3$. Please refer to Section 5.7.1 for additional details.

- **Comparing relative distance between the locations is easy.** Answering *relative distance* queries of the form ‘Is 1 closer to 3, or is 5 closer to 6?’ does not require any extra knowledge about other records in the dataset. For the 6 images in the example, we queried relative distance queries and the final clusters constructed for $k = 3$ achieved an F-score of 1.

In summary, we observe that humans have an innate understanding of the domain knowledge and can answer *relative distance queries* between records easily. Motivated by the aforementioned observations, we consider a *quadruplet* comparison oracle that compares the relative distance between two pairs of points (u_1, u_2) and (v_1, v_2) and outputs the pair with smaller distance between them breaking ties arbitrarily. Such oracle models have been studied extensively in the literature [122, 79, 51, 99, 181, 180, 120]. Even though quadruplet queries are easier than binary optimal queries, some oracle queries maybe harder than the rest. In a comparison

query, if there is a significant gap between the two distances being compared, then such queries are easier to answer [70, 44]. However, when the two distances are close, the chances of an error could increase. For example, ‘Is location in image 1 closer to 3, or 2 is closer to 6?’ maybe difficult to answer.

To capture noise in quadruplet comparison oracle answers, we consider two noise models. In the first noise model, when the pairwise distances are comparable, the oracle can return the pair of points that are farther instead of closer. Moreover, we assume that the oracle has access to all previous queries and can answer queries by acting adversarially. More formally, there is a parameter $\mu > 0$ such that if $\frac{\max d(u_1, u_2), d(v_1, v_2)}{\min d(u_1, u_2), d(v_1, v_2)} \leq (1 + \mu)$, then adversarial error may occur, otherwise the answers are correct. We call this ”Adversarial Noise Model”. This model is considered as a formalism to analyze settings where the oracle is an honest yet fallible adversary, who may answer certain difficult questions incorrectly [130, 20]. In the second noise model called ”Probabilistic Noise Model”, given a pair of distances, we assume that the oracle answers correctly with a probability of $1 - p$ for some fixed constant $p < \frac{1}{2}$. We consider a *persistent* probabilistic noise model, where our oracle answers are *persistent* i.e., query responses remain unchanged even upon repeating the same query multiple times. Such noise models have been studied extensively [144, 165, 99, 44, 46, 87] since the error due to oracles often does not change with repetition, and in fact, sometimes increases upon repeated querying [144, 165, 87]. This is in contrast to the noise models studied in [79] where response to every query is independently noisy. Persistent query models are much more difficult to handle than independent query models where repeating each query is sufficient to generate the correct answer by majority voting.

5.1.1 Chapter Outline and Contributions

We present algorithms for finding the maximum, nearest and farthest neighbors which are then used to explain techniques for k -center clustering and hierarchical clustering objectives under the adversarial and probabilistic noise model using comparison oracle. We show that the presented techniques have provable approximation guarantees for both the noise models, are efficient and obtain good query complexity. We empirically evaluate the robustness and efficiency of our techniques on real world datasets.

(i) **Farthest and Nearest Neighbor:** Finding farthest is similar to the problem of identifying the maximum over the set of considered record pairs. Maximum has received significant attention under both adversarial and probabilistic models [20, 81, 95, 70, 44, 94, 129, 96]. In this chapter, we provide the following results.

- **Adversarial model.** We present an algorithm that returns a value within $(1 + \mu)^3$ of the maximum among a set of n values V with probability $1 - \delta^1$ using $O(n \log^2(1/\delta))$ oracle queries and running time (Theorem 1).

To contrast our results with the state of the art, Ajtai et al. [20] study a slightly different additive adversarial error model where the answer of a maximum query is correct if the compared values differ by θ (for some $\theta > 0$) and otherwise the oracle answers adversarially. Under this setting, they give an additive 3θ -approximation with $O(n)$ queries. Although our model cannot be directly compared with theirs, we note that our model is scale invariant, and thus, provides a much stronger bound when distances are small. As a consequence, our algorithm can be used under an additive adversarial model as well providing the same approximation guarantees (Theorem 2).

Rest of the work in finding maximum allow repetition of queries and assume the answers are independent [81, 70]. As discussed earlier, persistent errors are much more

¹ δ is the confidence parameter and is standard in the literature of randomized algorithms.

difficult to handle than independent errors. In [81], the authors present an algorithm that finds the maximum using $O(n \log 1/\delta)$ queries and succeeds with probability $1 - \delta$. Therefore, even under persistent errors, we obtain guarantees close to the existing ones which assume independent error. The algorithms of [81, 70] do not extend to our model.

Nearest neighbor queries can be cast as “finding minimum” among a set of distances. Prior techniques have studied nearest neighbor search under noisy distance queries [142], where the oracle returns a noisy estimate of a distance between queried points, and repetitions are allowed. Neither the algorithm of [142], nor other techniques developed for maximum [20, 81] and top- k [70] extend for the nearest neighbor under our noise models.

(ii) **k -center Clustering:** k -center clustering is one of the fundamental models of clustering and is extremely well-studied [204, 188].

- **k -center under adversarial model.** We design an algorithm that returns a clustering that is a $2 + \mu$ approximation for small values of μ with probability $1 - \delta$ using $O(nk^2 + nk \log^2(k/\delta))$ queries (Theorem 3). In contrast, even when exact distances are known, k -center cannot be approximated better than a 2-factor unless $P = NP$ [188]. Therefore, we achieve near-optimal results.

- **k -center under probabilistic noise model.** For probabilistic noise, when optimal k -center clusters are of size at least $\Omega(\sqrt{n})$, our algorithm returns a clustering that achieves constant approximation with probability $1 - \delta$ using $O(nk \log^2(n/\delta))$ queries (Theorem 5).

To the best of our knowledge, even though k -center clustering is an extremely popular and basic clustering paradigm, it hasn’t been studied under the comparison oracle model, and we provide the first results in this domain.

(iii) **Single Linkage and Complete Linkage – Agglomerative Hierarchical**

Clustering: Under adversarial noise, we show a clustering technique that loses only a multiplicative factor of $(1 + \mu)^3$ in each merge operation and has an overall query complexity of $O(n^2)$. Prior work [99] considers comparison oracle queries to perform average linkage in which the unobserved pairwise similarities are generated according to a normal distribution. These techniques do not extend to our noise models.

Other Related Work. For finding the maximum among a given set of values, it is known that techniques based on tournament obtain optimal guarantees and are widely used [70]. For the problem of finding the nearest neighbor, techniques based on locality sensitive hashing generally work well in practice [24]. Clustering points using k -center objective is NP-hard and there are many well known heuristics and approximation algorithms [204] with the classical greedy algorithm achieving an approximation ratio of 2. All these techniques are not applicable when pairwise distances are unknown. As distances between points cannot always be accurately estimated, many recent techniques leverage supervision in the form of an oracle. Most oracle based clustering frameworks consider ‘optimal cluster’ queries [144, 121, 145, 58, 107] to identify ground truth clusters. Recent techniques for distance based clustering objectives, such as k -means [25, 54, 127, 128] and k -median [19] use optimal cluster queries in addition to distance information for obtaining better approximation guarantees. As ‘optimal cluster’ queries can be costly or sometimes infeasible, there has been recent interest in leveraging distance based comparison oracles for other problems similar to our quadruplet oracles [79, 99].

Distance based comparison oracles have been used to study a wide range of problems and we list a few of them – learning fairness metrics [122], top-down hierarchical clustering with a different objective [79, 51, 99], correlation clustering [181] and classification [180, 120], identify maximum [111, 189], top- k elements [129, 164, 63, 70, 133, 76], information retrieval [126], skyline computation [190].

To the best of our knowledge, there is no work that considers *quadruplet* comparison oracle queries to perform k -center clustering and single/complete linkage based hierarchical clustering.

Closely related to finding maximum, sorting has also been well studied under various comparison oracle based noise models [44, 43]. The work of [70] considers a different probabilistic noise model with error varying as a function of difference in the values but they assume that each query is independent and therefore repetition can help boost the probability of success. Using a quadruplet oracle, [99] studies the problem of recovering a hierarchical clustering under a planted noise model and is not applicable for single linkage.

5.2 Preliminaries

Let $V = \{v_1, v_2, \dots, v_n\}$ be a collection of n records such that each record maybe associated with a value $val(v_i), \forall i \in [1, n]$. We assume that there exists a total ordering over the values of elements in V . For simplicity we denote the value of record v_i as v_i instead of $val(v_i)$ whenever it is clear from the context.

Given this setting, we define a comparison oracle that compares the values of any pair of records (v_i, v_j) and outputs **Yes** if $v_i \leq v_j$ and **No** otherwise.

Definition 4 (Comparison Oracle). *An oracle is a function $\mathcal{O}_c : V \times V \rightarrow \{\text{Yes}, \text{No}\}$. Each oracle query considers two values as input and outputs $\mathcal{O}_c(v_1, v_2) = \text{Yes}$ if $v_1 \leq v_2$ and **No** otherwise.*

Note that a comparison oracle is defined for any pair of values. Given this oracle setting, we define the problem of identifying the maximum over the records V .

Problem 1 (Maximum). *Given a collection of n records $V = \{v_1, \dots, v_n\}$ and access to a comparison oracle \mathcal{O}_c , identify the $\arg \max_{v_i \in V} v_i$ with minimum number of queries to the oracle.*

The problem of identifying the record corresponding to the smallest value in V is a natural extension of Problem 1.

5.2.1 Quadruplet Oracle Comparison Query

In applications that consider distance based comparison of records like nearest neighbor identification, the records $V = \{v_1, \dots, v_n\}$ are generally considered to be present in a high-dimensional metric space along with a distance $d : V \times V \rightarrow \mathbb{R}^+$ defined over pairs of records. We assume that the embedding of records in latent space is not known, but there exists an underlying ground truth [24]. Prior techniques mostly assume complete knowledge of accurate distance metric and are not applicable in our setting. In order to capture the setting where we can compare distances between pairs of records, we define a quadruplet oracle below.

Definition 5 (Quadruplet Oracle). *An oracle is a function $\mathcal{O}_c : V \times V \times V \times V \rightarrow \{\text{Yes}, \text{No}\}$. Each oracle query considers two pairs of records as input and outputs $\mathcal{O}_c(v_1, v_2, v_3, v_4) = \text{Yes}$ if $d(v_1, v_2) \leq d(v_3, v_4)$ and **No** otherwise.*

The quadruplet oracle is equivalent to the comparison oracle discussed before with a difference that the two values being compared are associated with pairs of records as opposed to individual records. While implementing the oracle over crowdsourcing platforms, crowd workers require context of the query to answer comparison queries. For example, while comparing the distance between pictures in Figure 5.1, oracle needs context that relative distance query is with respect to geographical distance between records and not architecture. In terms of architecture, Eiffel tower in Paris and its replica in Las Vegas would be labelled closer than any of the other pairs in Figure 5.1, even though they are geographically the farthest. The context used to evaluate distance may be different between the two pairs (v_1, v_2) and (v_3, v_4) . For example, the task to cluster different companies may consider difference in revenue while comparing well established companies but may consider customer base while

comparing startups. In this work, we assume that the context is provided as input to the oracle and do not study different ways to specify the context.

Given this oracle setting, we define the problem of identifying the farthest record over V with respect to a query point q as follows.

Problem 2 (Farthest point). *Given a collection of n records $V = \{v_1, \dots, v_n\}$, a query record q and access to a quadruplet oracle \mathcal{O}_c , identify $\arg \max_{v_i \in V \setminus \{q\}} d(q, v_i)$.*

Similarly, the nearest neighbor query returns a point that satisfies $\arg \min_{u_i \in V \setminus \{q\}} d(q, u_i)$. Now, we formally define the k-center clustering problem.

Problem 3 (k-center clustering). *Given a collection of n records $V = \{v_1, \dots, v_n\}$ and access to a comparison oracle \mathcal{O}_c , identify k centers (say $S \subseteq V$) and a mapping of records to corresponding centers, $\pi : V \rightarrow S$, such that the maximum distance of any record from its center, i.e., $\max_{v_i \in V} d(v_i, \pi(v_i))$ is minimized.*

We assume that the points $v_i \in V$ exist in a metric space and the distance between any pair of points is not known. We denote the *unknown* distance between any pair of points (v_i, v_j) where $v_i, v_j \in V$ as $d(v_i, v_j)$ and use k to denote the number of clusters. Optimal clusters are denoted as C^* with $C^*(v_i) \subseteq V$ denoting the set of points belonging to the optimal cluster containing v_i . Similarly, $C(v_i) \subseteq V$ refers to the nodes belonging to the cluster containing v_i for any clustering given by $C(\cdot)$.

In addition to the k-center clustering, we study single linkage and complete linkage-agglomerative clustering techniques where the distance metric over the records is not known apriori. These techniques initialize each record v_i in a separate singleton cluster and sequentially merge the pair of clusters having the least distance between them. In case of single linkage, the distance between two clusters C_1 and C_2 is characterized by the closest pair of records defined as:

$$d_{SL}(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2} d(v_i, v_j)$$

In complete linkage, the distance between a pair of clusters C_1 and C_2 is calculated by identifying the farthest pair of records, $d_{CL}(C_1, C_2) = \max_{v_i \in C_1, v_j \in C_2} d(v_i, v_j)$.

5.2.2 Noise Models

The oracle models discussed in Problem 1, 2 and 3 assume that the oracle answers every comparison query correctly. In real world applications, however, the answers can be wrong which can lead to noisy results. To formalize the notion of noise, we consider two different models. First, an adversarial noise model considers a setting where a comparison query can be adversarially wrong if the two values being compared are within a multiplicative factor of $(1 + \mu)$ for some constant $\mu > 0$.

$$\mathcal{O}_c(v_1, v_2) = \begin{cases} \text{Yes, if } v_1 < \frac{1}{(1+\mu)}v_2 \\ \text{No, if } v_1 > (1+\mu)v_2 \\ \text{adversarially incorrect if } \frac{1}{(1+\mu)} \leq \frac{v_1}{v_2} \leq (1+\mu) \end{cases}$$

The parameter μ corresponds to the degree of error. For example, $\mu = 0$ implies a perfect oracle. The model extends to the quadruplet oracle as follows.

$$\mathcal{O}_c(v_1, v_2, v_3, v_4) = \begin{cases} \text{Yes, if } d(v_1, v_2) < \frac{1}{(1+\mu)}d(v_3, v_4) \\ \text{No, if } d(v_1, v_2) > (1+\mu)d(v_3, v_4) \\ \text{adversarially incorrect if } \frac{1}{(1+\mu)} \leq \frac{d(v_1, v_2)}{d(v_3, v_4)} \leq (1+\mu) \end{cases}$$

The second model considers a probabilistic noise model where each comparison query is incorrect independently with a probability $p < \frac{1}{2}$ and asking the same query yields the same response. We discuss ways to estimate μ and p from real data in Section 5.7.

5.3 Greedy k -center Clustering Algorithm

In this section, we present the traditional greedy algorithm for k -center clustering [105]. The greedy algorithm [105] initializes with an arbitrary point as the first cluster center and then iteratively identifies the next centers. In each iteration, it assigns all the points to the current set of clusters, by identifying the closest center for each point. Then, it finds the farthest point among the clusters and uses it as the new center. This technique requires $O(nk)$ distance comparisons in the absence of noise and guarantees 2-approximation of the optimal clustering objective. We provide the pseudo code for this approach in Algorithm 12. If we use Algorithm 12 where we replace every comparison with an oracle query, the generated clusters can be arbitrarily worse even for small error. In order to improve its robustness, Section 5.4 presents algorithms to perform assignment of points to respective clusters and farthest point identification. Section 5.5 uses these subroutines to analyze the quality of k -center clustering algorithm.

Algorithm 12 Greedy Algorithm

```

1: Input : Set of points  $V$ 
2: Output : Clusters  $\mathcal{C}$ 
3:  $s_1 \leftarrow$  arbitrary point from  $V$ ,  $S = \{s_1\}$ ,  $C = \{\{V\}\}$ .
4: for  $i = 2$  to  $k$  do
5:    $s_i \leftarrow \text{APPROX-FARTHEST}(S, C)$ 
6:    $S \leftarrow S \cup \{s_i\}$ 
7:    $C \leftarrow \text{ASSIGN}(S)$ 
8: end for
9: return  $\mathcal{C}$ 

```

5.4 Finding Maximum, Farthest and Nearest

In this section, we present robust algorithms to identify the record corresponding to a maximum value in V using comparison oracle under the noise models. Later we extend the algorithms to find the farthest and the nearest neighbor. We note that our algorithms for the adversarial model are parameter free (do not depend on μ) and

the algorithms for the probabilistic model can use $p = 0.5$ as a conservative estimate of the noise.

5.4.1 Adversarial Noise

Consider a trivial approach that maintains a running maximum value while sequentially processing the records, i.e., if a larger value is encountered, the current maximum value is updated to the larger value. This approach requires $n - 1$ comparisons. However, in the presence of adversarial noise, our output can have a significantly lower value compared to the correct maximum. In general, if v_{max} is the true maximum of V , then the above approach can return an approximate maximum whose value could be as low as $v_{max}/(1 + \mu)^{n-1}$. To see this, assume $v_1 = 1$, and $v_i = (1 + \mu - \epsilon)^i$ where $\epsilon > 0$ is very close to 0. It is possible that while comparing v_i and v_{i+1} , the oracle returns v_i as the larger element. If this mistake is repeated for every i , then, v_1 will be declared as the maximum element whereas the correct answer is $v_n \approx v_1(1 + \mu)^{n-1}$.

To improve upon this naive strategy, we introduce a natural *keeping score* based idea where given a set $S \subseteq V$ of records, we maintain $\text{Count}(v, S)$ that is equal to the number of values smaller than v in S .

$$\text{Count}(v, S) = \sum_{x \in S \setminus \{v\}} 1\{\mathcal{O}_c(v, x) == \text{No}\}$$

It is easy to observe that when the oracle makes no mistakes, $\text{Count}(s_{\max}, S) = |S| - 1$ and obtains the highest score, where s_{\max} is the maximum value in S . Using this observation, in Algorithm 13, we output the value with the highest Count score.

Given a set of records V , we show in Lemma 1 that $\text{COUNT-MAX}(V)$ obtained using Algorithm 13 always returns a good approximation of the maximum value in V .

Algorithm 13 COUNT-MAX(S) : finds the max. value by counting

```

1: Input : A set of values  $S$ 
2: Output : An approximate maximum value of  $S$ 
3: for  $v \in S$  do
4:   Calculate  $\text{Count}(v, S)$ 
5: end for
6:  $u_{\max} \leftarrow \arg \max_{v \in S} \text{Count}(v, S)$ 
7: return  $u_{\max}$ 

```

Lemma 1. *Given a set of values V with maximum value v_{\max} , COUNT-MAX(V) returns a value u_{\max} where $u_{\max} \geq v_{\max}/(1 + \mu)^2$ using $O(|V|^2)$ oracle queries.*

Proof. Let $v_{\max} = \max\{x \in V\}$. Consider a value $w \in V$ such that $w < \frac{v_{\max}}{(1+\mu)^2}$. We compare the **Count** values for v_{\max} and w given by, $\text{Count}(v_{\max}, V) = \sum_{x \in V} 1\{\mathcal{O}_c(v_{\max}, x) == \text{No}\}$ and $\text{Count}(w, V) = \sum_{x \in V} 1\{\mathcal{O}_c(w, x) == \text{No}\}$. We argue that w can never be returned by Algorithm 13, i.e., $\text{Count}(w, S) < \text{Count}(v_{\max}, V)$.

$$\begin{aligned}
\text{Count}(v_{\max}, V) &= \sum_{x \in V} 1\{\mathcal{O}_c(v_{\max}, x) == \text{No}\} \\
&\geq \sum_{x \in V \setminus \{v_{\max}\}} 1\{x < v_{\max}/(1 + \mu)\} \\
&= 1\{\mathcal{O}_c(v_{\max}, w) == \text{No}\} + \sum_{x \in S \setminus \{v_{\max}, w\}} 1\{x < v_{\max}/(1 + \mu)\} \\
&= 1 + \sum_{x \in V \setminus \{v_{\max}, w\}} 1\{x < v_{\max}/(1 + \mu)\}
\end{aligned}$$

$$\begin{aligned}
\text{Count}(w, V) &= \sum_{y \in V} 1\{\mathcal{O}_c(w, y) == \text{No}\} = \sum_{y \in S \setminus \{w, v_{\max}\}} 1\{\mathcal{O}_c(w, y) == \text{No}\} \\
&\leq \sum_{y \in V \setminus \{w, v_{\max}\}} 1\{y \leq (1 + \mu)w\} \\
&\leq \sum_{y \in V \setminus \{w, v_{\max}\}} 1\{y \leq v_{\max}/(1 + \mu)\}
\end{aligned}$$

Combining the two, we have :

$$\mathbf{Count}(v_{\max}, V) > \mathbf{Count}(w, V)$$

This shows that the **Count** of v_{\max} is strictly greater than the count of any point w with $w < \frac{v_{\max}}{(1+\mu)^2}$. Therefore, our algorithm would have output v_{\max} instead of w . For calculating the **Count** for all values in V , we make at most $|V|^2$ oracle queries as we compare every value with every other value. Finally, we output the maximum value as the value with the highest **Count**. Hence, the claim. \square

Using Example 7, when $\mu = 1$, we demonstrate that $(1 + \mu)^2 = 4$ approximation ratio is achieved by Algorithm 13.

Example 7. *Let S denote a set of four records u, v, w and t with ground truth values 51, 101, 102 and 202, respectively. While identifying the maximum value under adversarial noise with $\mu = 1$, the oracle must return a correct answer to $\mathcal{O}_c(u, t)$ and all other oracle query answers can be incorrect adversarially. If the oracle answers all other queries incorrectly, we have, **Count** values of t, w, u, v are 1, 1, 2, and 2 respectively. Therefore, u and v are equally likely, and when Algorithm 13 returns u , we have a $202/51 \approx 3.96$ approximation.*

From Lemma 1, we have that $O(n^2)$ oracle queries where $|V| = n$, are required to get $(1 + \mu)^2$ approximation. In order to improve the query complexity, we use a *tournament* to obtain the maximum value. Algorithm 14 presents pseudo code of the approach that takes values V as input and outputs an approximate maximum value. It constructs a balanced λ -ary tree \mathcal{T} containing n leaf nodes such that a random permutation of the values V is assigned to the leaves of \mathcal{T} . In a *tournament*, the internal nodes of \mathcal{T} are processed bottom-up such that at every internal node w , we assign the value that is largest among the children of w . To identify the largest value,

we calculate $\arg \max_{v \in \text{children}(w)} \text{Count}(v, \text{children}(w))$ at the internal node w , where $\text{Count}(v, X)$ refers to the number of elements in X that are considered smaller than v . Finally, we return the value at the root of \mathcal{T} as our output. In Lemma 2, we show that Algorithm 14 returns a value that is a $(1 + \mu)^{2 \log_\lambda n}$ multiplicative approximation of the maximum value.

Algorithm 14 TOURNAMENT : finds the maximum value using a tournament tree

```

1: Input : Set of values  $V$ , Degree  $\lambda$ 
2: Output : An approximate maximum value  $u_{\max}$ 
3: Construct a balanced  $\lambda$ -ary tree  $\mathcal{T}$  with  $|V|$  nodes as leaves.
4: Let  $\pi_V$  be a random permutation of  $V$  assigned to leaves of  $\mathcal{T}$ 
5: for  $i = 1$  to  $\log_\lambda |V|$  do
6:   for internal node  $w$  at level  $\log_\lambda |V| - i$  do
7:     Let  $U$  denote the children of  $w$ .
8:     Set the internal node  $w$  to  $\text{COUNT-MAX}(U)$ 
9:   end for
10: end for
11:  $u_{\max} \leftarrow$  value at root of  $\mathcal{T}$ 
12: return  $u_{\max}$ 

```

Lemma 2. Suppose v_{\max} is the maximum value among the set of records V . Algorithm 14 outputs a value u_{\max} such that $u_{\max} \geq \frac{v_{\max}}{(1+\mu)^{2 \log_\lambda n}}$ using $O(n\lambda)$ oracle queries.

Proof. From Lemma 1, we have that we lose a factor of $(1 + \mu)^2$ in each level of the tournament tree, we have that after $\log_\lambda n$ levels, the final output will have an approximation guarantee of $(1 + \mu)^{2 \log_\lambda n}$. The total number of queries used is given by : $\sum_{i=0}^{\log_\lambda n} \frac{|V_i|}{\lambda} \lambda^2 = O(n\lambda)$ where V_i is the number of records at level i . \square

According to Lemma 2, Algorithm 14 identifies a constant approximation when $\lambda = \Theta(n)$, μ is a fixed constant and has a query complexity of $\Theta(n^2)$. By reducing the degree of the tournament tree from λ to 2, we can achieve $\Theta(n)$ query complexity, but with a worse approximation ratio of $(1 + \mu)^{\log n}$. The idea of using a *tournament* for finding maximum has been studied in the past [70, 81].

Now, we describe our main algorithm (Algorithm 16) that uses the following observation to improve the overall query complexity.

Observation 1. *At an internal node $w \in \mathcal{T}$, the identified maximum is incorrect only if there exists $x \in \text{children}(w)$ that is very close to the true maximum (say w_{\max}), i.e. $\frac{w_{\max}}{(1+\mu)} \leq x \leq (1+\mu)w_{\max}$.*

Based on the above observation, our Algorithm MAX-ADV uses two steps to identify a good approximation of v_{\max} . Consider the case when there are a lot of values that are close to v_{\max} . In Algorithm MAX-ADV, we use a subset $\tilde{V} \subseteq V$ of size \sqrt{nt} (for a suitable choice of parameter t) obtained using uniform sampling with replacement. We show that using a sufficiently large subset \tilde{V} , obtained by sampling, we ensure that at least one value that is closer to v_{\max} is in \tilde{V} , thereby giving a good approximation of v_{\max} .

In order to handle the case when there are only a few values closer to v_{\max} , we divide the entire data set into l disjoint parts (for a suitable choice of l) and run the TOURNAMENT algorithm with degree $\lambda = 2$ on each of these parts separately (Algorithm 15). As there are very few points close to v_{\max} , the probability of comparing any such value with v_{\max} is small, and this ensures that in the partition containing v_{\max} , TOURNAMENT returns v_{\max} . We collect the maximum values returned by Algorithm 14 from all the partitions and include these values in T in Algorithm MAX-ADV. We repeat this procedure t times and set $l = \sqrt{n}, t = 2 \log(2/\delta)$ to achieve the desired success probability $1 - \delta$. We combine the outputs of both the steps, i.e., \tilde{V} and T and output the maximum among them using COUNT-MAX. This ensures that we get a good approximation as we use the best of both the approaches.

Theoretical Guarantees. In order to prove approximation guarantee of Algorithm 16, we first argue that the sample \tilde{V} contains a good approximation of the maximum value v_{\max} with a high probability. Let C denote the set of values that are very close to v_{\max} . Suppose $C = \{u : v_{\max}/(1+\mu) \leq u \leq v_{\max}\}$. In Lemma 3,

Algorithm 15 TOURNAMENT-PARTITION

1: **Input** : Set of values V , number of partitions l
2: **Output** : A set of maximum values from each partition
3: Randomly partition V into l equal parts V_1, V_2, \dots, V_l
4: **for** $i = 1$ to l **do**
5: $p_i \leftarrow \text{TOURNAMENT}(V_i, 2)$
6: $T \leftarrow T \cup \{p_i\}$
7: **end for**
8: **return** T

Algorithm 16 MAX-ADV : Maximum with Adversarial Noise

1: **Input** : Set of values V , number of iterations t , partitions l
2: **Output** : An approximate maximum value u_{\max}
3: $i \leftarrow 1, T \leftarrow \phi$
4: Let \tilde{V} denote a sample of size \sqrt{nt} selected uniformly at random (with replacement) from V .
5: **for** $i \leq t$ **do**
6: $T_i \leftarrow \text{TOURNAMENT-PARTITION}(V, l)$
7: $T \leftarrow T \cup T_i$
8: **end for**
9: $u_{\max} \leftarrow \text{COUNT-MAX}(\tilde{V} \cup T)$
10: **return** u_{\max}

we first show that \tilde{V} contains a value $v_j \in \tilde{V}$ such that $v_j \geq v_{\max}/(1 + \mu)$, whenever the size of C is large, i.e., $|C| > \sqrt{n}/2$. Otherwise, we show that we can recover v_{\max} correctly with probability $1 - \delta/2$ whenever $|C| \leq \sqrt{n}/2$.

Lemma 3. 1. If $|C| > \sqrt{n}/2$, then there exists a value $v_j \in \tilde{V}$ satisfying $v_j \geq v_{\max}/(1 + \mu)$ with probability of $1 - \delta/2$.

2. Suppose $|C| \leq \sqrt{n}/2$. Then, T contains v_{\max} with probability at least $1 - \delta/2$.

Proof. Case 1: Consider the first step where we use a uniformly random sample \tilde{V} of $\sqrt{nt} = 2\sqrt{n} \log(2/\delta)$ values from V (obtained by sampling with replacement). Given $|C| \geq \frac{\sqrt{n}}{2}$, probability that \tilde{V} contains a value from C is given by

$$\Pr[\tilde{V} \cap C \neq \phi] = 1 - \left(1 - \frac{|C|}{n}\right)^{|\tilde{V}|} > 1 - \left(1 - \frac{1}{2\sqrt{n}}\right)^{2\sqrt{n} \log(2/\delta)} > 1 - \delta/2$$

So, with probability $1 - \delta/2$, there exists a value $u \in C \cap \tilde{V}$.

Case 2:

In every iteration $i \leq t$ of Algorithm 16, we have that $v_{\max} \in T_i$ with probability $\frac{1}{2}$ (Using Lemma 4). To increase the success probability, we run this procedure t times and obtain all the outputs. Among the $t = 2 \log(2/\delta)$ runs of Algorithm 14, we have that v_{\max} is never compared with any value of C in at least one of the iterations with a probability at least

$$1 - (1 - 1/2)^{2 \log(2/\delta)} \geq 1 - \frac{\delta}{2}$$

Hence, $T = \cup_i T_i$ contains v_{\max} with a probability $1 - \frac{\delta}{2}$. \square

Now, we briefly provide a sketch of the proof of Lemma 3. Consider the first step, where we use a uniformly random sample \tilde{V} of $\sqrt{n}t$ points from V (obtained with replacement). When $|C| \geq \sqrt{n}/2$, probability that \tilde{V} contains a value from C is given by $1 - (1 - |C|/n)^{|\tilde{V}|} = 1 - (1 - \frac{1}{2\sqrt{n}})^{2\sqrt{n} \log(2/\delta)} \approx 1 - \delta/2$.

In the second step, Algorithm 16 uses a modified tournament tree that partitions the set V into $l = \sqrt{n}$ parts of size $n/l = \sqrt{n}$ each and identifies a maximum p_i from each partition V_i using Algorithm 14. We have that the expected number of elements from C in a partition V_i containing v_{\max} is $|C|/l = \sqrt{n}/(2\sqrt{n}) = 1/2$. Thus by Markov's inequality, the probability that V_i contains a value from C is $\leq 1/2$. With $1/2$ probability, v_{\max} will never be compared with any point from C in the partition V_i . To increase the success probability, we run this procedure t times and obtain all the outputs. Among the t runs of Algorithm 14, we argue that v_{\max} is never compared with any value of C in at least one of the iterations with a probability at least $1 - (1 - 1/2)^{2 \log(2/\delta)} \geq 1 - \delta/2$.

In Lemma 1, we show that using COUNT-MAX we get a $(1 + \mu)^2$ multiplicative approximation. Combining it with Lemma 3, we have that u_{\max} returned by Algorithm 16 satisfies $u_{\max} \geq \frac{v_{\max}}{(1+\mu)^3}$ with probability $1 - \delta$. For query complexity, Algorithm 15 identifies $\sqrt{n}t$ samples denoted by \tilde{V} . These identified values, along

with T are then processed by COUNT-MAX to identify the maximum u_{\max} . This step requires $O(|\tilde{V} \cup T|^2) = O(n \log^2(1/\delta))$ oracle queries.

Theorem 1. *Given a set of values V , Algorithm 16 returns a $(1 + \mu)^3$ approximation of maximum value with probability $1 - \delta$ using $O(n \log^2(1/\delta))$ oracle queries.*

Proof. In Algorithm 16, we first identify an approximate maximum value using Sampling. If $|C| \geq \frac{\sqrt{n}}{2}$, then, from Lemma 3, we have that the value returned is a $(1 + \mu)$ approximation of the maximum value of V . Otherwise, T contains v_{\max} with a probability $1 - \delta/2$. As we use COUNT-MAX on the set $\tilde{V} \cup T$, we know that the value returned, i.e., u_{\max} is a $(1 + \mu)^2$ of the maximum among values from $\tilde{V} \cup T$. Therefore, $u_{\max} \geq \frac{v_{\max}}{(1+\mu)^3}$. Using union bound, the total probability of failure is δ .

For query complexity, Algorithm 15 obtains a set \tilde{V} of \sqrt{nt} sample values. Along with the set T obtained (where $|T| = \frac{nt}{l}$), we use COUNT-MAX on $\tilde{V} \cup T$ to output the maximum u_{\max} . This step requires $O(|\tilde{V} \cup T|^2) = O((\sqrt{nt} + \frac{nt}{l})^2)$ oracle queries. In an iteration i , for obtaining T_i , we make $O(\sum_j |V_j|) = O(n)$ oracle queries (Claim 1), and for t iterations, we make $O(nt)$ queries. Using $t = 2 \log(2/\delta)$, $l = \sqrt{n}$, in total, we make $O(nt + (\sqrt{nt} + \frac{nt}{l})^2) = O(n \log^2(1/\delta))$ oracle queries. Hence, the theorem. \square

Extension to Farthest and Nearest Neighbor. Given a set of records V , the farthest record from a query u corresponds to the record $u' \in V$ such that $d(u, u')$ is maximum. This query is equivalent to finding maximum in the set of distance values given by $D(u) = \{d(u, u') \mid \forall u' \in V\}$ containing n values for which we already developed algorithms in Section 5.4. Since the ground truth distance between any pair of records is not known, we require quadruplet oracle (instead of comparison oracle) to identify the maximum element in $D(u)$. Similarly, the nearest neighbor of query record u corresponds to finding the record with minimum distance value

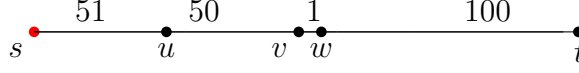


Figure 5.2: Example for Lemma 1 with $\mu = 1$.

in $D(u)$. Algorithms for finding maximum from previous sections, extend for these settings with similar guarantees.

Example 8. Figure 5.2 shows a worst-case example for the approximation guarantee to identify the farthest point from s (with $\mu = 1$). Similar to Example 7, we have, *Count* values of t, w, u, v are 1, 1, 2, 2 respectively. Therefore, u and v are equally likely, and when Algorithm 13 outputs u , we have a ≈ 3.96 approximation.

5.4.1.1 Helper Lemmas

In this section, we present the helper lemmas that are required to analyze the proposed algorithm. Let the maximum value among V be denoted by v_{\max} and the set of records for which the oracle answer *can* be incorrect is given by

$$C = \{u \mid u \in V, u \geq \frac{v_{\max}}{1 + \mu}\}$$

Claim 1. For any partition V_i , $\text{TOURNAMENT}(V_i)$ uses at most $2|V_i|$ oracle queries.

Proof. Consider the i th round in TOURNAMENT . We can observe that the number of remaining values is at most $\frac{|V_i|}{2^i}$. So, we make $\frac{|V_i|}{2^{i+1}}$ many oracle queries in this round. Total number of oracle queries made is

$$\sum_{i=0}^{\log n} \frac{|V_i|}{2^{i+1}} \leq 2|V_i|$$

□

We now prove the helper lemma to consider the case when $|C| \leq \sqrt{n}/2$.

Lemma 4. *Suppose the partition V_i contains the maximum value v_{\max} of V . If $|C| \leq \sqrt{n}/2$, then, $\text{TOURNAMENT}(V_i)$ returns the v_{\max} with probability $1/2$.*

Proof. Algorithm 16 uses a modified tournament tree that partitions the set V into $l = \sqrt{n}$ parts of size $\frac{n}{l} = \sqrt{n}$ each and identifies a maximum p_i from each partition V_i using Algorithm 14. If $v_{\max} \in V_i$, then,

$$\mathbf{E}[|C \cap V_i|] = \frac{|C|}{l} = \frac{\sqrt{n}}{2\sqrt{n}} = \frac{1}{2}$$

Using Markov's inequality, the probability that V_i contains a value from C is given by :

$$\Pr[|C \cap V_i| \geq 1] \leq \mathbf{E}[|C \cap V_i|] \leq \frac{1}{2}$$

Therefore, with at least a probability of $\frac{1}{2}$, v_{\max} will never be compared with any point from C in the partition V_i containing v_{\max} . Hence, v_{\max} is returned by $\text{TOURNAMENT}(V_i)$ with probability $1/2$.

□

5.4.2 Probabilistic Noise

For probabilistic noise, the algorithms described in Section 5.4.1 do not extend. In this section, we show that it is possible to compute the farthest point within a small additive error under the probabilistic model, if the data set satisfies an additional property discussed below. For the simplicity of exposition, we assume $p \leq 0.40$, though our algorithms work for any value of $p < 0.5$.

One of the challenges in developing robust algorithms for farthest identification is that every relative distance comparison of records from u ($\mathcal{O}_c(u, v_i, u, v_j)$ for some $v_i, v_j \in V$) may be answered incorrectly with constant error probability p and the success probability cannot be boosted by repetition. We overcome this challenge by performing pairwise comparisons in a robust manner. Suppose the desired failure

probability is δ , we observe that if $\Theta(\log(1/\delta))$ records closest to the query u are known (say S) and $\max_{x \in S} \{d(u, x)\} \leq \alpha$ for some $\alpha > 0$, then each pairwise comparison of the form $\mathcal{O}_c(u, v_i, u, v_j)$ can be replaced by Algorithm PAIRWISECOMP and use it to execute Algorithm 16. Algorithm 17 takes the two records v_i and v_j as input along with S and outputs **Yes** or **No** where **Yes** denotes that v_i is closer to u . We calculate $\mathbf{FCOUNT}(v_i, v_j) = \sum_{x \in S} \mathbf{1}\{\mathcal{O}_c(v_i, x, v_j, x) == \mathbf{Yes}\}$ as a robust estimate where the oracle considers v_i to be closer to x than v_j . If $\mathbf{FCOUNT}(v_i, v_j)$ is smaller than $0.3|S| \leq (1-p)|S|/2$ then we output **No** and **Yes** otherwise. Therefore, every pairwise comparison query is replaced with $\Theta(\log(1/\delta))$ quadruplet queries using Algorithm 17.

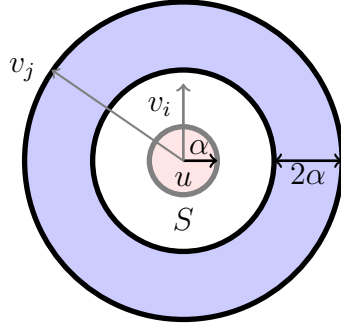
We argue that Algorithm 17 will output the correct answer with a high probability if $|d(u, v_j) - d(u, v_i)| \geq 2\alpha$ (See Fig 5.3). In Lemma 5, we show that, if $d(u, v_j) > d(u, v_i) + 2\alpha$, then, $\mathbf{FCOUNT}(v_i, v_j) \geq 0.3|S|$ with probability $1 - \delta$.

Lemma 5. *Suppose $\max_{v_i \in S} d(u, v_i) \leq \alpha$ and $|S| \geq 6 \log(1/\delta)$. Consider two records v_i and v_j such that $d(u, v_i) < d(u, v_j) - 2\alpha$ then $\mathbf{FCOUNT}(v_i, v_j) \geq 0.3|S|$ with a probability of $1 - \delta$*

Proof. Since $d(u, v_i) < d(u, v_j) - 2\alpha$, for a point $x \in S$,

$$\begin{aligned}
d(v_j, x) &\geq d(u, v_j) - d(u, x) \\
&> d(u, v_i) + 2\alpha - d(u, x) \\
&\geq d(v_i, x) - d(u, x) + 2\alpha - d(u, x) \\
&\geq d(v_i, x) + 2\alpha - 2d(u, x) \\
&\geq d(v_i, x)
\end{aligned}$$

So, $\mathcal{O}(v_i, x, v_j, x)$ is **No** with a probability p . As $p \leq 0.4$, we have :



In this example, $\mathcal{O}_c(u, v_i, u, v_j)$ is answered correctly with a probability $1 - p$. To boost the correctness probability, **FCOUNT** uses the queries $\mathcal{O}_c(x, v_i, x, v_j)$, $\forall x$ in the red region around u , denoted by S .

Figure 5.3: Algorithm 17 returns ‘Yes’ as $d(u, v_i) < d(u, v_j) - 2\alpha$.

$$\mathbf{E}[\mathbf{FCOUNT}(v_i, v_j)] = (1 - p)|S|$$

$$\Pr[\mathbf{FCOUNT}(v_i, v_j) \leq 0.3|S|] \leq \Pr[\mathbf{FCOUNT}(v_i, v_j) \leq (1 - p)|S|/2]$$

From Hoeffding’s inequality (with binary random variables), we have with a probability $\exp(-\frac{|S|(1-p)^2}{2}) \leq \delta$ (using $|S| \geq 6 \log(1/\delta)$, $p < 0.4$) : $\mathbf{FCOUNT}(v_i, v_j) \leq (1 - p)|S|/2$. Therefore, with probability at most δ , we have, $\mathbf{FCOUNT}(v_i, v_j) \leq 0.3|S|$. \square

With the help of Algorithm 17, relative distance query of any pair of records v_i, v_j from u can be answered correctly with a high probability provided $|d(u, v_i) - d(u, v_j)| \geq 2\alpha$. Therefore, the output of Algorithm 17 is equivalent to an additive adversarial error model where any quadruplet query can be adversarially incorrect if the distance $|d(u, v_i) - d(u, v_j)| < 2\alpha$ and correct otherwise. Algorithm 16 can be extended to the additive adversarial error model, such that each comparison $\mathcal{O}_c(u, v_i, u, v_j)$ is replaced by **PAIRWISECOMP** (Algorithm 17). We give an approximation guarantee, that loses an additive 6α following a similar analysis of Theorem 1.

Algorithm 17 **PAIRWISECOMP** (u, v_i, v_j, S)

- 1: Calculate $\mathbf{FCOUNT}(v_i, v_j) = \sum_{x \in S} \mathbf{1}\{\mathcal{O}_c(x, v_i, x, v_j) == \text{Yes}\}$
 - 2: **if** $\mathbf{FCOUNT}(v_i, v_j) < 0.3|S|$ **then**
 - 3: **return** No
 - 4: **else return** Yes
 - 5: **end if**
-

Farthest under probabilistic noise. For the sake of completeness, we restate the **Count** definition that is used in Algorithm COUNT-MAX. For every oracle comparison, we replace it with the *pairwise comparison* query described in Section 5.4.1. Let u be a query point and S denote a set of $\Theta(\log(n/\delta))$ points within a distance of α from u . We maintain a **Count** score for a given point $v_i \in V$ as :

$$\text{Count}(u, v_i, S, V) = \sum_{v_j \in V \setminus \{v_i\}} 1\{\text{PAIRWISE-COMP}(u, v_i, v_j, S) == \text{No}\}$$

Theorem 2. *Given a query vertex u and a set S with $|S| = \Omega(\log(n/\delta))$ such that $\max_{v \in S} d(u, v) \leq \alpha$ then the farthest identified using Algorithm 16 (with PAIRWISECOMP), denoted by u_{\max} is within 6α distance from the optimal farthest point, i.e., $d(u, u_{\max}) \geq \max_{v \in V} d(u, v) - 6\alpha$ with a probability of $1 - \delta$. Further the query complexity is $O(n \log^3(n/\delta))$.*

Proof. The proof is similar to Theorem 1. We first identify an approximate maximum value using Sampling. If $|C| \geq \frac{\sqrt{n}}{2}$, then we have that one of the sampled values is a 2α additive approximation of the maximum value of V . Otherwise, T contains v_{\max} with a probability $1 - \delta/2$. As we use COUNT-MAX on the set $\tilde{V} \cup T$, we know that the value returned, i.e., u_{\max} is a 4α of the maximum among values from $\tilde{V} \cup T$. Therefore, $d(u, u_{\max}) \geq d(u, v_{\max}) - 6\alpha$. Using union bound over $n \cdot t$ comparisons, the total probability of failure is δ .

For query complexity, Algorithm obtains a set \tilde{V} of \sqrt{nt} sample values. Along with the set T obtained (where $|T| = \frac{nt}{l}$), we use COUNT-MAX on $\tilde{V} \cup T$ to output the maximum u_{\max} . This step requires $O(|\tilde{V} \cup T|^2 |S|) = O((\sqrt{nt} + \frac{nt}{l})^2 \log(n/\delta))$ oracle queries. In an iteration i , for obtaining T_i , we make $O(\sum_j |V_j| \log(n/\delta)) = O(n \log(n/\delta))$ oracle queries (Claim 1), and for t iterations, we make $O(nt \log(n/\delta))$ queries. Using $t = 2 \log(2n/\delta)$, $l = \sqrt{n}$, in total, we make $O(nt \log(n/\delta) + (\sqrt{nt} + \frac{nt}{l})^2 \log(n/\delta)) = O(n \log^3(n/\delta))$ oracle queries. Hence, the theorem. \square

5.5 k-center Clustering

We now use the subroutines discussed in Section 5.4 to solve k-center clustering problem with quadruplet oracles.

5.5.1 Adversarial Noise

In this case, the algorithm to identify k-centers is same as Algorithm 12 with modified functions for assignment and finding the farthest point. Now, we describe the two steps (APPROX-FARTHEST and ASSIGN) of the Greedy Algorithm that will complete the description of Algorithm 12.

Approx-Farthest. Given a clustering \mathcal{C} , and a set of centers S , we construct the pairs (v_i, s_j) where v_i is assigned to cluster $C(s_j)$ centered at $s_j \in S$. Using Algorithm 16, we identify the point, center pair that have the maximum distance i.e. $\arg \max_{v_i \in V} d(v_i, s_j)$, which corresponds to the farthest point. For the parameters, we use $l = \sqrt{n}$, $t = \log(2k/\delta)$ and number of samples $\tilde{V} = \sqrt{nt}$.

Assign. After identifying the farthest point, we reassign all the points to the centers (now including the farthest point as the new center) closest to them. We calculate a movement score called **MCount** for every point with respect to each center. $\text{MCount}(u, s_j) = \sum_{s_k \in S \setminus \{s_j\}} 1\{\mathcal{O}_c((s_j, u), (s_k, u)) == \text{Yes}\}$, for any record $u \in V$ and $s_j \in S$. This step is similar to COUNT-MAX Algorithm. We assign the point u to the center with the highest **MCount** value.

Example 9. Suppose we run k-center algorithm with $k = 2$ and $\mu = 1$ on the points in Example 8. The optimal centers are u and t with radius 51. On running our algorithm, suppose w is chosen as the first center and APPROX-FARTHEST calculates **Count** values similar to Example 7. We have, **Count** values of s, t, u, v are 1, 2, 3, 0 respectively. Therefore, our algorithm identifies u as the second center, achieving 3-approximation.

Theoretical Guarantees. We now prove the approximation guarantee obtained by Algorithm 12. In each iteration, we show that ASSIGN reassigns each point to a center with distance approximately similar to the distance from the closest center. This is surprising given that we only use MCount scores for assignment. Similarly, we show that APPROX-FARTHEST (Algorithm 16) identifies a close approximation to the true farthest point. Concretely, we show that u is assigned to a center which is a $(1 + \mu)^2$ approximation; Algorithm 16 identifies the farthest point w which is a $(1 + \mu)^5$ approximation.

In every iteration of the Greedy algorithm, if we identify an α -approximation of the farthest point, and a β -approximation when reassigning the points, then, we show that the clusters output are a $2\alpha\beta^2$ -approximation to the k -center objective. Combining all the claims, for a given error parameter μ , we obtain:

Theorem 3. *For $\mu < \frac{1}{18}$, Algorithm 12 achieves a $(2 + O(\mu))$ -approximation for the k -center objective using $O(nk^2 + nk \cdot \log^2(k/\delta))$ oracle queries with probability $1 - \delta$.*

We first prove the helper lemmas that analyze ASSIGN and APPROX-FARTHEST separately, which are then used to prove Theorem 3.

Lemma 6. *Suppose in an iteration t of Greedy algorithm, centers are given by S_t and we reassign points using ASSIGN which is a β -approximation to the correct assignment. In iteration $t + 1$, using this assignment, if we obtain an α -approximate farthest point using APPROX-FARTHEST, then, after k iterations, Greedy algorithm obtains a $2\alpha\beta^2$ -approximation for the k -center objective.*

Proof. Consider an optimum clustering C^* with centers u_1, u_2, \dots, u_k respectively: $C^*(u_1), C^*(u_2), \dots, C^*(u_k)$. Let the centers obtained by Algorithm 12 be denoted by S . If $|S \cap C^*(u_i)| = 1$ for all i , then, for some point $x \in C^*(u_i)$ assigned to $s_j \in S$ by Algorithm ASSIGN, we have

$$d(x, S \cap C^*(u_i)) \leq d(x, u_i) + d(u_i, S \cap C^*(u_i)) \leq 2OPT$$

$$\implies d(x, s_j) \leq \beta \min_{s_k \in S} d(x, s_k) \leq \beta d(x, S \cap C^*(u_i)) \leq 2\beta OPT$$

Therefore, every point in V is at a distance of at most $2\beta OPT$ from a center assigned in S .

Suppose for some j we have $|S \cap C^*(u_j)| \geq 2$. Let $s_1, s_2 \in S \cap C^*(u_j)$ and s_2 appeared after s_1 in iteration $t+1$. As $s_1 \in S_t$, we have $\min_{w \in S_t} d(w, s_2) \leq d(s_1, s_2)$. In iteration t , we know that the farthest point s_2 is an α -approximation of the farthest point (say f_t). Moreover, suppose s_2 assigned to cluster with center s_k in iteration t that is a β -approximation of its true center. Therefore,

$$\frac{1}{\alpha} \min_{w \in S_t} d(w, f_t) \leq d(s_k, s_2) \leq \beta \min_{w \in S_t} d(w, s_2) \leq \beta d(s_1, s_2)$$

Because s_1 and s_2 are in the same optimum cluster, from triangle inequality we have $d(s_1, s_2) \leq 2OPT$. Combining all the above we get $\min_{w \in S_t} d(w, f_t) \leq 2\alpha\beta OPT$ which means that farthest point of iteration t is at a distance of $2\alpha\beta OPT$ from S_t . In the subsequent iterations, the distance of any point to the final set of centers, given by S only gets smaller. Hence,

$$\max_v \min_{w \in S} d(v, w) \leq \max_v \min_{w \in S_t} d(v, w) = \min_{w \in S_t} d(f_t, w) \leq 2\alpha\beta OPT$$

However, when we output the final clusters and centers, the farthest point after k -iterations (say f_k) could be assigned to center $v_j \in S$ that is a β -approximation of the distance to true center.

$$d(f_k, v_j) \leq \beta \min_{w \in S} d(f_k, w) \leq 2\alpha\beta^2 OPT$$

Therefore, every point is assigned to a cluster with distance at most $2\alpha\beta^2 OPT$. Hence the claim. \square

Lemma 7. *Given a set S of centers, Algorithm ASSIGN assigns a point u to a cluster $s_j \in S$ such that $d(u, s_j) \leq (1 + \mu)^2 \min_{s_t \in S} \{d(u, s_t)\}$ using $O(nk)$ queries.*

Proof. The proof is essentially the same as Lemma 1 and uses MCount instead of Count. \square

Lemma 8. *Given a set of centers S , Algorithm 16 identifies a point v_j with probability $1 - \delta/k$, such that*

$$\min_{s_j \in S} d(v_j, s_j) \geq \max_{v_t \in V} \min_{s_t \in S} \frac{d(v_t, s_t)}{(1 + \mu)^5}$$

Proof. Suppose v_t is the farthest point assigned to center $s_t \in S$. Let v_j , assigned to $s_j \in S$ be the point returned by Algorithm 16. From Theorem 1, we have :

$$\begin{aligned} d(v_j, s_j) &\geq \frac{\max_{v_i \in V} d(v_i, s_i)}{(1 + \mu)^3} \\ &\geq \frac{d(v_t, s_t)}{(1 + \mu)^3} \geq \frac{\min_{s'_t \in S} d(v_t, s'_t)}{(1 + \mu)^3} \end{aligned}$$

Due to error in assignment, using Lemma 7

$$d(v_j, s_j) \leq (1 + \mu)^2 \min_{s'_j \in S} d(v_j, s'_j)$$

Combining the above equations we have

$$\min_{s'_j \in S} d(v_j, s'_j) \geq \frac{\min_{s'_t \in S} d(v_t, s'_t)}{(1 + \mu)^5}$$

For APPROX-FARTHEST, we use $l = \sqrt{n}$ and $t = \log(2k/\delta)$ and $\tilde{V} = \sqrt{n}t$. So, following the proof in Theorem 1, we succeed with probability $1 - \delta/k$. Hence, the lemma. \square

Lemma 9. *Given a current set of centers S ,*

1. **ASSIGN** assigns a point u to a cluster $C(s_i)$ such that $d(u, s_i) \leq (1 + \mu)^2 \min_{s_j \in S} \{d(u, s_j)\}$ using $O(nk)$ oracle queries additionally.
2. **APPROX-FARTHEST** identifies a point w in cluster $C(s_i)$ such that $\min_{s_j \in S} d(w, s_j) \geq \max_{v_t \in V} \min_{s_t \in S} d(v_t, s_t) / (1 + \mu)^5$ with probability $1 - \frac{\delta}{k}$ using $O(n \log^2(k/\delta))$ oracle queries.

Proof. (1) From Lemma 7, we have the claim. We assign a point to a cluster based on the scores the cluster center received in comparison to other centers. Except for the newly created center, we have previously queried every center with every other center. Therefore, number of *new* oracle queries made for every point is $O(k)$; that gives us a total of $O(nk)$ additional new queries used by **ASSIGN**.

(2) From Lemma 8, we have that $\min_{s_j \in S} d(w, s_j) \geq \max_{v_t \in V} \min_{s_t \in S} \frac{d(v_t, s_t)}{(1 + \mu)^5}$ with probability $1 - \delta/k$. As the total number of queries made by Algorithm 16 is $O(nt + (\frac{nt}{l} + \sqrt{nt})^2)$. For **APPROX-FARTHEST**, we use $l = \sqrt{n}$ and $t = \log(2k/\delta)$ and $\tilde{V} = \sqrt{nt}$, therefore, the query complexity is $O(n \log^2(k/\delta))$. \square

Theorem 4 (Theorem 3 restated). *For $\mu < \frac{1}{18}$, Algorithm 12 achieves a $(2 + O(\mu))$ -approximation for the k -center objective using $O(nk^2 + nk \cdot \log^2(k/\delta))$ oracle queries with probability $1 - \delta$.*

Proof. From the above discussed claim and Lemma 9, we have that Algorithm 12 achieves a $2(1 + \mu)^9$ approximation for k -center objective. When $\mu < \frac{1}{18}$, we can simplify the approximation factor to $2 + 18\mu$, i.e., $2 + O(\mu)$. From Lemma 9, we have that in each iteration, we succeed with probability $1 - \delta/k$. Using union bound, the failure probability is given by δ . For query complexity, as there are k iterations, and in each iteration we use **ASSIGN** and **APPROX-FARTHEST**, using Lemma 9, we have the theorem. \square

5.5.2 Probabilistic Noise

For probabilistic noise, each query can be incorrect with probability p and therefore, Algorithm 12 may lead to poor approximation guarantees. Algorithm 18 presents the pseudo-code of our algorithm for probabilistic noise. We denote the size of minimum cluster among optimum clusters C^* to be m , and total failure probability of our algorithms to be δ . We assume $p \leq 0.40$, a constant strictly less than $\frac{1}{2}$. Let $\gamma = 450$ be a large constant used in our algorithms which obtains the claimed guarantees.

Overview. Algorithm 18 operates in two phases. In the first phase (lines 3-12), we sample each point with a probability $\gamma \log(n/\delta)/m$ to identify a small sample of $\approx \frac{\gamma \log(n/\delta)}{m}$ points (denoted by \tilde{V}) and use Algorithm 18 to identify k centers iteratively. In this process, we also identify a *core* for each cluster (denoted by R). Formally, *core* is defined as a set of $\Theta(\log(n/\delta))$ points that are very close to the center with high probability. The cores are then used in the second phase (line 15) for the assignment of remaining points.

Algorithm 18 Greedy Clustering

```

1: Input : Set of points  $V$ , smallest cluster size  $m$ .
2: Output : Clusters  $C$ 
3: For every  $u \in V$ , include  $u$  in  $\tilde{V}$  with probability  $\frac{\gamma \log(n/\delta)}{m}$ 
4:  $s_1 \leftarrow$  select an arbitrary point from  $\tilde{V}$ ,  $S \leftarrow \{s_1\}$ 
5:  $C(s_1) \leftarrow \tilde{V}$ 
6:  $R(s_1) \leftarrow \text{IDENTIFY-CORE}(C(s_1), s_1)$ 
7: for  $i = 2$  to  $k$  do
8:    $s_i \leftarrow \text{APPROX-FARTHEST}(S, C)$ 
9:    $C, R \leftarrow \text{ASSIGN}(S, s_i, R)$ 
10:   $S \leftarrow S \cup \{s_i\}$ 
11: end for
12:  $C \leftarrow \text{ASSIGN-FINAL}(S, R, V \setminus \tilde{V})$ 
13: return  $C$ 

```

Now, we describe the main challenge in extending APPROX-FARTHEST and ASSIGN ideas of Algorithm 12. Given a cluster C containing the center s_i , when we find the APPROX-FARTHEST, the ideas from Section 5.4.1 give poor approximation. As shown in section 5.4.1, we can improve the approximation guarantee by

considering a set of $\Theta(\log(n/\delta))$ points closest to s_i , denoted by $R(s_i)$ and call them *core* of s_i . We argue that such an assumption of set R is justified. For example, consider the case when clusters are of size $\Theta(n)$ and sampling $k \log(n/\delta)$ points gives us $\log(n/\delta)$ points from each optimum cluster; which means that there are $\log(n/\delta)$ points within a distance of 2OPT from every sampled point where OPT refers to the optimum k -center objective.

Assign. Consider a point s_i such that we have to assign points to form the cluster $C(s_i)$ centered at s_i . We calculate an *assignment* score (called **ACount** in line 4) for every point u of a cluster $C(s_j) \setminus R(s_j)$ centered at s_j . **ACount** captures the total number of times u is considered to belong to the same cluster as that of x for each x in the core $R(s_j)$. Intuitively, points that belong to the same cluster as that of s_i are expected to have higher **ACount** score. Based on the scores, we move u to $C(s_i)$ or keep it in $C(s_j)$.

Algorithm 19 ASSIGN(S, s_i, R)

```

1:  $C(s_i) \leftarrow \{s_i\}$ 
2: for  $s_j \in S$  do
3:   for  $u \in C(s_j) \setminus R(s_j)$  do
4:      $\text{ACount}(u, s_i, s_j) = \sum_{v_k \in R(s_j)} \mathbf{1}\{\mathcal{O}_c(u, s_i, u, v_k) == \text{Yes}\}$ 
5:     if  $\text{ACount}(u, s_i, s_j) > 0.3|R(s_j)|$  then
6:        $C(s_i) \leftarrow C(s_i) \cup \{u\}; C(s_j) \leftarrow C(s_j) \setminus \{u\}$ 
7:     end if
8:   end for
9: end for
10:  $R(s_i) \leftarrow \text{IDENTIFY-CORE}(C(s_i), s_i)$ 
11: return C, R

```

Algorithm 20 IDENTIFY-CORE($C(s_i), s_i$)

```

1: for  $u \in C(s_i)$  do
2:    $\text{Count}(u) = \sum_{x \in C(s_i)} \mathbf{1}\{\mathcal{O}_c(s_i, x, s_i, u) == \text{No}\}$ 
3: end for
4:  $R(s_i)$  denote set of  $8\gamma \log(n/\delta)/9$  points with the highest Count values.
5: return  $R(s_i)$ 

```

Identify-Core. After forming cluster $C(s_i)$, we identify the *core* of s_i . For this, we calculate a score, denoted by **Count** and captures the number of times it is closer to s_i compared to other points in $C(s_i)$. Intuitively, we expect points with high values of **Count** to belong to $C^*(s_i)$ i.e., optimum cluster containing s_i . Therefore we sort these **Count** scores and return the highest scored points.

Approx-Farthest. For a set of clusters \mathcal{C} , and a set of centers S , we construct the pairs (v_i, s_j) where v_i is assigned to cluster $C(s_j)$ centered at $s_j \in S$ and each center $s_j \in S$ has a corresponding core $R(s_j)$. The farthest point can be found by finding the maximum distance (point, center) pair among all the points considered. To do so, we use the ideas developed in section 5.4.1.

We leverage CLUSTERCOMP (Algorithm 21) to compare the distance of two points, say v_i, v_j from their respective centers s_i, s_j . CLUSTERCOMP gives a robust answer to a pairwise comparison query to the oracle $\mathcal{O}_c(v_i, s_i, v_j, s_j)$ using the cores $R(s_i)$ and $R(s_j)$. CLUSTERCOMP can be used as a pairwise comparison subroutine in place of PAIRWISECOMP for the algorithm in Section 5.4 to calculate the farthest point. For every $s_i \in S$, let $\tilde{R}(s_i)$ denote an arbitrary set of $\sqrt{R(s_i)}$ points from $R(s_i)$. For a CLUSTERCOMP comparison query between the pairs (v_i, s_i) and (v_j, s_j) , we use these subsets in Algorithm 21 to ensure that we only make $\Theta(\log(n/\delta))$ oracle queries for every comparison. However, when the query is between points of the same cluster, say $C(s_i)$, we use all the $\Theta(\log(n/\delta))$ points from $R(s_i)$. For the parameters used to find maximum using Algorithm 16, we use $l = \sqrt{n}$, $t = \log(n/\delta)$.

Example 10. Suppose we run k -center Algorithm 18 with $k = 2$ and $m = 2$ on the points in Example 8. Let w denote the first center chosen and Algorithm 18 identifies the core $R(w)$ by calculating **Count** values. If $\mathcal{O}_c(u, w, s, w)$ and $\mathcal{O}_c(s, w, t, w)$ are answered incorrectly (with probability p), we obtain **Count** values of v, s, u, t as 3, 2, 1, 0 respectively; and v is added to $R(w)$. We identify the second center u by calculating

FCount for s, u and t (See Fig. 5.3). After assigning (using **ASSIGN**), the clusters identified are $\{w, v\}, \{u, s, t\}$, achieving 3-approximation.

Algorithm 21 CLUSTERCOMP (v_i, s_i, v_j, s_j)

```

1: comparisons  $\leftarrow 0$ , FCount( $v_i, v_j$ )  $\leftarrow 0$ 
2: if  $s_i = s_j$  then
3:   Let FCount( $v_i, v_j$ ) =  $\sum_{x \in R(s_i)} \mathbf{1}\{\mathcal{O}_c(v_i, x, v_j, x) == \text{Yes}\}$ 
4:   comparisons  $\leftarrow |R(s_i)|$ 
5: else Let FCount( $v_i, v_j$ ) =  $\sum_{x \in \tilde{R}(s_i), y \in \tilde{R}(s_j)} \mathbf{1}\{\mathcal{O}_c(v_i, x, v_j, y) == \text{Yes}\}$ 
6:   comparisons  $\leftarrow |\tilde{R}(s_i)| \cdot |\tilde{R}(s_j)|$ 
7: end if
8: if FCount( $v_i, v_j$ )  $< 0.3 \cdot$  comparisons then
9:   return No
10: else return Yes
11: end if

```

Assign-Final. After obtaining k clusters on the set of sampled points \tilde{V} , we assign the remaining points using **ACount** scores, similar to the one described in **ASSIGN**. For every point that is not sampled, we first assign it to $s_1 \in S$, and if $\text{ACount}(u, s_2, s_1) \geq 0.3|R(s_1)|$, we re-assign it to s_2 , and continue this process iteratively. After assigning all the points, the clusters are returned as output.

Theoretical Guarantees. Our algorithm first constructs a sample $\tilde{V} \subseteq V$ and runs the greedy algorithm on this sampled set of points. Our main idea to ensure that good approximation of the k -center objective lies in identifying a good *core* around each center. Using a sampling probability of $\gamma \log(n/\delta)/m$ ensures that we have at least $\Theta(\log(n/\delta))$ points from each of the optimal clusters in our sampled set \tilde{V} . By finding the closest points using **Count** scores, we identify $O(\log(n/\delta))$ points around every center that are in the optimal cluster. Essentially, this forms the core of each cluster. These cores are then used for robust pairwise comparison queries (similar to Section 5.4.1), in our **APPROX-FARTHEST** and **ASSIGN** subroutines. We give the following theorem, which guarantees a constant, i.e., $O(1)$ approximation with high probability.

Theorem 5. Given $p \leq 0.4$, a failure probability δ , and $m = \Omega(\log^3(n/\delta)/\delta)$. Then, Algorithm 18 achieves a $O(1)$ -approximation for the k -center objective using $O(nk \log(n/\delta) + \frac{n^2}{m^2} k \log^2(n/\delta))$ oracle queries with probability $1 - O(\delta)$.

In this proof, we first calculate the number of elements from different optimal clusters in \tilde{V} and then analyze the quality of clusters.

Lemma 10. Consider the sample $\tilde{V} \subseteq V$ of points obtained by selecting each point with a probability $\frac{450 \log(n/\delta)}{m}$. Then, we have $\frac{400n \log(n/\delta)}{m} \leq |\tilde{V}| \leq \frac{500n \log(n/\delta)}{m}$ and for every $i \in [k]$, $|C^*(s_i) \cap \tilde{V}| \geq 400 \log(n/\delta)$ with probability $1 - O(\delta)$ for sufficiently large $\gamma > 0$.

Proof. We include every point in \tilde{V} with a probability $\frac{450 \log(n/\delta)}{m}$ where the size of the smallest cluster is m . Using Chernoff bound, with probability $1 - O(\delta)$, we have :

$$\frac{400n \log(n/\delta)}{m} \leq |\tilde{V}| \leq \frac{500n \log(n/\delta)}{m}$$

Consider an optimal cluster $C^*(v_i)$ with center v_i . As every point is included with probability $\frac{450 \log(n/\delta)}{m}$:

$$\mathbf{E}[|C^*(s_i) \cap \tilde{V}|] = |C^*(s_i)| \cdot \frac{450 \log(n/\delta)}{m} \geq 450 \log(n/\delta)$$

Using Chernoff bound, with probability at least $1 - \delta/n$, we have

$$|C^*(s_i) \cap \tilde{V}| \geq 400 \log(n/\delta)$$

Using union bound for all the k clusters, we have the lemma. □

Assignment. We now analyze the quality of assignment in each iteration of the algorithm.

$$\mathbf{ACount}(u, s_i, s_j) = \sum_{x \in R(s_i)} \mathbf{1}\{\mathcal{O}_c(u, x, u, s_j) == \mathbf{Yes}\}$$

Lemma 11. Consider a point u and $s_j \neq s_i$ such that $d(u, s_i) \leq d(u, s_j) - 2 \text{OPT}$ and $|R(s_i)| \geq 12 \log(n/\delta)$, then, $\mathbf{ACount}(u, s_i, s_j) \geq 0.3|R(s_i)|$ with a probability of $1 - \frac{\delta}{n^2}$.

Proof. Using triangle inequality, for any $x \in R(s_i)$

$$d(u, x) \leq d(u, s_i) + d(s_i, x) \leq d(u, s_j) - 2 \text{OPT} + d(s_i, x) \leq d(u, s_j)$$

So, $\mathcal{O}_c(u, x, u, s_j)$ is **Yes** with a probability at least $1 - p$. We have:

$$\mathbf{E}[\mathbf{ACount}(u, s_i, s_j)] = \sum_{x \in R(s_i)} \mathbf{E}[\mathbf{1}\{\mathcal{O}_c(u, x, u, s_j) == \mathbf{Yes}\}] \geq (1 - p)|R(s_i)|$$

Using Hoeffding's inequality, with a probability of $\exp(-|R(s_i)|(1 - p)^2/2) \leq \frac{\delta}{n^2}$ (using $p \leq 0.4$), we have

$$\mathbf{ACount}(u, s_i, s_j) \leq (1 - p)|R(s_i)|/2$$

We have $\Pr[\mathbf{ACount}(u, s_i, s_j) \leq 0.3|S|] \leq \Pr[\mathbf{ACount}(u, s_i, s_j) \leq (1 - p)|S|/2]$. Therefore, with probability $\frac{\delta}{n^2}$, we have $\mathbf{ACount}(u, s_i, s_j) \leq 0.3|S|$. Hence, the lemma. \square

Lemma 12. Suppose $u \in C^*(s_i)$ and for some $s_j \in S$, if $d(s_i, s_j) \geq 6 \text{OPT}$, then, Algorithm 19 assigns u to center s_i with probability $1 - \frac{\delta}{n^2}$.

Proof. As $u \in C^*(s_i)$, we have $d(u, s_i) \leq 2 \text{OPT}$. Therefore,

$$d(s_j, u) - d(s_i, u) \geq d(s_i, s_j) - 2d(s_i, u) \geq 2 \text{OPT}$$

$$d(s_j, u) \geq d(s_i, u) + 2 \text{OPT}$$

From Lemma 11, we have that if $d(u, s_i) \leq d(u, s_j) - 2 \text{OPT}$, then, we will assign u to s_i with probability $1 - \frac{\delta}{n^2}$. \square

Lemma 13. *Given a set of centers S , every $u \in V$ is assigned to a cluster s_i such that $d(u, s_i) \leq \min_{s_j \in S} d(u, s_j) + 2 \text{OPT}$ with a probability of $1 - 1/n^2$.*

Proof. From Lemma 11, we have that a point u is assigned to s_l from s_m if $d(u, s_l) \leq d(u, s_m) - 2 \text{OPT}$. If s_i is the final assigned center of u , then, for every s_j , it must be true that $d(u, s_j) \geq d(u, s_i) - 2 \text{OPT}$, which implies $d(u, s_i) \leq \min_{s_j \in S} d(u, s_j) + 2 \text{OPT}$. Using union bound over at most n points, we have with a probability of $1 - \frac{\delta}{n}$, every point u is assigned as claimed. \square

Core Calculation. Consider a cluster $C(s_i)$ with center s_i . Let S_a^b denote the number of points in the set $|\{x : a \leq d(x, s_i) < b\}|$.

$$\text{Count}(u) = \sum_{x \in C(s_i)} \mathbf{1}\{\mathcal{O}_c(s_i, x, s_i, u) == \text{No}\}$$

Lemma 14. *Consider any two points $u_1, u_2 \in C(s_i)$ such that $d(u_1, s_i) \leq d(u_2, s_i)$, then $\mathbf{E}[\text{Count}(u_1)] - \mathbf{E}[\text{Count}(u_2)] = (1 - 2p)S_{d(u_1, s_i)}^{d(u_2, s_i)}$*

Proof. For a point $u \in C(s_i)$

$$\begin{aligned} \mathbf{E}[\text{Count}(u)] &= \mathbf{E} \left[\sum_{x \in C(s_i)} \mathbf{1}\{\mathcal{O}(s_i, x, s_i, u) == \text{No}\} \right] \\ &= S_0^{d(u, s_i)} p + S_{d(u, s_i)}^\infty (1 - p) \end{aligned}$$

$$\begin{aligned} \mathbf{E}[\text{Count}(u_1)] - \mathbf{E}[\text{Count}(u_2)] &= \left(S_0^{d(u_1, s_i)} p + S_{d(u_1, s_i)}^{d(u_2, s_i)} (1 - p) + S_{d(u_2, s_i)}^\infty (1 - p) \right) \\ &\quad - \left(S_0^{d(u_1, s_i)} p + S_{d(u_1, s_i)}^{d(u_2, s_i)} p + S_{d(u_2, s_i)}^\infty (1 - p) \right) \\ &= (1 - 2p) S_{d(u_1, s_i)}^{d(u_2, s_i)} \end{aligned}$$

\square

Lemma 15. Consider any two points $u_1, u_2 \in C(s_i)$ such that $d(u_1, s_i) \leq d(u_2, s_i)$ and $|S_{d(u_1, s_i)}^{d(u_2, s_i)}| \geq \sqrt{100|C(s_i)| \log(n/\delta)}$. Then, $\text{Count}(u_1) > \text{Count}(u_2)$ with probability $1 - \delta/n^2$.

Proof. Suppose $u_1, u_2 \in C(s_i)$. We have that $\text{Count}(u_1)$ and $\text{Count}(u_2)$ is a sum of $|C(s_i)|$ binary random variables.

Using Hoeffding's inequality, we have with probability $\exp(-\beta^2/2|C(s_i)|)$ that

$$\text{Count}(u_1) \leq \mathbf{E}[\text{Count}(u_1)] - \frac{\beta}{2}$$

$$\text{Count}(u_2) > \mathbf{E}[\text{Count}(u_2)] + \frac{\beta}{2}$$

Using union bound, with probability at least $1 - 2 \exp(-\beta^2/2|C(s_i)|)$, we can conclude that

$$\text{Count}(u_1) - \text{Count}(u_2) > \mathbf{E}[\text{Count}(u_1) - \text{Count}(u_2)] - \beta > (1 - 2p)S_{d(u_1, s_i)}^{d(u_2, s_i)} - \beta$$

Choosing $\beta = (1 - 2p)S_{d(u_1, s_i)}^{d(u_2, s_i)}$, we have $\text{Count}(u_1) > \text{Count}(u_2)$ with a probability (for constant $p \leq 0.4$)

$$1 - 2 \exp(-(1 - 2p)^2 \left(S_{d(u_1, s_i)}^{d(u_2, s_i)}\right)^2 / 2|C(s_i)|) \geq 1 - 2 \exp(-0.02 \left(S_{d(u_1, s_i)}^{d(u_2, s_i)}\right)^2 / |C(s_i)|).$$

Further, simplifying using $S_{d(u_1, s_i)}^{d(u_2, s_i)} \geq \sqrt{100|C(s_i)| \log(n/\delta)}$, we get probability of failure is $2 \exp(-2 \log(n/\delta)) = O(\delta/n^2)$ \square

Lemma 16. If $|C(s_i)| \geq 400 \log(n/\delta)$, then, $|R(s_i)| \geq 200 \log(n/\delta)$ with probability $1 - |C(s_i)|^2 \delta/n^2$.

Proof. From Lemma 15, we have that if there are points u_1, u_2 with $\sqrt{100|C(s_i)| \log(n/\delta)}$ many points between them, then, we can identify the closer one

correctly. When $|C(s_i)| \geq 400 \log(n/\delta)$, we have $\sqrt{100|C(s_i)| \log(n/\delta)} \geq 200 \log(n/\delta)$ points between every point and the point with the rank $200 \log(n/\delta)$. Therefore, $|R(s_i)| \geq 200 \log(n/\delta)$. Using union bound over all pairs of points in the cluster, we get the claim. \square

Lemma 17. *If $x \in C^*(s_i)$, then, $x \in C(s_i)$ or x is assigned to a cluster s_j such that $d(x, s_j) \leq 8 \text{OPT}$.*

Proof. If $x \in C^*(s_i)$, we argue that it will be assigned to $C(s_i)$. For the sake of contradiction, suppose x is assigned to a cluster $C(s_j)$ for some $s_j \in S$. We have $d(x, s_i) \leq 2 \text{OPT}$ and let $d(s_i, s_j) \geq 6 \text{OPT}$

$$d(s_i, s_j) \leq d(s_j, x) + d(s_i, x)$$

$$d(s_j, x) \geq 4 \text{OPT}$$

However, we know that $d(s_j, x) \leq d(s_i, x) + 2 \text{OPT} \leq 4 \text{OPT}$ from Lemma 11. We have a contradiction. Therefore, x is assigned to s_i . If $d(s_i, s_j) \leq 6 \text{OPT}$, we have $d(x, s_j) \leq d(x, s_i) + 2 \text{OPT} \leq 8 \text{OPT}$. Hence, the lemma. \square

Farthest point computation. Let $R(s_i)$ represent the core of the cluster $C(s_i)$ and contains $\Theta(\log(n/\delta))$ points. We define **FCOUNT** for comparing two points v_i, v_j from their centers s_i, s_j respectively. If $s_i \neq s_j$, we let :

$$\text{FCOUNT}(v_i, v_j) = \sum_{x \in \tilde{R}(s_i), y \in \tilde{R}(s_j)} \mathbf{1}\{\mathcal{O}_c(v_i, x, v_j, y) == \text{Yes}\}$$

Otherwise, we let $\text{FCOUNT}(v_i, v_j) = \sum_{x \in R(s_i)} \mathbf{1}\{\mathcal{O}_c(v_i, x, v_j, x) == \text{Yes}\}$. First, we observe that each of the summation is over $|R(s_i)|$ many terms, because $|\tilde{R}(s_i)| = \sqrt{|R(s_i)|}$.

Lemma 18. *Consider two records v_i, v_j in different clusters $C(s_i), C(s_j)$ respectively such that $d(s_i, v_i) < d(s_j, v_j) - 4 \text{ OPT}$ then $\mathbf{FCOUNT}(v_i, v_j) \geq 0.3|\tilde{R}(s_i)||\tilde{R}(s_j)|$ with a probability of $1 - \frac{\delta}{n^2}$.*

Proof. We know $\max_{v_i \in \tilde{R}(s_i)} d(u, v_i) \leq 2 \text{ OPT}$ and $\max_{v_j \in \tilde{R}(s_j)} d(v_j, s_j) \leq 2 \text{ OPT}$.

For a point $x \in R(s_i), y \in R(s_j)$

$$\begin{aligned} d(v_j, y) &\geq d(s_j, v_j) - d(s_j, y) \\ &> d(v_i, s_i) + 4 \text{ OPT} - d(s_j, y) \\ &> d(v_i, x) - d(x, s_i) + 4 \text{ OPT} - d(s_j, y) \\ &> d(v_i, x) \end{aligned}$$

So, $O(v_i, x, v_j, y)$ is **No** with a probability p . As $p \leq 0.4$, we have :

$$\mathbf{E}[\mathbf{FCOUNT}(v_i, v_j)] = (1 - p)|\tilde{R}(s_i)||\tilde{R}(s_j)|$$

$$\Pr[\mathbf{FCOUNT}(v_i, v_j) \leq 0.3|\tilde{R}(s_i)||\tilde{R}(s_j)|] \leq \Pr[\mathbf{FCOUNT}(v_i, v_j) \leq (1 - p)|\tilde{R}(s_i)||\tilde{R}(s_j)|/2]$$

From Hoeffding's inequality (with binary random variables), we have with a probability $\exp(-\frac{|\tilde{R}(s_i)||\tilde{R}(s_j)|(1-p)^2}{2}) \leq \frac{\delta}{n^2}$ (using $|\tilde{R}(s_i)||\tilde{R}(s_j)| \geq 12 \log(n/\delta)$, $p < 0.4$) : $\mathbf{FCOUNT}(v_i, v_j) \leq (1 - p)|\tilde{R}(s_i)||\tilde{R}(s_j)|/2$. Therefore, with probability at most δ/n^2 , we have, $\mathbf{FCOUNT}(v_i, v_j) \leq 0.3|\tilde{R}(s_i)||\tilde{R}(s_j)|$.

□

In order to calculate the farthest point, we use the ideas discussed in Section 5.4 to identify the point that has the maximum distance from its assigned center. As noted in Section 5.4.1, our approximation guarantees depend on the maximum distance of points in the core from the center. In the next lemma, we show that assuming a maximum distance of a point in the core (See Lemma 17), we can obtain a good approximation for the farthest point.

Lemma 19. *Let $\max_{s_j \in S, u \in R(s_j)} d(u, s_j) \leq \alpha$. In every iteration, if the farthest point is at a distance more than $(6\alpha + 3 \text{OPT})$, then, APPROX-FARTHEST outputs a $(6\alpha / \text{OPT} + 3)$ -approximation. Otherwise, the point output is at most $(6\alpha + 3 \text{OPT})$ away.*

Proof. The farthest point output APPROX-FARTHEST is a 6α additive approximation. However, the assignment of points to the cluster also introduces another additive approximation of 2OPT , resulting in a total $6\alpha + 2 \text{OPT}$ approximation. Suppose in the current iteration, the distance of the farthest point is βOPT , then the point output by APPROX-FARTHEST is at least $\beta \text{OPT} - (6\alpha + 2 \text{OPT})$ away. So, the approximation ratio is $\frac{\beta}{\beta - (6\alpha + 2 \text{OPT})}$. If $\beta \text{OPT} \geq 6\alpha + 3 \text{OPT}$, we have $\frac{\beta \text{OPT}}{\beta \text{OPT} - (6\alpha + 2 \text{OPT})} \leq \beta$. As we are trying to minimize the approximation ratio, we set $\beta \text{OPT} = 6\alpha + 3 \text{OPT}$ and get the claimed guarantee. \square

Final Guarantees. Throughout this section, we assume that $m = \Omega\left(\frac{\log^3(n/\delta)}{\delta}\right)$ for a given failure probability $\delta > 0$.

Lemma 20. *Given a current set of centers S , and $\max_{v_j \in S, u \in R(v_j)} d(u, v_j) \leq \alpha$, we have :*

1. *Every point u is assigned to a cluster $C(s_i)$ such that $d(u, s_i) \leq \min_{s_j \in S} d(u, s_j) + 2 \text{OPT}$ using $O(nk \log(n/\delta))$ oracle queries with probability $1 - O(\delta)$.*
2. *APPROX-FARTHEST identifies a point w in cluster $C(s_i)$ such that $\min_{v_j \in S} d(w, v_j) \geq \max_{v_j \in V} \min_{s_j \in S} d(v_j, s_j) / (6\alpha / \text{OPT} + 3)$ with probability $1 - O(\delta/k)$ using $O(|\tilde{V}| \log^3(n/\delta))$ oracle queries.*

Proof. (1) First, we argue that cores are calculated correctly. From Lemma 12, we have that a point $u \in C^*(s_i)$ is assigned to the center correctly s_i . Therefore, all the points from $\tilde{V} \cap C^*(S_i)$ move to $C(S_i)$. As the size of $|C(S_i)| \geq |\tilde{V} \cap C^*(S_i)| \geq 400 \log(n/\delta)$, we have $|R(s_i)| \geq 200 \log(n/\delta)$ with a

probability $1 - |C(s_i)|^2 \delta / n^2$ (From Lemma 15). Using union bound, we have that all the cores are calculated correctly with a failure probability of $\sum_i |C(s_i)|^2 / n^2 = \delta$.

For every point, we compare the distance with every cluster center by maintaining a center that is the current closest. From Lemma 11, we have that the query will fail with a probability of δ / n^2 . Using union bound, we have that the failure probability is $O(kn\delta / n^2) = \delta$. From Lemma 11, we have the approximation guarantee.

(2) From Lemma 19, we have our claim regarding the approximation guarantees. For APPROX-FARTHEST, we use the parameters $t = 2 \log(2k/\delta)$, $l = \sqrt{|\tilde{V}|}$. As we make $O(|\tilde{V}| \log^2(k/\delta))$ cluster comparisons using Algorithm CLUSTERCOMP (for APPROX-FARTHEST), we have that the total number of oracle queries is $O(|\tilde{V}| \log(n/\delta) \log^2(k/\delta)) = O(|\tilde{V}| \log^3(n/\delta))$. Using union bound, we have that the failure probability is $O(\delta/k + |\tilde{V}| \log^2(k/\delta) / n^2) = O(\delta/k)$. \square

Theorem 6. [Theorem 5 restated] Given $p \leq 0.4$, a failure probability δ , and $m = \Omega\left(\frac{\log^3(n/\delta)}{\delta}\right)$. Then, Algorithm 18 achieves a $O(1)$ -approximation for the k -center objective using $O(nk \log(n/\delta) + \frac{n^2}{m^2} k \log^2(n/\delta))$ oracle queries with probability $1 - O(\delta)$.

Proof. Using similar proof as Lemma 6, we have that the approximation ratio of Algorithm 18 is $4(6\alpha / \text{OPT} + 3) + 2$. Using $\alpha = 8 \text{OPT}$ from Lemma 17, we have that the approximation factor is 206. For the first stage, from Lemma 20, we have that for all the k iterations, the number of oracle queries is $O(|\tilde{V}| k \log^3(n/\delta))$. Using union bound over k iterations, success probability is $1 - O(\delta)$. For the calculation of *core*, the query complexity is $O(|\tilde{V}|^2 k)$. For assignment, the query complexity is $O(nk \log(n/\delta))$. Therefore, total query complexity is $O(nk \log(n/\delta) + \frac{n}{m} k \log^4(n/\delta) + \frac{n^2}{m^2} k \log^2(n/\delta)) = O(nk \log(n/\delta) + \frac{n^2}{m^2} k \log^2(n/\delta))$. \square

5.6 Hierarchical Clustering

In this section, we present robust algorithms for agglomerative hierarchical clustering using single linkage and complete linkage objectives. The naive algorithms initialize every record as a singleton cluster and merge the closest pair of clusters iteratively. For a set of clusters $\mathcal{C} = \{C_1, \dots, C_t\}$, the distance between any pair of clusters C_i and C_j , for single linkage clustering, is defined as the minimum distance between any pair of records in the clusters, $d_{SL}(C_1, C_2) = \min_{v_1 \in C_1, v_2 \in C_2} d(v_1, v_2)$. For complete linkage, cluster distance is defined as the maximum distance between any pair of records. All algorithms discussed in this section can be easily extended for complete linkage, and therefore we study single linkage clustering. The main challenge to implement single linkage clustering in the presence of *adversarial* noise is identification of minimum value in a list of at most $\binom{n}{2}$ distance values. In each iteration, the closest pair of clusters can be identified by using Algorithm 16 (with $t = 2 \log(n/\delta)$) to calculate the minimum over the set containing pairwise distances. For this algorithm, Lemma 21 shows that the pair of clusters merged in any iteration are a constant approximation of the optimal merge operation at that iteration. The proof of this lemma follows from Theorem 1.

Lemma 21. *Given a collection of clusters $\mathcal{C} = \{C_1, \dots, C_r\}$, our algorithm to calculate the closest pair (using Algorithm 16) identifies C_1 and C_2 to merge according to single linkage objective if $d_{SL}(C_1, C_2) \leq (1 + \mu)^3 \min_{C_i, C_j \in \mathcal{C}} d(C_i, C_j)$ with $1 - \delta$ probability and requires $O(r^2 \log^2(n/\delta))$ queries.*

Proof. In each iteration, our algorithm considers a list of $\binom{r}{2}$ distance values and calculates the closest using Algorithm 16. The claim follows from the proof of Theorem 1 □

Overview. Agglomerative clustering techniques are known to be inefficient. Each iteration of merge operation compares at most $\binom{n}{2}$ pairs of distance values and the

Algorithm 22 Greedy Algorithm

```
1: Input : Set of points  $V$ 
2: Output : Hierarchy  $H$ 
3:  $H \leftarrow \{\{v\} \mid v \in V\}, \mathcal{C} \leftarrow \{\{v\} \mid v \in V\}$ 
4: for  $C_i \in \mathcal{C}$  do
5:    $\widetilde{C}_i \leftarrow \text{NEARESTNEIGHBOR of } C_i \text{ among } \mathcal{C} \setminus \{C_i\} \text{ using Sec 5.4.1}$ 
6: end for
7: while  $|\mathcal{C}| > 1$  do
8:   Let  $(C_j, \widetilde{C}_j)$  be the closest pair among  $(C_i, \widetilde{C}_i), \forall C_i \in \mathcal{C}$ 
9:    $C' \leftarrow C_j \cup \widetilde{C}_j$ 
10:  Update Adjacency list of  $C'$  with respect to  $\mathcal{C}$ 
11:  Add  $C'$  as parent of  $\widetilde{C}_j$  and  $C_j$  in  $H$ .
12:   $\mathcal{C} \leftarrow (\mathcal{C} \setminus \{C_j, \widetilde{C}_j\}) \cup \{C'\}$ 
13:   $\widetilde{C}' \leftarrow \text{NEARESTNEIGHBOR of } C' \text{ from its adjacency list}$ 
14: end while
15: return  $H$ 
```

algorithm operates n times to construct the hierarchy. This yields an overall query complexity of $O(n^3)$. To improve their query complexity, **SLINK** algorithm [175] was proposed to construct the hierarchy in $O(n^2)$ comparisons. To implement this algorithm with a comparison oracle, for every cluster $C_i \in \mathcal{C}$, we maintain an adjacency list containing every cluster C_j in \mathcal{C} along with a pair of records with the distance equal to the distance between the clusters. For example, the entry for C_j in the adjacency list of C_i contains the pair of records (v_i, v_j) such that $d(v_i, v_j) = \min_{v_i \in C_i, v_j \in C_j} d(v_i, v_j)$. Algorithm 22 presents the pseudo code for single linkage clustering under the adversarial noise model. The algorithm is initialized with singleton clusters where every record is a separate cluster. Then, we identify the closest cluster for every $C_i \in \mathcal{C}$, and denote it by \widetilde{C}_i . This step takes n nearest neighbor queries, each requiring $O(n \log^2(n/\delta))$ oracle queries. In every subsequent iteration, we identify the closest pair of clusters (Using section 5.4.1), say C_j and \widetilde{C}_j from \mathcal{C} .

After merging these clusters, the data structure is updated as follows. To update the adjacency list, we need the pair of records with minimum distance between the merged cluster $C' \equiv C_j \cup \widetilde{C}_j$ and every other cluster $C_k \in \mathcal{C}$. In the previous iteration of the algorithm, we already have the minimum distance record pair for

(C_j, C_k) and (\widetilde{C}_j, C_k) . Therefore a single query between these two pairs of records is sufficient to identify the minimum distance edge between C' and C_k (formally: $d_{SL}(C_j \cup \widetilde{C}_j, C_k) = \min\{d_{SL}(C_j, C_k), d_{SL}(\widetilde{C}_j, C_k)\}$). The nearest neighbor of the merged cluster is identified by running a minimum calculation over its adjacency list. In Algorithm 22, as we identify the closest pair of clusters, each iteration requires $O(n \log^2(n/\delta))$ queries. As our Algorithm terminates in at most n iterations, it has an overall query complexity of $O(n^2 \log^2(n/\delta))$. In Theorem 7, we gave an approximation guarantee for every merge operation of Algorithm 22.

Theorem 7. *In any iteration, suppose the distance between a cluster $C_j \in \mathcal{C}$ and its identified nearest neighbor \widetilde{C}_j is α -approximation of its distance from the optimal nearest neighbor, then the distance between pair of clusters merged by Algorithm 22 is $\alpha(1 + \mu)^3$ approximation of the optimal distance between the closest pair of clusters in \mathcal{C} with a probability of $1 - \delta$ using $O(n \log^2(n/\delta))$ oracle queries.*

Proof. Algorithm 22 iterates over the list of pairs $(C_i, \widetilde{C}_i), \forall C_i \in \mathcal{C}$ and identifies the closest pair using Algorithm 16. The claim follows from the proof of Theorem 1 \square

Probabilistic Noise model. The above discussed algorithms do not extend to the probabilistic noise due to constant probability of error for each query. However, when we are given apriori, a partitioning of V into clusters of size $> \log n$ such that the maximum distance between any pair of records in every cluster is smaller than α (a constant), Algorithm 22 can be used to construct the hierarchy correctly. For this case, the algorithm to identify the closest and farthest pair of clusters is the same as the one discussed in Section 5.4.1.

Note that agglomerative clustering algorithms are known to require $\Omega(n^2)$ queries, which can be infeasible for million scale datasets. However, blocking based techniques present efficient heuristics to prune out low similarity pairs [159]. Devising provable algorithms with better time complexity is outside the scope of this work.

5.7 Experiments

This section evaluates the effectiveness of our techniques on various real world datasets and answers the following questions.

Q1: Is quadruplet oracle practically feasible? How do the different types of queries compare in terms of quality and time taken by annotators?

Q2: Are proposed techniques robust to different levels of noise in oracle answers?

Q3: How does the query complexity and solution quality of proposed techniques compare with optimum for varied levels of noise?

Datasets. We consider the following real-world datasets.

(1) **cities** dataset [10] comprises 36K cities of the United States. The different features of the cities include state, county, zip code, population, time zone, latitude and longitude.

(2) **caltech** dataset comprises 11.4K images from 20 categories. The ground truth distance between records is calculated using the hierarchical categorization described in [108].

(3) **amazon** dataset contains 7K images and textual descriptions collected from amazon.com [116]. For obtaining the ground truth distances we use Amazon’s hierarchical catalog.

(4) **monuments** dataset comprises 100 images belonging to 10 tourist locations around the world.

(5) **dblp** contains 1.8M titles of computer science papers from different areas [208]. From these titles, noun phrases were extracted and a dictionary of all the phrases was constructed. Euclidean distance in word2vec embedding space is considered as the ground truth distance between concepts.

Baselines. We compare our techniques with the optimal solution (whenever possible) and the following baselines. (a) **Tour2** constructs a binary tournament tree over the

entire dataset to compare the values and the root node corresponds to the identified maximum/minimum value (Algorithm 14 with $\lambda = 2$). This approach is an adaptation of the maximum calculation algorithm in [70] with a difference that each query is not repeated multiple times to increase success probability. We also use them to identify the farthest and nearest point in the greedy k -center Algorithm 12 and closest pair of clusters in hierarchical clustering. (b) **Samp** considers a sample of \sqrt{n} records and identifies the farthest/nearest by performing a quadratic number of comparisons over the sampled points using COUNT-MAX. For k -center, **Samp** considers a sample of $k \log n$ points to identify k centers over these samples using the greedy algorithm. It then assigns all the remaining points to the identified centers by querying each record with every pair of centers.

Calculating optimal clustering objective for k -center is NP-hard even in the presence of accurate pairwise distance [204]. So, we compare the solution quality with respect to the greedy algorithm on the ground truth distances, denoted by **TDist**. For farthest, nearest neighbor and hierarchical clustering, **TDist** denotes the optimal technique that has access to ground truth distance between records.

Our algorithm is labelled **Far** for farthest identification, **NN** for nearest neighbor, **kC** for k -center and **HC** for hierarchical clustering with subscript a denoting the adversarial model and p denoting the probabilistic noise model. All algorithms were implemented in C++ and run on a server with 64GB RAM. The reported results are averaged over 100 randomly chosen iterations. Unless specified, we set $t = 1$ in Algorithm 16 and $\gamma = 2$ in Algorithm 18.

Evaluation Metric. For finding maximum and nearest neighbors, we compare different techniques by evaluating the true distance of the returned solution from the queried points. For k -center, we use the objective value, i.e., maximum radius of the returned clusters as the evaluation metric and compare against the true greedy algorithm (**TDist**) and other baselines. For datasets where ground truth clusters are

known (`amazon`, `caltech` and `monuments`), we use F-score over intra-cluster pairs for comparing it with the baselines [87]. For hierarchical clustering, we compute the pairs of clusters merged in every iteration and compare the average true distance between these clusters. In addition to the quality of the returned solution, we compare the query complexity and running time of the proposed techniques with the baselines described above.

Noise Estimation. For `cities`, `amazon`, `caltech`, and `monuments` datasets, we ran a user study on Amazon Mechanical Turk to estimate the noise in oracle answers over a small sample of the dataset, often referred to as the validation set. Using crowd responses, we trained a classifier (random forest [184] obtained the best results) using active learning to act as the quadruplet oracle, and reduce the number of queries to the crowd. Our active learning algorithm [183] uses a batch of 20 queries and we stop it when the classifier accuracy on the validation set does not improve by more than 0.01 [103]. To efficiently construct a small set of candidates for active learning and pruning low similarity pairs for `dblp`, we employ token based blocking [159] for the datasets. For the synthetic oracle, we simulate a quadruplet oracle with different values of the noise parameters.

5.7.1 User study

In this section, we evaluate the users ability to answer quadruplet queries and compare it with other types of queries.

Setup. We ran a user study on Amazon Mechanical Turk platform for four datasets `cities`, `amazon`, `caltech` and `monuments`. We consider the ground truth distance between record pairs and discretize them into buckets, and assign a pair of records to a bucket if the distance falls within its range. For every pair of buckets, we query a random subset of $\log n$ quadruplet oracle queries (where n is size of dataset). Each

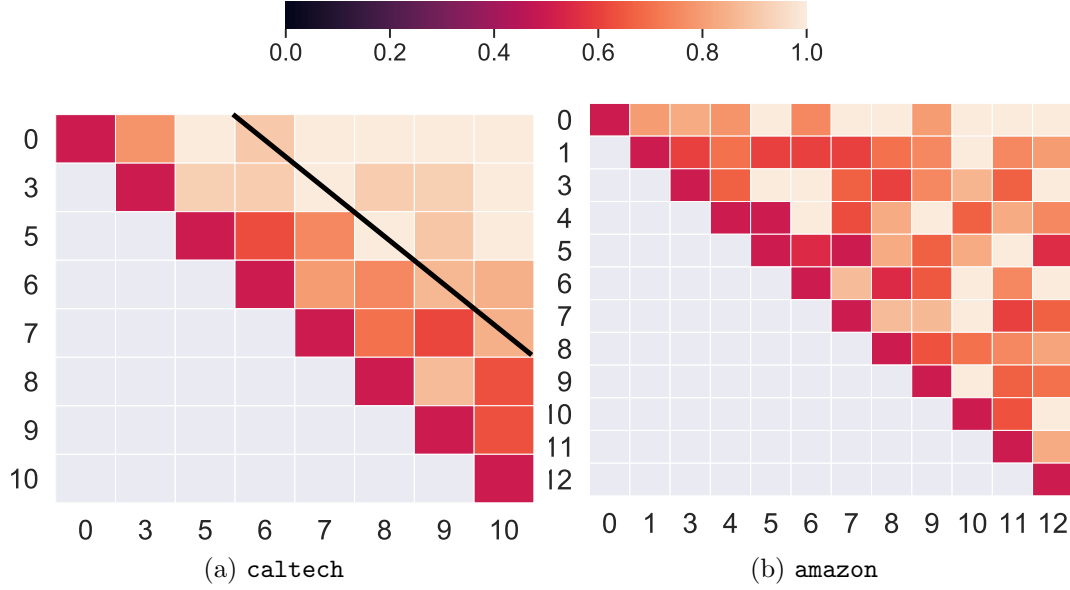


Figure 5.4: Accuracy values (denoted by the color of a cell) for different distance ranges observed during our user study. The diagonal entries refer to the quadruplets with similar distance between the corresponding pairs and the distance increases as we go further away from the diagonal.

query is answered by three different crowd workers and a majority vote is taken as the answer to the query.

Qualitative Analysis of Oracle. In Figure 5.4, for every pair of buckets, using a heat map, we plot the accuracy of answers obtained from the crowd workers for quadruplet queries. For all datasets, average accuracy of quadruplet queries is more than 0.83 and the accuracy is minimum whenever both pairs of records belong to the same bucket (as low as 0.5). However, we observe varied behavior across datasets as the distance between considered pairs increases.

For the `caltech` dataset, we observe that when the ratio of the distances is more than 1.45 (indicated by a black line in the Figure 5.4a), there is no noise (or close to zero noise) observed in the query responses. As we observe a sharp decline in noise as the distance between the pairs increases, it suggests that adversarial noise is satisfied for this dataset. We observe a similar pattern for the `cities` and `monuments` datasets. For the `amazon` dataset, we observe that there is substantial noise across all distance

ranges (See Figure 5.4b) rather than a sharp decline, suggesting that the probabilistic model is satisfied.

Comparison with Pairwise Querying Mechanisms. To evaluate the benefit of quadruplet queries, we compare the quality of quadruplet comparison oracle answers with the following pairwise oracle query models. (a) Optimal cluster query: This query asks questions of type ‘do u and v refer to the same/similar type?’. (b) Distance query: How similar are the records x and y ? In this query, the annotator scores the similarity of the pair within 1 to 10.

We make the following observations. (i) Optimal cluster queries are answered correctly only if the ground truth clusters refer to different entities (each cluster referring to a distinct entity). Crowd workers tend to answer ‘No’ if the pair of records refer to different entities. Therefore, we observe high precision (more than 0.90) but low recall (0.50 on `amazon` and 0.30 on `caltech` for $k = 10$) of the returned labels. (ii) We observed very high variance in the distance estimation query responses. For all record pairs with identical entities, the users returned distance estimates that were within 20% of the correct distances. In all other cases, we observe the estimates to have errors of upto 50%. We provide more detailed comparison on the quality of clusters identified by pairwise query responses along with quadruplet queries in the next section.

5.7.2 Crowd Oracle: Solution Quality & Query Complexity

In this section, we compare the quality of our proposed techniques for the datasets on which we performed the user study. Following the findings of Section 5.7.1, we use the probabilistic model based algorithm for `amazon`(with $p = 0.50$) and adversarial noise model based algorithm for `caltech`, `monuments` and `cities`.

Finding Max and Farthest/Nearest Neighbor. Figure 5.5 compares the quality of farthest and nearest neighbor (NN) identified by proposed techniques along with

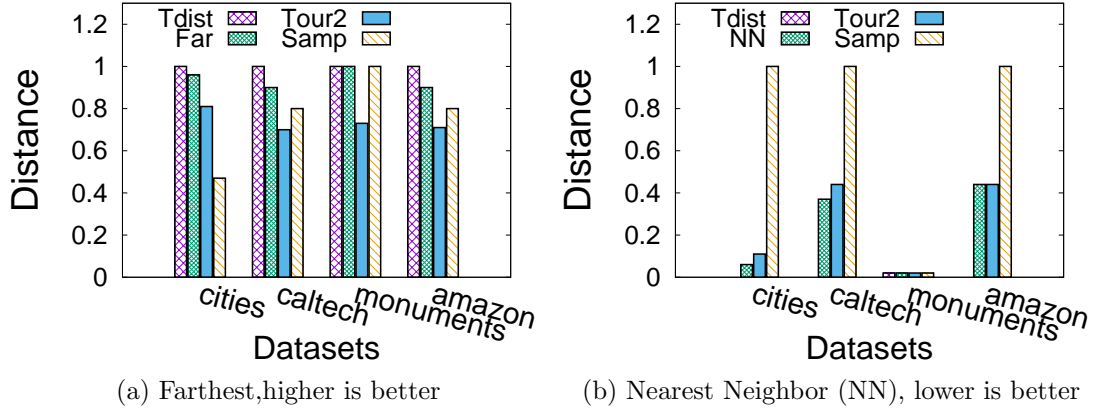


Figure 5.5: Comparison of farthest and NN techniques for crowdsourced oracle queries.

other baselines. The values are normalized according to the maximum value to present all datasets on the same scale. Across all datasets, the point identified by **Far** and **NN** is closest to the optimal value, **TDist**. In contrast, the farthest returned by **Tour2** is better than that of **Samp** for **cities** dataset but not for **caltech**, **monuments** and **amazon**. We found that this difference in quality across datasets is due to varied distance distribution between pairs. The **cities** dataset has a skewed distribution of distance between record pairs, leading to a unique optimal solution to the farthest/NN problem. Due to this reason, the set of records sampled by **Samp** does not contain any record that is a good approximation of the optimal farthest. However, ground truth distances between record pairs in **amazon**, **monuments** and **caltech** are less skewed with more than $\log n$ records satisfying the optimal farthest point for all queries. Therefore, **Samp** performs better than **Tour2** on these datasets. We observe **Samp** performs worse for NN because our sample does not always contain the closest point.

k -center Clustering. We evaluate the F-score² of the clusters generated by our techniques along with baselines and techniques for pairwise optimal query mechanism

²Optimal clusters are identified from the original source of the datasets (**amazon** and **caltech**) and manually for **monuments**.

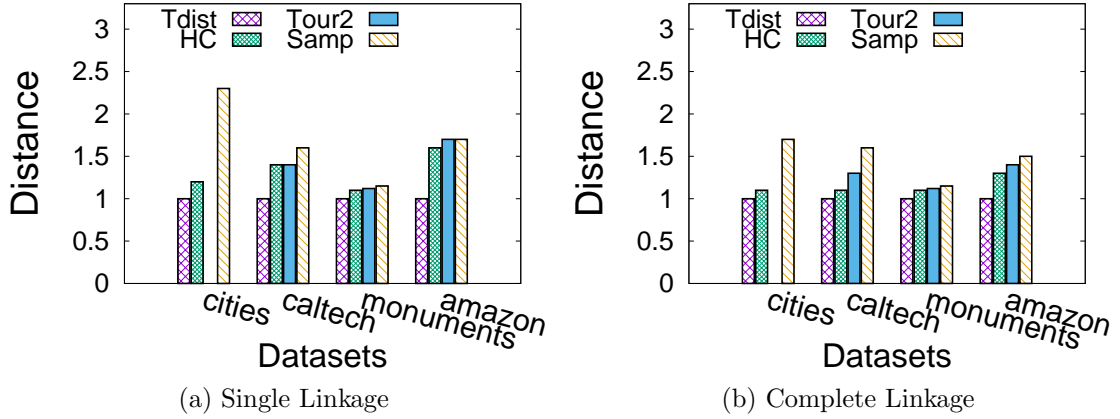


Figure 5.6: Comparison of Hierarchical clustering techniques with crowdsourced oracle.

(denoted as $0q$)³. Table 5.1 presents the summary of our results for different values of k . Across all datasets, our technique achieves more than 0.90 F-score. On the other hand, **Tour2** and **Samp** do not identify the ground truth clusters correctly, leading to low F-score. Similarly, $0q$ achieves poor recall (and hence low F-score) as it labels many record pairs to belong to separate clusters. For example, a *frog* and a *butterfly* belong to the same optimal cluster for **caltech** ($k=10$) but the two records are assigned to different clusters by $0q$.

Hierarchical Clustering. Figure 5.6 compares the average distance of the merged clusters across different iterations of the agglomerative clustering algorithm. **Tour2** has $O(n^3)$ complexity and does not run for **cities** dataset in less than 48 hrs. The objective value of different techniques are normalized by the optimal value with **Tdist** denoting 1. For all datasets, **HC** performs better than **Samp** and **Tour2**. Among datasets, the quality of hierarchies generated for **monuments** is similar for all techniques due to low noise.

³We report the results on the sample of queries asked to the crowd as opposed to training a classifier because the classifier generates noisier results and has poorer F-score than the quality of labels generated by crowdsourcing

| Technique | kC | Tour2 | Samp | 0q* |
|-----------------------|-------------|-------|------|------|
| caltech ($k = 10$) | 1 | 0.88 | 0.91 | 0.45 |
| caltech ($k = 15$) | 1 | 0.89 | 0.88 | 0.49 |
| caltech ($k = 20$) | 0.99 | 0.93 | 0.87 | 0.58 |
| monuments ($k = 5$) | 1 | 0.95 | 0.97 | 0.77 |
| amazon ($k = 7$) | 0.96 | 0.74 | 0.57 | 0.48 |
| amazon ($k = 14$) | 0.92 | 0.66 | 0.54 | 0.72 |

Table 5.1: F-score comparison of k-center clustering. 0q is marked with * as it was computed on a sample of 150 pairwise queries to the crowd³. All other techniques were run on the complete dataset using a classifier.

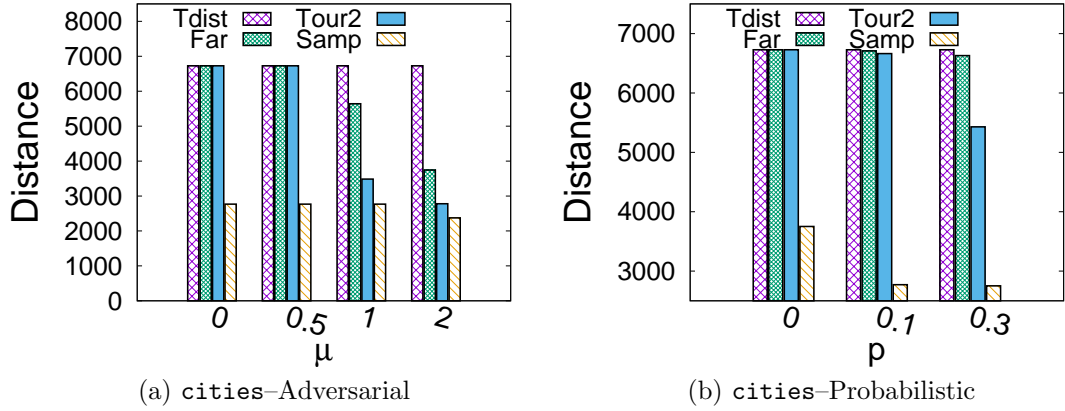


Figure 5.7: Comparison of farthest identification techniques for adversarial and probabilistic noise models.

Query Complexity. To ensure scalability, we trained active learning based classifier for all the aforementioned experiments. In total, **amazon**, **cities**, and **caltech** required 540 (cost: \$32.40), 220 (cost: \$13.20) and 280 (cost: \$16.80) queries to the crowd respectively.

5.7.3 Simulated Oracle: Solution Quality & Query Complexity

In this section, we compare the robustness of the techniques where the query response is simulated synthetically for given μ and p .

Finding Max and Farthest/Nearest Neighbor. In Figure 5.7a, $\mu = 0$ denotes the setting where the oracle answers all queries correctly. In this case, **Far** and **Tour2** identify the optimal solution but **Samp** does not identify the optimal solution for

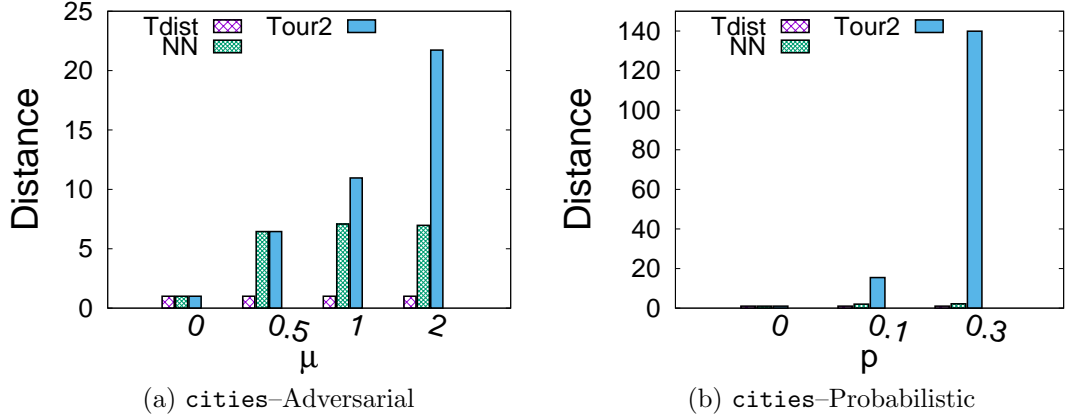


Figure 5.8: Comparison of nearest neighbor techniques for adversarial and probabilistic noise model (lower is better).

cities. In both datasets, **Far** identifies the correct farthest point for $\mu < 1$. Even with an increase in noise (μ), we observe that the farthest is always at a distance within 4 times the optimal distance (See Fig 5.7a). We observe that the quality of farthest identified by **Tour2** is close to that of **Far** for smaller μ because the optimal farthest point v_{\max} has only a few points in the confusion region C (See Section 5.4) that contains the points that are close to v_{\max} . For e.g., less than 10% are present in C when $\mu = 1$ for **cities** dataset, i.e., less than 10% points return an erroneous answer when compared with v_{\max} .

In Figure 5.7b, we compare the true distance of the identified farthest points for the case of probabilistic noise with error probability p . We observe that **Far_p** identifies points with distance values very close to the farthest distance **Tdist**, across all data sets and error values. This shows that **Far** performs significantly better than the theoretical approximation presented in Section 5.4. On the other hand, the solution returned by **Samp** is more than $4\times$ smaller than the value returned by **Far_p** for an error probability of 0.3. **Tour2** has a similar performance as that of **Far_p** for $p \leq 0.1$, but we observe a decline in solution quality for higher noise (p) values.

In Figures 5.8a, 5.8b, we compare the true distance of the identified nearest neighbor with different baselines. **NN** shows superior performance as compared to **Tour2**

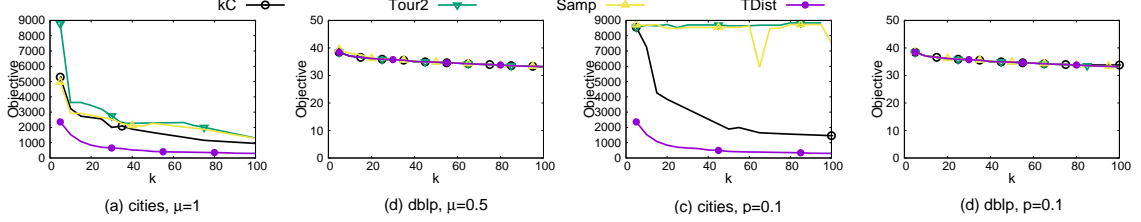


Figure 5.9: k -center clustering objective comparison for adversarial and probabilistic noise model.

across all error values. This justifies the lack of robustness of **Tour2** as discussed in Section 5.4. The solution quality of **NN** does not worsen with increase in error. We omit **Samp** from the plots because the returned points had very poor performance (as bad as 700 even in the absence of error). We observed similar behavior for other datasets. In terms of query complexity, **NN** requires around 53×10^3 queries for **cities** dataset and the number of queries grows linearly with the dataset size. Among baselines, **Tour2** uses 37×10^3 queries and **Samp** uses 18×10^3 .

*“In conclusion, we observe that our techniques achieve the best quality across all data sets and error values, while **Tour2** performs similar to **Far** for low error, and its quality degrades with increasing error.”*

k -center Clustering. Figure 5.9 compares the k -center objective of the returned clusters for varying k in the adversarial and probabilistic noise model. **Tdist** denotes the best possible clustering objective, which is guaranteed to be a 2-approximation of the optimal objective. The set of clusters returned by **kC** are consistently very close to **TDist** across all datasets, validating the theory. For higher values of k , **kC** approaches closer to **TDist**, thereby improving the approximation guarantees. The quality of clusters identified by **kC** are similar to that of **Tour2** and **Far** for adversarial noise (Figure 5.9a,b) but considerably better for probabilistic noise (Figure 5.9c,d).

Running time. Table 5.2 compares the running time and the number of required quadruplet comparisons for various problems under adversarial noise model with $\mu = 1$ for the largest **dblp** dataset. **Far** and **NN** requires less than 6 seconds for both

| Problem | Our Approach | | Tour2 | | Samp | |
|------------------|--------------|--------|-------|--------|------|--------|
| | Time | # Comp | Time | # Comp | Time | # Comp |
| Farthest | 0.1 | 2.2M | 0.06 | 2M | 0.07 | 1M |
| Nearest | 0.075 | 2M | 0.07 | 2M | 0.61 | 1M |
| kC (k=50) | 450 | 120M | 375.3 | 95M | 477 | 105M |
| Single Linkage | 1813 | 990M | DNF | | 1760 | 940M |
| Complete Linkage | 1950 | 940M | DNF | | 1940 | 920M |

Table 5.2: Running time (in minutes) and number of quadruplet comparisons (denoted by # Comp, in millions) of different techniques for **dblp** dataset under the adversarial noise model with $\mu = 1$. DNF denotes ‘did not finish’.

adversarial and probabilistic error models. Our k -center clustering technique requires less than 450 min to identify 50 centers for **dblp** dataset across different noise models; the running time grows linearly with k . While the running time of our algorithms are slightly higher than **Tour2** for farthest, nearest and k -center, **Tour2** did not finish in 48 hrs due to $O(n^3)$ running time for single and complete linkage hierarchical clustering. We observe similar performance for the probabilistic noise model. Note that even though the number of comparisons are in millions, this dataset requires only 740 queries to the crowd workers to train the classifier.

5.8 Summary and Future Work

This chapter formalizes the notion of a noisy comparison oracle and develops robust techniques to perform metric based clustering. The presented oracle models are validated by running case studies on Amazon Mechanical Turk. The proposed technique is demonstrated to be highly effective in recovering ground truth summaries over these datasets. The key takeaways from the chapter are summarized below.

- Quadruplet comparison queries are easy to answer when the two distances being compared are well separated. Crowdworkers (oracle in general) can provide accurate knowledge about the relative ordering of distance values.
- k -center clustering technique scales linearly with n and achieves a constant approximation.

- Empirically, the proposed algorithms generate accurate clusters even in the presence of noise.

Future Work. In this chapter, we studied data summarization assuming that the ground truth clusters are fixed with respect to a given distance function. However, a dataset may have different summaries based on the application at hand [140]. For example, a traveler who is planning to visit different tourist locations maybe interested in clustering based on geographical distance between records. However, an artist who is looking for different types of pictures to paint maybe interested in clustering them based on architecture. Extending our techniques to generate a personalized data summary is an interesting direction for further exploration.

Additionally, the presented techniques do not use pairwise similarity values (calculated by automated techniques) to optimize for the oracle queries. It would be interesting to design techniques that consider noisy similarity values (calculated using automated techniques like jaccard similarity over textual features) to guide the oracle querying procedure. Extending the proposed techniques for other metric-based clustering algorithms like k-median and k-means clustering is also an interesting direction for future research. The presented oracle strategy presents a mechanism to compare ground truth distance between pairs of records. In terms of other applications, it would be interesting to consider comparison oracle-based supervision for various NLP tasks like sentiment prediction, information extraction, etc.

PART II: GENERATIVE MODELS FOR CLUSTERING

CHAPTER 6

CLUSTERING WITH GENERATIVE MODELS

In this chapter, we propose the geometric block model, a novel generative model that captures correlated edge formation. This generative model is validated on two different real-world datasets. In order to recover ground truth clusters, we propose a simple triangle counting-based algorithm and analyze its efficacy.

Section 6.2 defines the geometric block model (GBM) and the cluster recovery problem. Section 6.3 verifies our hypothesis about GBM on academic collaboration networks and Amazon co-purchase networks. Section 6.4 proposed the motif-counting algorithm to recover clusters and Section 6.5 analyzes its quality. Section 6.6 empirically evaluates the quality of proposed techniques on three real-world datasets.

6.1 Introduction

Graph Clustering consists of partitioning the vertices into clusters that refer to similar type of entities. Clustering forms one of the important problems in machine learning and data mining with applications in data integration, outlier detection, community detection, medical analysis, among others. Due to the lack of ground truth in most applications, clustering techniques have been studied to optimize for an objective hoping to identify meaningful communities. This motivated the study of generative models to better understand the interactions between nodes of different clusters and benchmark clustering techniques on these models.

The *planted-partition* model or the *stochastic block model* (SBM) is a random graph model for community detection that generalizes the well-known Erdős-Renyi graphs

[119, 77, 71, 13, 12, 115, 56, 150]. It is one of the most popular models to study graph clustering. Consider a graph $G(V, E)$, where $V = C_1 \sqcup C_2 \sqcup \dots \sqcup C_k$ is a disjoint union of k clusters denoted by C_1, \dots, C_k . The edges of the graph are drawn randomly: there is an edge between $u \in C_i$ and $v \in C_j$ with probability $q_{i,j}$, $1 \leq i, j \leq k$. Given the adjacency matrix of such a graph, the task is to find exactly (or approximately) the partition $C_1 \sqcup C_2 \sqcup \dots \sqcup C_k$ of V .

Recent theoretical works focus on characterizing a sharp threshold of recovering the partition in the SBM. For example, when there are only two communities of exactly equal sizes, and the inter-cluster edge probability is $\frac{b \log n}{n}$ and intra-cluster edge probability is $\frac{a \log n}{n}$, it is known that perfect recovery is possible if and only if $\sqrt{a} - \sqrt{b} > \sqrt{2}$ [12, 150]. The regime of the probabilities being $\Theta\left(\frac{\log n}{n}\right)$ has been put forward as one of the most interesting ones because in an Erdős-Renyi random graph, this is the threshold for graph connectivity [40]. This result has been subsequently generalized for k communities [13, 14, 114] (for constant k or when $k = o(\log n)$), and under the assumption that the communities are generated according to a probabilistic generative model (there is a prior probability p_i of an element being in the i th community) [13]. Note that, the results are not only of theoretical interest, many real-world networks exhibit a “sparsely connected” community feature [138], and any efficient recovery algorithm for SBM has many potential applications.

One aspect that the SBM does not account for is a “transitivity rule” (‘friends having common friends’) inherent to many social and other community structures. To be precise, consider any three vertices x, y and z . If x and y are connected by an edge (or they are in the same community), and y and z are connected by an edge (or they are in the same community), then it is more likely than not that x and z are connected by an edge. This phenomenon can be seen in many network structures - predominantly in social networks, blog-networks and advertising. SBM, primarily a generalization of Erdős-Renyi random graph, does not consider this characteristic,

and in particular, probability of an edge between x and z there is independence of the fact that there exist edges between x and y and y and z . However, one needs to be careful such that by allowing such “transitivity”, the simplicity and elegance of the SBM is not lost.

Inspired by the above question, we propose a random graph community detection model analogous to the stochastic block model, that we call the *geometric block model* (GBM). The GBM depends on the basic definition of the *random geometric graph* that has found a lot of practical use in wireless networking because of its inclusion of the notion of proximity between nodes [162].

Definition 11 (Random Geometric Graph). *A random geometric graph (RGG) on n vertices has parameters n , an integer $t > 1$ and a real number $\beta \in [-1, 1]$. It is defined by assigning a vector $Z_i \in \mathbb{R}^t$ to vertex i , $1 \leq i, n$, where Z_i , $1 \leq i \leq n$ are independent and identical random vectors uniformly distributed in the Euclidean sphere $\mathcal{S}^{t-1} \equiv \{x \in \mathbb{R}^t : \|x\|_{\ell_2} = 1\}$. There will be an edge between vertices i and j if and only if $\langle Z_i, Z_j \rangle \geq \beta$.*

Note that, the definition can be further generalized by considering Z_i s to have a sample space other than \mathcal{S}^{t-1} , and by using a different notion of distance than inner product (i.e., the Euclidean distance). We simply stated one of the many equivalent definitions [48].

Random geometric graphs are often proposed as an alternative to Erdős-Renyi random graphs. They are quite well studied theoretically (though not nearly as much as the Erdős-Renyi graphs), and very precise results exist regarding their connectivity, clique numbers and other structural properties [112, 163, 73, 26, 102]. For a survey of early results on geometric graphs and the analogy to results in Erdős-Renyi graphs, we refer the reader to [162]. A very interesting question of distinguishing an Erdős-Renyi graph from a geometric random graph has also recently been studied [48]. This will

provide a way to test between the models which better fits a scenario, a potentially great practical use.

As mentioned earlier, the “transitivity” feature led to random geometric graphs being used extensively to model wireless networks (for example, see [113, 37]). Surprisingly, however, to the best of our knowledge, random geometric graphs are never used to model community detection problems. In this chapter we take the first step towards this direction. The chapter is organized as follows.

- We define a random generative model (Section 6.2) to study canonical problems of community detection, called the *geometric block model* (GBM). This model takes into account a measure of proximity between nodes and this proximity measure characterizes the likelihood of two nodes being connected when they are in the same or different communities. The geometric block model inherits the connectivity properties of the random geometric graphs, in particular the likelihood of “transitivity” in triplet nodes (or more).
- We experimentally validate the GBM on various real-world datasets (Section 6.3). We show that many practical community structures exhibit properties of the GBM. We also compare these features with the corresponding notions in SBM to show how GBM better models data in many practical situations.
- We propose a simple motif-based efficient algorithm for community detection on the GBM (Section 6.4). We rigorously show that this algorithm is optimal up to a constant fraction (to be properly defined later) even in the regime of sparse graphs (average degree $\sim \log n$).
- The motif-counting algorithms are extensively tested on both synthetic and real-world datasets. They exhibit very good performance in three real datasets, compared to the spectral-clustering algorithm (see Section 6.6). Since simple motif-counting is known to be far from optimum in stochastic block model (see

Section 6.5), these experiments give further validation to GBM as a real-world model.

Given any simple random graph model, it is possible to generalize it to a random block model of communities much in line with the SBM. We, however, stress that the geometric block model is perhaps the simplest possible model of real-world communities that also captures the transitive/geometric features of communities. Moreover, the GBM explains behaviors of many real-world networks as we will exemplify subsequently.

6.2 The Geometric Block Model

Let $V \equiv C_1 \sqcup C_2 \sqcup \cdots \sqcup C_k$ be the set of vertices that is a disjoint union of k clusters, denoted by C_1, \dots, C_k . Given an integer $t \geq 2$, for each vertex $u \in V$, define a random vector $Z_u \in \mathbb{R}^t$ that is uniformly distributed in $\mathcal{S}^{t-1} \subset \mathbb{R}^t$, the $t - 1$ -dimensional sphere.

Definition 12 (Geometric Block Model $(V, t, \beta_{i,j}, 1 \leq i < j \leq k)$). *Given V, t and a set of real numbers $\beta_{i,j} \in [-1, 1], 1 \leq i < j \leq k$, the geometric block model is a random graph with vertices V and an edge exists between $v \in C_i$ and $u \in C_j$ if and only if $\langle Z_u, Z_v \rangle \geq \beta_{i,j}$.*

The case of $t = 2$: In this chapter we particularly analyze our algorithm for $t = 2$. In this special case, the above definition is equivalent to choosing random variable θ_u uniformly distributed in $[0, 2\pi]$, for all $u \in V$. Then there will be an edge between two vertices $u \in C_i, v \in C_j$ if and only if $\cos \theta_u \cos \theta_v + \sin \theta_u \sin \theta_v = \cos(\theta_u - \theta_v) \geq \beta_{i,j}$ or $\min\{|\theta_u - \theta_v|, 2\pi - |\theta_u - \theta_v|\} \leq \arccos \beta_{i,j}$. This in turn, is equivalent to choosing a random variable X_u uniformly distributed in $[0, 1]$ for all $u \in V$, and there exists an edge between two vertices $u \in C_i, v \in C_j$ if and only if

$$d_L(X_u, X_v) \equiv \min\{|X_u - X_v|, 1 - |X_u - X_v|\} \leq r_{i,j},$$

where $r_{i,j} \in [0, \frac{1}{2}]$, $0 \leq i, j \leq k$, are a set of real numbers.

For the rest of this chapter, we concentrate on the case when $r_{i,i} = r_s$ for all $i \in \{1, \dots, k\}$, which we call the “intra-cluster distance” and $r_{i,j} = r_d$ for all $i, j \in \{1, \dots, k\}, i \neq j$, which we call the “inter-cluster distance,” mainly for the clarity of exposition. To allow for edge density to be higher inside the clusters than across the clusters, assume $r_s \geq r_d$.

The main problem that we seek to address is the following.

Problem 4. *Given the adjacency matrix of a geometric block model with k clusters, and t, r_d, r_s , $r_s \geq r_d$, find the partition C_1, C_2, \dots, C_k .*

6.3 Real-world Validation

We experiment with two different types of real-world datasets to verify our hypothesis about geometric block model and the role of distance in the formation of edges. The first one is a dataset with academic collaboration, and the second one is a product purchase dataset from Amazon.

6.3.1 Academic Collaboration Network

We consider the collaboration network of academicians in Computer Science in 2016 (data obtained from csrankings.org). According to the area of expertise of the authors, we consider five different communities: Data Management (MOD), Machine Learning and Data Mining (ML), Artificial Intelligence (AI), Robotics (ROB), Architecture (ARCH). If two authors share the same affiliation, or shared affiliation in the past, we assume that they are geographically close.

We would like to hypothesize that two authors in the same communities might collaborate even when they are geographically far. However, two authors in different communities are more likely to collaborate only if they share the same affiliation (or are geographically close). Table 6.1 describes the number of edges across the

Table 6.1: On the left we count the number of inter-cluster edges when authors shared the same affiliation and different affiliations. On the right, we count the same for intra-cluster edges.

| Area 1 | Area 2 | same | different | Area | same | different |
|--------|--------|------|-----------|------|------|-----------|
| MOD | AI | 10 | 2 | MOD | 19 | 35 |
| ARCH | MOD | 6 | 1 | ARCH | 13 | 15 |
| ROB | ARCH | 3 | 0 | ROB | 24 | 16 |
| MOD | ROB | 4 | 0 | AI | 39 | 32 |
| ML | MOD | 7 | 1 | ML | 14 | 42 |

communities. It is evident that the authors from the same community are likely to collaborate irrespective of the affiliations and the authors of different communities collaborate much frequently when they share affiliations or are close geographically. This clearly indicates that the inter cluster edges are likely to form if the distance between the nodes is quite small, motivating the fact $r_d < r_s$ in the GBM.

6.3.2 Amazon Co-purchase Network

The next dataset that we use in our experiments is the Amazon product meta-data on SNAP (<https://snap.stanford.edu/data/amazon-meta.html>), that has 548552 products and each product is one of the following types {Books, Music CD’s, DVD’s, Videos}. Moreover, each product has a list of attributes, for example, a book may have attributes like {“General”, “Sermon”, “Preaching”}. We consider the co-purchase network over these products. We make two observations here: (1) edges get formed (that is items are co-purchased) more frequently if they are similar, where we measure similarity by the number of common attributes between products, and (2) two products that share an edge have more common neighbors (no of items that are bought along with both those products) than two products with no edge in between.

Figures 6.1 and 6.2 show average similarity of products that were bought together, and not bought together. From the distribution, it is quite evident that edges in a co-purchase network gets formed according to distance, a salient feature of random geometric graphs, and the GBM.

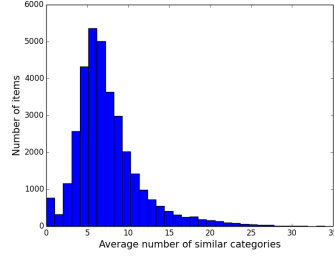


Figure 6.1: Histogram: similarity of products bought together (mean ≈ 6)

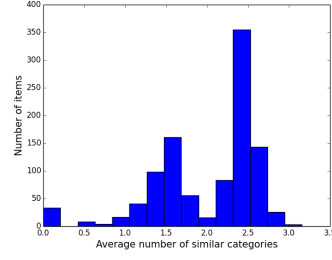


Figure 6.2: Histogram: similarity of products not bought together (mean ≈ 2)

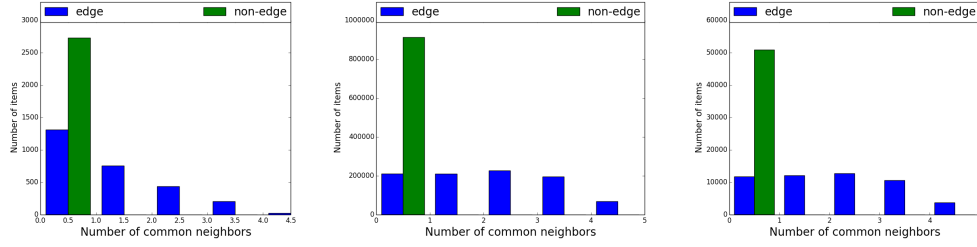


Figure 6.3: Histogram of common neighbors of edges and non-edges in the co-purchase network, from left to right: Book-DVD, Book-Book, DVD-DVD

We next take equal number of product pairs inside Book (also inside DVD, and across Book and DVD) that have an edge in-between and do not have an edge respectively. Figure 6.3 shows that the number of common neighbors when two products share an edge is much higher than when they do not—in fact, almost all product pairs that do not have an edge in between also do not share any common neighbor. This again strongly suggests towards GBM due to its transitivity property. On the other hand, this also suggests that SBM is not a good model for this network, as in SBM, two nodes having common neighbors is independent of whether they share an edge or not.

Difference between SBM and GBM. It is important to stress that the network structures generated by the SBM and the GBM are quite different, and it is significantly difficult to analyze any algorithm or lower bound on GBM compared to SBM. This difficulty stems from the highly correlated edge generation in GBM (while

edges are independent in SBM). For this reason, analyses of the sphere-comparison algorithm and spectral methods for clustering on GBM cannot be derived as straightforward adaptations. Whereas, even for simple algorithms, a property that can be immediately seen for SBM, will still require a proof for GBM.

6.4 The Motif-Counting Algorithm

Suppose, we are given a graph $G = (V, E)$ that comprises of two disjoint clusters, $C_1, C_2 \subseteq V$ and is generated according to $GBM(V, t, r_s, r_d)$. Our clustering algorithm is based on counting motifs, where a motif is simply defined as a configuration of triplets in the graph. Let us explain this principle by one particular motif, a triangle. For any two vertices u and v in V , where (u, v) is an edge, we count the total number of common neighbors of u and v . We show that, whenever $r_s \geq 4r_d$, this count is different when u and v belong to the same cluster, compared to when they belong to different clusters. We assume G is connected, because otherwise it is impossible to recover the clusters with certainty. For every pair of vertices in the graph that share an edge, we decide whether they are in the same cluster or not by this count of triangles. In reality, we do not have to check every such pair, instead we can stop when we form a spanning tree. At this point, we can transitively deduce the partition of nodes into clusters.

The main new idea of this algorithm is to use this triangle-count (or motif-count in general), since they carry significantly more information regarding the connectivity of the graph than an edge count. However, we can go to statistics of higher order (such as the two-hop common neighbors) at the expense of increased complexity. Surprisingly, the simple greedy algorithm that rely on triplets can separate clusters when r_d and r_s are $\Omega(\frac{\log n}{n})$, which is also a minimal requirement for connectivity of random geometric graphs [162]. Therefore this algorithm is optimal up to a constant factor. It is interesting to note that this motif-counting algorithm is not optimal

for SBM (as we observe), in particular, it will not detect the clusters in the sparse threshold region of $\frac{\log n}{n}$, however, it does so for GBM.

The pseudocode of the algorithm is described in Algorithm 23. The algorithm looks at individual pairs of vertices to decide whether they belong to the same cluster or not. We go over pair of vertices and label them same/different, till we have enough labels to partition the graphs into clusters.

At any stage, the algorithm picks up an unassigned node v and queries it with another node $u : (u, v) \in E$ that has already been assigned to one of the clusters. Note that it is always possible to find such a vertex v because otherwise the graph would not be connected. To decide whether these two points u and v belong to the same cluster, the algorithm calls a subroutine named **process**. The **process** function counts the number of common neighbors of u and v to make a decision. The node v is assigned to its respective cluster depending upon the output of **process** subroutine. This procedure is continued till all nodes in V are assigned to one of the clusters.

Algorithm 23 Cluster recovery in GBM

Require: GBM $G = (V, E)$, r_s, r_d

Ensure: $V = C_1 \sqcup C_2$

```

1: Choose any  $u \in V$ 
2:  $C_1 \leftarrow \{u\}$ ,  $C_2 \leftarrow \emptyset$ 
3: while  $V \neq C_1 \sqcup C_2$  do
4:   Choose  $(u, v) \in E \mid u \in C_1 \sqcup C_2, v \in V \setminus (C_1 \sqcup C_2)$ 
5:   if process( $u, v, r_s, r_d$ ) then
6:     if  $u \in C_1$  then
7:        $C_1 \leftarrow C_1 \cup \{v\}$ 
8:     else
9:        $C_2 \leftarrow C_2 \cup \{v\}$ 
10:    end if
11:  else
12:    if  $u \in C_2$  then
13:       $C_2 \leftarrow C_2 \cup \{v\}$ 
14:    else
15:       $C_1 \leftarrow C_1 \cup \{v\}$ 
16:    end if
17:  end if
18: end while

```

Algorithm 24 process

Require: u, v, r_s, r_d **Ensure:** true/false

- 1: $\text{count} \leftarrow |\{z : (z, u) \in E, (z, v) \in E\}|$
 - 2: **if** $|\frac{\text{count}}{n} - E_S(r_d, r_s)| < |\frac{\text{count}}{n} - E_D(r_d, r_s)|$ **then return** true
 - 3: **end if** **return** false
-

The **process** function counts the number of common neighbors of two nodes and then compares the difference of the count with two functions of r_d and r_s , called E_D and E_S .

We have compiled the distribution of the number of common neighbors along with other motifs (other patterns of triplets, given $(u, v) \in E$) in Table 6.2. We provide the values of E_D and E_S in Theorem 6 for the regime of $r_s > 4r_d$. In this table we have assumed that there are only two clusters of equal size. The functions change when the cluster sizes are different. Our analysis described in later sections can be used to calculate new function values. In the table, $u \sim v$ means u and v are in the same cluster.

Similarly, the **process** function can be run on other set of motifs by fixing two nodes. On considering a larger set of motifs, the **process** function can take a majority vote over the decisions received from different motifs. Note that our algorithm counts motifs only for edges, and does not count motifs for more than $n - 1$ edges, as there are only n vertices to be assigned to clusters.

Remark 1. *If we are given k clusters ($k > 2$), our analysis can be extended to calculate new values of E_S and E_D . If there exists a palpable gap between the two values, we can extend Algorithm 23 to identify the true assignment of each node.*

6.5 Analysis of the Algorithm

The critical observation that we have to make to analyze the motif-counting algorithm is the fact that given a GBM graph $G(V, E)$ with two clusters $V = C_1 \sqcup C_2$,

Table 6.2: Distribution of motif count for an edge (u, v) conditioned on the distance between them $d_L(X_u, X_v) = x$, when there are two equal sized clusters. Here $\text{Bin}(n, p)$ denotes a binomial random variable with mean np .

| Motif: $(u, v) \in E$ $d_L(X_u, X_v) = x$ | Distribution of count $(r_s > 2r_d)$ | | Distribution of count $(r_s \leq 2r_d)$ | |
|--|--|---|---|---|
| | $u \sim v, x \leq r_s$ | $u \not\sim v, x \leq r_d$ | $u \sim v, x \leq r_s$ | $u \not\sim v, x \leq r_d$ |
| Motif 1: $z \mid (z, u) \in E, (z, v) \in E$ | $\text{Bin}(\frac{n}{2} - 2, 2r_s - x) + \mathbb{1}\{x \leq 2r_d\} \text{Bin}(\frac{n}{2}, 2r_d - x)$ | $\text{Bin}(n - 2, 2r_d)$ | $\text{Bin}(\frac{n}{2} - 2, 2r_s - x) + \text{Bin}(\frac{n}{2}, 2r_d - x)$ | $\text{Bin}(n - 2, \min(r_s + r_d - x, 2r_d))$ |
| Motif 2: $z \mid (z, u) \in E, (z, v) \notin E$ | $\text{Bin}(\frac{n}{2} - 2, x) + \text{Bin}(\frac{n}{2}, \min(x, 2r_d))$ | $\text{Bin}(\frac{n}{2} - 1, 2(r_s - r_d))$ | $\text{Bin}(n - 2, x)$ | $\text{Bin}(\frac{n}{2} - 1, r_s - r_d + x + \max(r_s - x - r_d, 0)) + \text{Bin}(\frac{n}{2} - 1, \max(x + r_d - r_s, 0))$ |
| Motif 3: $z \mid (z, u) \notin E, (z, v) \in E$ | $\text{Bin}(\frac{n}{2} - 2, x) + \text{Bin}(\frac{n}{2}, \min(x, 2r_d))$ | $\text{Bin}(\frac{n}{2} - 1, 2(r_s - r_d))$ | $\text{Bin}(n - 2, x)$ | $\text{Bin}(\frac{n}{2} - 1, r_s + r_d - x + \max(r_s - x - r_d, 0)) + \text{Bin}(\frac{n}{2} - 1, \max(x + r_d - r_s, 0))$ |
| Motif 4: $z \mid (z, u) \notin E, (z, v) \notin E$ | $\text{Bin}(\frac{n}{2} - 2, 1 - (x + 2r_s)) + \mathbb{1}\{x \leq 2r_d\} \text{Bin}(\frac{n}{2}, 1 - (x + 2r_d)) + \mathbb{1}\{x > 2r_d\} \text{Bin}(\frac{n}{2}, 1 - 4r_d)$ | $\text{Bin}(n - 2, 1 - 2r_s)$ | $\text{Bin}(\frac{n}{2} - 2, 1 - (x + 2r_s)) + \text{Bin}(\frac{n}{2}, 1 - (x + 2r_d))$ | $\mathbb{1}\{x \leq r_s - r_d\} \text{Bin}(n - 2, 1 - 2r_s) + \mathbb{1}\{x > r_s - r_d\} \text{Bin}(n - 2, 1 - (x + r_s + r_d))$ |

and a pair of vertices $u, v \in V$, the events $\mathcal{E}_z^{u,v}, z \in V$ of any other vertex z being a common neighbor of both u and v given $(u, v) \in E$ are dependent (this is not true in SBM); however given the distance between the corresponding random variables $d_L(X_u, X_v) = x$, the events are independent. Moreover, the probabilities of $\mathcal{E}_z^{u,v} \mid (u, v) \in E$ are different when u and v are in the same cluster and when they are in different clusters. Therefore the count of the common neighbors are going to be different, and substantially separated with high probability for two vertices in cases when they are from the same cluster or from different clusters. This will lead the function `process` to correctly characterize two vertices as being from same or different clusters with high probability.

Let us now show this more formally. We have the following two lemmas for a GBM graph $G(V, E)$ with two equal-sized (unknown) clusters $V = C_1 \sqcup C_2$, and parameters r_s, r_d .

Lemma 11. *For any two vertices $u, v \in C_i : (u, v) \in E, i = 1, 2$ belonging to the same cluster, the event $\mathcal{E}_z^{u,v} \equiv \{(u, z), (v, z) \in E\}$ is independent with $\mathcal{E}_w^{u,v} \equiv \{(u, w), (v, w) \in E\}$ conditional on the distance between X_u and X_v , $d_L(X_u, X_v) = x$.*

Proof. Let us assume that z, w belong to the same cluster as that of u, v (the proof is similar for other cases too, and we omit those cases here). The event $\mathcal{E}_z^{u,v} \cap \mathcal{E}_w^{u,v}$ given $d_L(X_u, X_v) = x$ is equivalent to having both X_z and X_w (the random variable corresponding to vertices z and w respectively) within a range of $2r_s - x$ if $x \leq 2r_s$ and can never happen if $x > 2r_s$. Hence $\Pr(\mathcal{E}_z^{u,v} \cap \mathcal{E}_w^{u,v} | d_L(X_u, X_v) = x) = (2r_s - x)^2$ for $x \leq 2r_s$.

On the other hand, the event $\mathcal{E}_z^{u,v}$ given $d_L(X_u, X_v) = x$ is equivalent to having X_z within a range of $2r_s - x$ if $x \leq 2r_s$ and 0 otherwise. Similarly the event $\mathcal{E}_w^{u,v}$ given $d_L(X_u, X_v) = x$ is equivalent to having X_w within a range of $2r_s - x$. Therefore $\Pr(\mathcal{E}_z^{u,v} \cap \mathcal{E}_w^{u,v} | d_L(X_u, X_v) = x) = \Pr(\mathcal{E}_z^{u,v} | d_L(X_u, X_v) = x) \Pr(\mathcal{E}_w^{u,v} | d_L(X_u, X_v) = x)$.

□

This observation leads to the derivation of distributions of counts of triangles involving $(u, v) \in E$ for the cases when u and v are in the same cluster and when they are not.

Lemma 12. *For any two vertices $u, v \in C_i : (u, v) \in E, i = 1, 2$ belonging to the same cluster and $d_L(X_u, X_v) = x$, the count of common neighbors $\mathbf{Count}_{u,v} \equiv |\{z \in V : (z, u), (z, v) \in E\}|$ is a random variable distributed according to $\text{Bin}(\frac{n}{2} - 2, 2r_s - x) + \text{Bin}(\frac{n}{2}, 2r_d - x)$ if $x \leq \min(2r_d, r_s)$ and according to $\text{Bin}(\frac{n}{2} - 2, 2r_s - x)$ if $2r_d < x \leq r_s$, where $\text{Bin}(n, p)$ is a binomial random variable with mean np .*

Proof of Lemma 12. Let $X_w \in [0, 1]$ be the uniform random variable associated with $w \in V$. Let us also denote by $d_L(X, Y) \equiv \min\{|X - Y|, 1 - |X - Y|\}$, $X, Y \in \mathbb{R}$. Without loss of generality, assume $u, v \in C_1$. For any vertex $z \in V$, let $\mathcal{E}_z^{u,v}(x) \equiv \{(u, z), (v, z) \in E | (u, v) \in E, d_L(u, v) = x\}$ be the event that z is a common neighbor

given that the vertices u and v have an edge and the distance between those vertices is x . For $z \in C_1$,

$$\Pr(\mathcal{E}_z^{u,v}(x)) = 2r_s - x, 0 \leq x \leq r_s.$$

For $z \in C_2$,

$$\Pr(\mathcal{E}_z^{u,v}(x)) = \begin{cases} 2r_d - x, & \text{if } x \leq 2r_d \\ 0, & \text{if } 2r_d < x \leq r_s. \end{cases}$$

Since we are conditioning on the fact that the vertices u and v have an edge, x can take a maximum value of r_s . Now since there are $\frac{n}{2} - 2$ points in $C_1 \setminus \{u, v\}$ and $\frac{n}{2}$ points in C_2 , we have the statement of the lemma. \square

Lemma 13. *For any two vertices $u \in C_1, v \in C_2 : (u, v) \in E$ belonging to different clusters and $d_L(X_u, X_v) = x$, the count of common neighbors $\mathbf{Count}_{u,v} \equiv |\{z \in V : (z, u), (z, v) \in E\}|$ is a random variable distributed according to $\text{Bin}(n - 2, 2r_d)$ when $r_s > 2r_d$ and according to $\text{Bin}(n - 2, \min(r_s + r_d - x, 2r_d))$ when $r_s \leq 2r_d$.*

Proof. Here u, v are from different clusters. For any vertex $z \in V$, let $\mathcal{E}_z^{u,v}(x) \equiv \{(u, z), (v, z) \in E \mid (u, v) \in E, d_L(X_u, X_v) = x\}$ be the event that z is a common neighbor. For $z \in V \setminus \{u, v\}$,

$$\begin{aligned} \Pr(\mathcal{E}_z^{u,v}(x)) &= \Pr((z, u) \in E, (z, v) \in E \mid (u, v) \in E, d_L(X_u, X_v) = x) \\ &= \min\{2r_d, r_d + r_s - x\} \\ &= \begin{cases} 2r_d & \text{if } 2r_d < r_s \\ r_s + r_d - x & \text{otherwise} \end{cases}. \end{aligned}$$

Now since there are $n - 2$ points in $V \setminus \{u, v\}$, we have the statement of the lemma. \square

These expressions can also be generalized when the clusters are of unequal sizes, but we omit those for clarity of exposition.

Consider the case when $r_s \geq 4r_d$. The above lemmas show that for all values of $d_L(X_u, X_v)$, the expected count of the number of triangles involving $(u, v) \in E$ is higher when u and v belong to the same cluster as opposed to different clusters. By leveraging the concentration of binomial random variables, we bound the count of the number of triangles in these two cases. We use Lemma 12 to first estimate the minimum value of triangle count when u and v belong to the same cluster and Lemma 13 to estimate the maximum value of triangle count when u and v belong to different clusters. Our algorithm will correctly resolve whether two points are in the same cluster or not if the minimum value in the former case is higher than the maximum value in the latter. While more general statements are possible, we give a theorem concentrating on the special case when $r_s, r_d \sim \frac{\log n}{n}$, which is at the order of the connectivity threshold of geometric random graphs [162].

Theorem 6. *Let $r_s = \frac{a \log n}{n}$ and $r_d = \frac{b \log n}{n}$, $a > 4b$, and $g(y) \equiv y + \sqrt{2a - y} + \sqrt{2b - y}$. Algorithm 23 with $E_D = (2b + \sqrt{6b}) \frac{\log n}{n}$ and*

$$E_S = \min \left(\frac{a}{2} - \sqrt{a}, a + b - \max_{0 \leq \nu \leq 2b} g(\nu) \right) \frac{\log n}{n},$$

can recover the clusters C_1, C_2 accurately with a probability of $1 - o(1)$ if

$$\min \left(\frac{a}{2} - \sqrt{a}, a + b - \max_{0 \leq \nu \leq 2b} g(\nu) \right) \geq 2b + \sqrt{6b}.$$

Proof. We need to consider the case of $r_s > 2r_d$ from Lemma 12 and Lemma 13. Let Z denote the random variable that equals the number of common neighbors of two nodes $u, v \in V : (u, v) \in E$. Let us also denote $\mu_s = \mathbb{E}(Z | u \sim v, d_L(X_u, X_v) = x)$ and $\mu_d = \mathbb{E}(Z | u \not\sim v, d_L(X_u, X_v) = x)$, where $u \sim v$ means u and v are in the

same cluster. We can easily find μ_s and μ_d from Lemmas 12, 13. We see that,

$$\mu_s = \begin{cases} n(r_s + r_d - x) - 4r_s + 2x, & \text{if } x \leq 2r_d \\ (\frac{n}{2} - 2)(2r_s - x), & \text{if } 2r_d < x \leq r_s \end{cases} \quad \text{and} \quad \mu_d = (n - 2)2r_d.$$

The value of μ_s is greater than that of μ_d for all values of x when $r_s \geq 4r_d$. We try to bound the values of Z in these two cases and then achieve the condition of correct resolution. Given a fixed $d_L(X_u, X_v)$, since Z is a sum of independent binary random variables, using the Chernoff bound, $\Pr(Z < (1 - \delta)\mathbb{E}(Z)) \leq e^{-\delta^2\mathbb{E}(Z)/2} = \frac{1}{n \log n}$, when $\delta = \sqrt{\frac{2(\log n + \log \log n)}{\mathbb{E}(Z)}}$. Now when u, v belong to the same cluster and $d_L(X_u, X_v) = x$, with probability at least $1 - \frac{2}{n \log n}$,

$$Z \geq F_{\sim}(x) \equiv \begin{cases} n(r_s + r_d - x) - 4r_s + 2x - \sqrt{(\log n + \log \log n)(n - 4)(2r_s - x)} \\ \quad - \sqrt{(\log n + \log \log n)n(2r_d - x)}, & \text{if } x \leq 2r_d \\ (\frac{n}{2} - 2)(2r_s - x) - \sqrt{(\log n + \log \log n)(n - 4)(2r_s - x)}, \\ \quad \text{if } 2r_d < x \leq r_s. \end{cases}$$

Using Chernoff bound, we also know that $\Pr(Z > (1 + \delta)\mathbb{E}(Z)) \leq e^{-\delta^2\mathbb{E}(Z)/3} = \frac{1}{n \log n}$, when $\delta = \sqrt{\frac{3(\log n + \log \log n)}{\mathbb{E}(Z)}}$. Hence, with probability at least $1 - \frac{1}{n \log n}$, Z is at most $F_{\infty} \equiv \mu_d + \sqrt{3(\log n + \log \log n)\mu_d}$ when u, v belong to different clusters.

We calculate the minimum value of $F_{\sim}(x)$ over all values of x to find the value closest to F_{∞} . When $2r_d < x < r_s$, $F_{\sim}(x)$ is a decreasing function with the minimum value of $(\frac{n}{2} - 2)r_s - \sqrt{(\log n + \log \log n)(n - 4)r_s}$ at $x = r_s$. Plugging in $r_s = \frac{a \log n}{n}$, $r_d = \frac{b \log n}{n}$ and $x = \frac{\nu \log n}{n}$ we get that the algorithm will be successful to label correctly with probability $1 - \frac{3}{n \log n}$ as long as,

$$\min\left(\frac{a}{2} - \sqrt{a}, \min_{0 \leq \nu \leq 2b} (a + b - \nu - \sqrt{2a - \nu} - \sqrt{2b - \nu})\right) \log n \geq (2b + \sqrt{6b}) \log n.$$

Now we need the correct assignment of vertices for $n - 1$ pairs of vertices (according to Algorithm 23). Applying union bound over $n - 1$ distinct pairs guarantees the probability of recovery as $1 - 3/\log n$. \square

Instead of relying only on the triangle (or common-neighbor) motif, we can consider other different motifs (as listed in Table 6.2) and use them to make similar analysis. Aggregating the different motifs by taking a majority vote decision may improve the results experimentally but it is difficult to say anything theoretically since the decisions of the different motifs are not independent. We refer the reader to Section 6.5.1 for the detailed analysis of incorporating other motifs to obtain analogous theorems.

Remark 2. *Instead of using Chernoff bound we could have used better concentration inequality (such as Poisson approximation) in the above analysis, to get tighter conditions on the constants. We again preferred to keep things simple.*

Remark 3 (GBM for $t = 3$ and above). *For GBM with $t = 3$, to find the number of common neighbors of two vertices, we need to find out the area of intersection of two spherical caps on the sphere. It is possible to do that. It can be seen that our algorithm will successfully identify the clusters as long as $r_s, r_d \sim \sqrt{\frac{\log n}{n}}$ again when the constant terms satisfy some conditions. However tight characterization becomes increasingly difficult. For general t , our algorithm should be successful when $r_s, r_d \sim \left(\frac{\log n}{n}\right)^{\frac{1}{t-1}}$, which is also the regime of connectivity threshold.*

Remark 4 (More than two clusters). *When there are more than two clusters, the same analysis technique is applicable and we can estimate the expected number of common neighbors. This generalization can be straightforward but tedious.*

Motif counting algorithm for SBM. While our algorithm is near optimal for GBM in the regime of $r_s, r_d \sim \frac{\log n}{n}$, it is far from optimal for the SBM in the same regime of average degree. Indeed, by using simple Chernoff bounds again, we see that

the motif counting algorithm is successful for SBM with inter-cluster edge probability q and intra-cluster probability p , when $p, q \sim \sqrt{\frac{\log n}{n}}$. The experimental success of our algorithm in real sparse networks therefore somewhat enforce the fact that GBM is a better model for those network structures than SBM.

6.5.1 Results for Other Motifs

Next, we describe two lemmas for a GBM graph $G(V, E)$ with two unknown clusters $V = C_1 \sqcup C_2$, and parameters r_s, r_d on considering other motifs than triangles (Motif 1). These results are used to populate Table 6.2. When we run Algorithm 23 with other motifs, the subroutine **process** uses the corresponding motifs to compute the variable ‘count’. Other than this the algorithm remains same.

Motif 2 and Motif 3.

Lemma 14. *For any two vertices $u, v \in V : (u, v) \in E, i = 1, 2$ belonging to the same cluster and $d_L(X_u, X_v) = x$, the count of number of nodes forming Motif 2 (see Table 6.2) with u and v (i.e., neighbors of u and non neighbors of v), $|\{z \in V : (z, u) \in E, (z, v) \notin E\}|$ is a random variable distributed according to $\text{Bin}(\frac{n}{2} - 2, x) + \text{Bin}(\frac{n}{2}, \min(2r_d, x))$, where $\text{Bin}(n, p)$ is a binomial random variable with mean np .*

Proof. Without loss of generality, assume $u, v \in C_1$. For any vertex $z \in V$, let $\mathcal{E}_z^{u,v}(x) = \{(u, z) \in E, (v, z) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x\}$ be the event that z is a neighbor of u and non neighbor of v . For $z \in C_1$,

$$\Pr(\mathcal{E}_z^{u,v}(x)) = \Pr((z, u) \in E, (z, v) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x) = r_s - (r_s - x) = x.$$

For $z \in C_2$, we have,

$$\begin{aligned}\Pr(\mathcal{E}_z^{u,v}(x)) &= \Pr((z, u) \in E, (z, v) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x) \\ &= \begin{cases} 2r_d & \text{if } x > 2r_d \\ x & \text{otherwise} \end{cases}.\end{aligned}$$

Now since there are $\frac{n}{2} - 2$ points in $C_1 \setminus \{u, v\}$ and $\frac{n}{2}$ points in C_2 , we have the statement of the lemma. \square

Lemma 15. *For any two vertices $u \in C_1, v \in C_2 : (u, v) \in E$ belonging to different clusters and $d_L(X_u, X_v) = x$, the count of number of nodes forming Motif 2 (see Table 6.2) with u and v (i.e. neighbor of u and non neighbor of v), $|\{z \in V : (z, u) \in E, (z, v) \notin E\}|$ is a random variable distributed according to $\text{Bin}(\frac{n}{2} - 1, 2(r_s - r_d))$, assuming $r_s > 2r_d$.*

Proof. For any vertex $z \in C_1$, let $\mathcal{E}_z^{u,v}(x) = \{(u, z) \in E, (v, z) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x\}$ be the event that z is a neighbor of u and a non neighbor of v . For $z \in C_1 \setminus \{u\}$

$$\begin{aligned}\Pr(\mathcal{E}_z^{u,v}(x)) &= \Pr((z, u) \in E, (z, v) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x) \\ &= 2(r_s - r_d).\end{aligned}$$

Now for $z \in C_2 \setminus \{v\}$, there cannot be an edge with u and no edge with v because $r_s > 2r_d$. Since there are $\frac{n}{2} - 1$ points in $C_1 \setminus \{u\}$, we have the statement of the lemma. \square

Theorem 7 (Motif 2 or 3). *If $r_s = \frac{a \log n}{n}$ and $r_d = \frac{b \log n}{n}$, $a > 4b$, Algorithm 23 with $E_S = \left(b + \frac{a}{2} + \sqrt{3b} + \sqrt{\frac{3a}{2}}\right) \frac{\log n}{n}$ and $E_D = \left((a - b) - \sqrt{2(a - b)}\right) \frac{\log n}{n}$, where the **process** subroutine counts Motif 2, can recover the clusters C_1, C_2 accurately with a probability of at least $1 - o(1)$ if*

$$(a - b) - \sqrt{2(a - b)} > b + \frac{a}{2} + \sqrt{3b} + \sqrt{\frac{3a}{2}}.$$

Proof. We need to consider the case of $r_s \geq 2r_d$ from Lemma 14 and Lemma 15. For $u, v \in V : (u, v) \in E$, let Z denote the random variable that equals the number of nodes that are neighbors of u and not-a-neighbor of v . Let us also denote $\mu_s = \mathbb{E}(Z|u \sim v, d_L(X_u, X_v) = x)$ and $\mu_d = \mathbb{E}(Z|u \not\sim v, d_L(X_u, X_v) = x)$, where $u \sim v$ means u and v are in the same cluster. We can easily find μ_s and μ_d from Lemmas 14 and 15. We see that,

$$\mu_s = \begin{cases} (n-2)x, & \text{if } x \leq 2r_d \\ (\frac{n}{2}-2)x + nr_d, & \text{if } 2r_d < x \leq r_s. \end{cases} \quad \text{and} \quad \mu_d = (n-2)(r_s - r_d).$$

The value of μ_s is less than that of μ_d for all values of x when $r_s \geq 4r_d$. We try to bound the values of Z in the two cases possible and then achieve the condition of correct resolution. Given a fixed $d_L(X_u, X_v)$, since Z is a sum of independent binary random variables, using the Chernoff bound, $\Pr(Z < (1-\delta)\mathbb{E}(Z)) \leq e^{-\delta^2\mathbb{E}(Z)/2} = \frac{1}{n \log n}$, when $\delta = \sqrt{\frac{2(\log n + \log \log n)}{\mathbb{E}(Z)}}$. Now with probability at least $1 - \frac{1}{n \log n}$, Z is atleast $F_{\sim}^1(x) \equiv \mu_d - \sqrt{2(\log n + \log \log n)\mu_d}$ when u and v belong to different clusters.

Using Chernoff bound, we also know that $\Pr(Z > (1+\delta)\mathbb{E}(Z)) \leq e^{-\delta^2\mathbb{E}(Z)/3} = \frac{1}{n \log n}$, when $\delta = \sqrt{\frac{3(\log n + \log \log n)}{\mathbb{E}(Z)}}$. Hence, with probability at least $1 - \frac{2}{n \log n}$,

$$Z \leq F_{\sim}^1(x) \equiv \begin{cases} (n-2)x + \sqrt{3(\log n + \log \log n)\frac{n}{2}x} + \sqrt{3(\log n + \log \log n)(\frac{n}{2}-2)x}, & \text{if } x \leq 2r_d \\ nr_d + (\frac{n}{2}-2)x + \sqrt{3(\log n + \log \log n)nr_d} \\ \quad + \sqrt{3(\log n + \log \log n)(\frac{n}{2}-2)x}, & \text{if } 2r_d < x \leq r_s. \end{cases}$$

when u, v belong to the same cluster and $d_L(X_u, X_v) = x$.

We calculate the maximum value of $F_{\sim}^1(x)$ over all values of x to find the value closest to $F_{\sim}^1(x)$. $F_{\sim}^1(x)$ is an increasing function $\forall x \leq r_s$ with maximum value of

$nr_d + (\frac{n}{2} - 2)r_s + \sqrt{3(\log n + \log \log n)nr_d} + \sqrt{3(\log n + \log \log n)(\frac{n}{2} - 2)r_s}$ at $x = r_s$.

Therefore the algorithm will be successful to label correctly with probability $1 - \frac{3}{n \log n}$ as long as,

$$\left((a - b) - \sqrt{2(a - b)} \right) \log n \geq \left(b + \frac{a}{2} + \sqrt{3b} + \sqrt{\frac{3a}{2}} \right) \log n.$$

The rest of the argument follows similar to Theorem 6. \square

Motif 4. In this part we are concerned with the motif where for $(u, v) \in E$, we seek nodes that are neighbors of neither u nor v .

Lemma 16. *For any two vertices $u, v \in V : (u, v) \in E, i = 1, 2$ belonging to the same cluster and $d_L(X_u, X_v) = x$, the count of nodes that form Motif 4 with u, v (i.e., non-neighbors of both u and v), $|\{z \in V : (z, u) \notin E, (z, v) \notin E\}|$ is a random variable distributed according to $\text{Bin}(\frac{n}{2} - 2, 1 - (x + 2r_s)) + \mathbb{1}\{x \leq 2r_d\} \text{Bin}(\frac{n}{2}, 1 - (x + 2r_d)) + \mathbb{1}\{x > 2r_d\} \text{Bin}(\frac{n}{2}, 1 - 4r_d)$, when $r_s > 2r_d$.*

Proof. Without loss of generality, assume $u, v \in C_1$. For any vertex $z \in V$, let $\mathcal{E}_z^{u,v}(x) = \{(u, z) \notin E, (v, z) \notin E\}$ be the event that z is neither a neighbor of u nor a neighbor of v . For $z \in C_1$,

$$\begin{aligned} \Pr(\mathcal{E}_z^{u,v}(x)) &= \Pr((z, u) \notin E, (z, v) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x) \\ &= 1 - \Pr((z, u) \in E \text{ or } (z, v) \in E \mid (u, v) \in E, d_L(X_u, X_v) = x) \\ &= 1 - (x + 2r_s). \end{aligned}$$

For $z \in C_2$, we have,

$$\begin{aligned}
\Pr(\mathcal{E}_z^{u,v}(x)) &= \Pr((z, u) \notin E, (z, v) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x) \\
&= 1 - \Pr((z, u) \in E \text{ or } (z, v) \in E \mid (u, v) \in E, d_L(X_u, X_v) = x) \\
&= \begin{cases} 1 - (2r_d + x) & \text{if } x \leq 2r_d \\ 1 - 4r_d & \text{otherwise} \end{cases}.
\end{aligned}$$

Now since there are $\frac{n}{2} - 2$ points in $C_1 \setminus \{u, v\}$ and $\frac{n}{2}$ points in C_2 , we have the statement of the lemma. \square

Lemma 17. *For any two vertices $u \in C_1, v \in C_2 : (u, v) \in E$ belonging to different clusters and $d_L(X_u, X_v) = x$, the count of number of nodes forming Motif 4 with u and v (i.e. non-neighbor of u and non-neighbor of v), $|\{z \in V : (z, u) \notin E, (z, v) \notin E\}|$ is a random variable distributed according to $\text{Bin}(n - 2, 1 - 2r_s)$, when $r_s > 2r_d$.*

Proof. For any vertex $z \in C_1$, let $\mathcal{E}_z^{u,v}(x) = \{(u, z) \notin E, (v, z) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x\}$ be the event that z is neither a neighbor of u and nor a neighbor of v . For $z \in C_1 \setminus \{u\}$

$$\begin{aligned}
\Pr(\mathcal{E}_z^{u,v}(x)) &= \Pr((z, u) \notin E, (z, v) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x) \\
&= 1 - 2r_s.
\end{aligned}$$

Similarly, for $z \in C_2 \setminus \{v\}$

$$\begin{aligned}
\Pr(\mathcal{E}_z^{u,v}(x)) &= \Pr((z, u) \notin E, (z, v) \notin E \mid (u, v) \in E, d_L(X_u, X_v) = x) \\
&= 1 - 2r_s.
\end{aligned}$$

Now since there are $\frac{n}{2} - 1$ points in $C_1 \setminus \{u\}$ and $\frac{n}{2} - 1$ points in $C_2 \setminus \{v\}$, we have the statement of the lemma. \square

It turns out that the simple Chernoff bound is not sufficient to prove any meaningful result for this motif. We recall Bernstein's inequality in Lemma 18 in order to prove Theorem 8 for the 4th motif.

Lemma 18 (Bernstein's Inequality [41]). *Let X_1, \dots, X_n be iid real-valued random variables with mean zero, such that $|X_i| \leq M \forall i$. Then with probability at least $1 - \delta$, we have*

$$\left| \sum_{i=1}^n X_i \right| \leq \sqrt{2n\mathbb{E}X_1^2 \log \frac{2}{\delta}} + \frac{2M \log \frac{2}{\delta}}{3}.$$

Theorem 8 (Motif 4). *If $r_s = \frac{a \log n}{n}$ and $r_d = \frac{b \log n}{n}$, $a > 4b$, Algorithm 23 with*

$$E_S = 1 - \frac{3}{2}r_s - 2r_d - \sqrt{3r_s \frac{\log n}{n}} - \sqrt{4r_d \frac{\log n}{n}} - \frac{4 \log n}{3n}$$

and

$$E_D = 1 - 2r_s + \sqrt{4r_s \frac{\log n}{n}} + \frac{2 \log n}{3n},$$

*where the **process** subroutine counts Motif 4, can recover the clusters C_1, C_2 accurately with a probability of $1 - o(1)$ if $|a - 4b| \geq 2(\sqrt{3a} + \sqrt{4b} + \sqrt{4a} + 2)$.*

Proof. We need to consider the case of $r_s \geq 2r_d$. Let Z denote the random variable that equals the number of common non-neighbors of two nodes $u, v \in V : (u, v) \in E$. Let us also denote $\mu_s = \mathbb{E}(Z|u \sim v, d_L(X_u, X_v) = x)$ and $\mu_d = \mathbb{E}(Z|u \not\sim v, d_L(X_u, X_v) = x)$, where $u \sim v$ means u and v are in the same cluster. We can easily find μ_s and μ_d from Lemmas 16, 17. We see that, $\mu_s = \begin{cases} (\frac{n}{2} - 2)(1 - x - 2r_s) + \frac{n}{2}(1 - x - 2r_d), & \text{if } x \leq 2r_d \\ (\frac{n}{2} - 2)(1 - x - 2r_s) + \frac{n}{2}(1 - 4r_d), & \text{if } 2r_d < x \leq r_s. \end{cases}$ and $\mu_d = (n - 2)(1 - 2r_s)$.

The value of μ_s is more than that of μ_d for all values of x when $r_s \geq 4r_d$. We try to bound the values of Z in the two cases possible and then achieve the condition of correct resolution. Now we will use Bernstein's inequality as defined in Lemma 18.

For a Bernoulli(p) random variable X we can define a corresponding zero mean random variable $\hat{X} \equiv X - \mathbb{E}[X]$. It is easy to observe that $\mathbb{E}[\hat{X}^2] = p(1-p) \leq 1-p$ and $|\hat{X}| \leq 1$. We use this simple translation for every random variable corresponding to each node forming such a motif with u and v and hence with a probability of at least $1 - \frac{2}{n \log n}$, we must have

$$Z \geq F_{\sim}^2(x) \equiv \begin{cases} \left(\frac{n}{2} - 2 \right) (1 - x - 2r_s) + \frac{n}{2} (1 - x - 2r_d) - \sqrt{n(x + 2r_s)(\log 2n + \log \log n)} \\ \quad - \sqrt{n(x + 2r_d)(\log 2n + \log \log n)} - \frac{4(\log 2n + \log \log n)}{3}, \\ \quad \text{if } x \leq 2r_d \\ \left(\frac{n}{2} - 2 \right) (1 - x - 2r_s) + \frac{n}{2} (1 - 4r_d) - \sqrt{n(x + 2r_s)(\log 2n + \log \log n)} \\ \quad - \sqrt{4nr_d(\log 2n + \log \log n)} - \frac{4(\log 2n + \log \log n)}{3}, \\ \quad \text{if } 2r_d < x \leq r_s, \end{cases}$$

when u and v are in the same cluster. Similarly, with probability at least $1 - \frac{1}{n \log n}$, Z is at most $F_{\sim}^2 \equiv \mu_d + \sqrt{4(n-2)r_s(\log n + \log \log n)} + \frac{2(\log n + \log \log n)}{3}$ when u, v belong to different clusters.

We calculate the minimum value of $F_{\sim}^2(x)$ over all values of x to find the value closest to F_{\sim}^2 . It can be easily observed that $F_{\sim}^2(x)$ is a decreasing function with the minimum value of $\left(\frac{n}{2} - 2 \right) (1 - 3r_s) + \frac{n}{2} (1 - 4r_d) - \sqrt{3nr_s(\log 2n + \log \log n)} - \sqrt{4nr_d(\log 2n + \log \log n)} - \frac{4(\log 2n + \log \log n)}{3}$ at $x = r_s$. Plugging in $r_s = \frac{a \log n}{n}$, $r_d = \frac{b \log n}{n}$ we get that the algorithm will be successful to resolve correctly with probability $1 - \frac{3}{n \log n}$ as long as,

$$\begin{aligned} & \left(\frac{n}{2} - 2 \right) (1 - 3r_s) + \frac{n}{2} (1 - 4r_d) - \sqrt{3nr_s(\log 2n + \log \log n)} \\ & \quad - \sqrt{4nr_d(\log 2n + \log \log n)} - \frac{4(\log 2n + \log \log n)}{3} \\ & \geq (n-2)(1-2r_s) + \sqrt{4(n-2)r_s(\log n + \log \log n)} + \frac{2(\log n + \log \log n)}{3}. \end{aligned}$$

Plugging in $r_s = \frac{a \log n}{n}$, $r_d = \frac{b \log n}{n}$ and ignoring $o(\log n)$ factors, we get that

$$a - 4b \geq 2(\sqrt{3a} + \sqrt{4b} + \sqrt{4a} + 2).$$

□

6.6 Experimental Results

In addition to validation experiments in Section 6.3.1 and 6.3.2, we also conducted an in-depth experimentation of our proposed model and techniques over a set of synthetic and real world networks. Additionally, we compared the efficacy and efficiency of our motif-counting algorithm with the popular spectral clustering algorithm using normalized cuts¹ and the correlation clustering algorithm [32].

Real Datasets. We use three real datasets described below.

- **Political Blogs.** [15] It contains a list of political blogs from the 2004 US Election classified as liberal or conservative, and links between the blogs. The clusters are of roughly the same size with a total of 1200 nodes and 20K edges.
- **DBLP.** [206] The DBLP dataset is a collaboration network where the ground truth communities are defined by the research community. The original graph consists of roughly 0.3 million nodes. We process it to extract the top two communities of size ~ 4500 and 7500 respectively. This is given as input to our algorithm.
- **LiveJournal.** [137] The LiveJournal dataset is a free online blogging social network of around 4 million users. Similar to DBLP, we extract the top two clusters of sizes 930 and 1400 which consist of around 11.5K edges.

We have not used the academic collaboration (Section 6.3.1) dataset here because it is quite sparse and below the connectivity threshold regime of both GBM and SBM.

¹<http://scikit-learn.org/stable/modules/clustering.html#spectral-clustering>

Synthetic Datasets. We generate synthetic datasets of different sizes according to the GBM with $t = 2, k = 2$ and for a wide spectrum of values of r_s and r_d , specifically we focus on the sparse region where $r_s = \frac{a \log n}{n}$ and $r_d = \frac{b \log n}{n}$ with variable values of a and b .

Experimental Setting. For real networks, it is difficult to calculate an exact threshold as the exact values of r_s and r_d are not known. Hence, we follow a three step approach. Using a somewhat large threshold T_1 we sample a subgraph S such that u, v will be in S if there is an edge between u and v , and they have at least T_1 common neighbors. We now attempt to recover the subclusters inside this subgraph by following our algorithm with a small threshold T_2 . Finally, for nodes that are not part of S , say $x \in V \setminus S$, we select each $u \in S$ that x has an edge with and use a threshold of T_3 to decide if u and x should be in the same cluster. The final decision is made by taking a majority vote. We can employ sophisticated methods over this algorithm to improve the results further, which is beyond the scope of this work.

We use the popular f-score metric which is the harmonic mean of precision (fraction of number of pairs correctly classified to total number of pairs classified into clusters) and recall (fraction of number of pairs correctly classified to the total number of pairs in the same cluster for ground truth), as well as the node error rate for performance evaluation. A node is said to be misclassified if it belongs to a cluster where the majority comes from a different ground truth cluster (breaking ties arbitrarily). Following this, we use the above described metrics to compare the performance of different techniques on various datasets.

Table 6.3: Performance on real world networks

| Dataset | Total no. of nodes | T_1 | T_2 | T_3 | Accuracy | | Running Time (sec) | |
|-----------------|-----------------------|-------|-------|-------|----------------|---------------------|--------------------|---------------------|
| | | | | | Motif-Counting | Spectral clustering | Motif-Counting | Spectral clustering |
| Political Blogs | 1222 | 20 | 2 | 1 | 0.788 | 0.53 | 1.62 | 0.29 |
| DBLP | 12138 | 10 | 1 | 2 | 0.675 | 0.63 | 3.93 | 18.077 |
| LiveJournal | 2366 | 20 | 1 | 1 | 0.7768 | 0.64 | 0.49 | 1.54 |

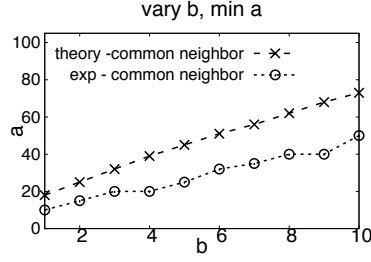
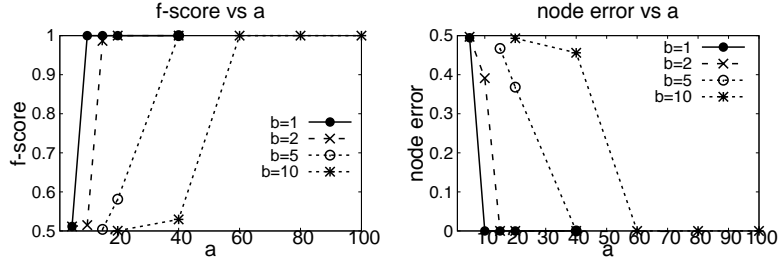


Figure 6.4: Triangle motif varying b and minimum value of a that satisfies the accuracy bound for a synthetic dataset with 5000 nodes.



(a) f-score with varying a , fixed b

(b) Fraction of nodes misclassified

Figure 6.5: Results of the motif-counting algorithm on a synthetic dataset with 5000 nodes.

Results. We compared our algorithm with the spectral clustering algorithm where we extracted two eigenvectors in order to extract two communities. Table 6.3 shows that our algorithm gives an accuracy as high as 78%. The spectral clustering performed worse compared to our algorithm for all real world datasets. It obtained the worst accuracy of 53% on political blogs dataset. The correlation clustering algorithm generates various small sized clusters leading to a very low recall, performing much worse than the motif-counting algorithm for the whole spectrum of parameter values.

We can observe in Table 6.3 that our algorithm is much faster than the spectral clustering algorithm for larger datasets (LiveJournal and DBLP). This confirms that motif-counting algorithm is more scalable than the spectral clustering algorithm. The spectral clustering algorithm also works very well on synthetically generated SBM networks even in the sparse regime [136, 169]. The superior performance of the sim-

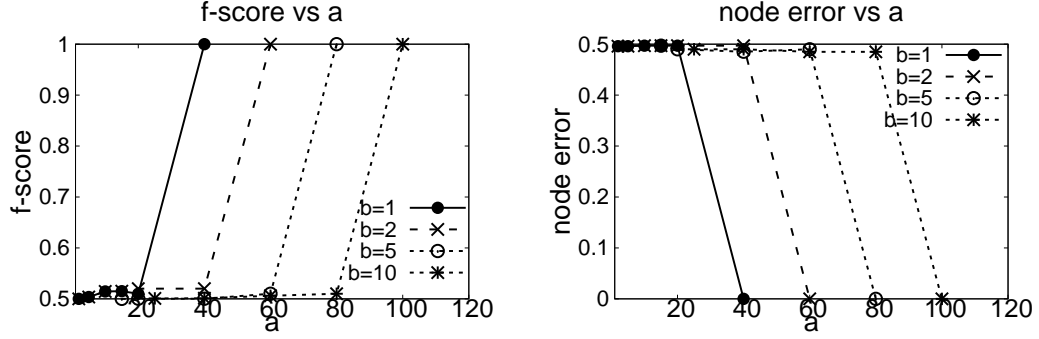


Figure 6.6: Results of the spectral clustering on a synthetic dataset with 5000 nodes.

ple motif clustering algorithm over the real networks provide a further validation of GBM over SBM. Correlation clustering takes 8-10 times longer as compared to motif-counting algorithm for the various range of its parameters. We also compared our algorithm with the Newman algorithm [101] that performs really well for the LiveJournal dataset (98% accuracy). But it is extremely slow and performs much worse on other datasets. This is because the LiveJournal dataset has two well defined subsets of vertices with very few intercluster edges. The reason for the worse performance of our algorithm is the sparseness of the graph. If we create a subgraph by removing all nodes of degrees 1 and 2, we get 100% accuracy with our algorithm. Finally, our algorithm is easily parallelizable to achieve better improvements. This clearly establishes the efficiency and effectiveness of motif-counting.

We observe similar gains on synthetic datasets. Figures 6.4, 6.5a and 6.5b report results on the synthetic datasets with 5000 nodes. Figure 6.4 plots the minimum gap between a and b that guarantees exact recovery according to Theorem 6 vs minimum value of a for varying b for which experimentally (with only triangle motif) we were able to recover the clusters exactly. Empirically, our results demonstrate much superior performance of our algorithm. The empirical results are much better than the theoretical bounds because the concentration inequalities applied in Theorem 6 assume the worst value of the distance between the pair of vertices that are under con-

sideration. We also see a clear threshold behavior on both f-score and node error rate in Figures 6.5a and 6.5b. We have also performed spectral clustering on this 5000-node synthetic dataset (Figures 6.6a and 6.6b). Compared to the plots of figures 6.5a and 6.5b, they show suboptimal performance, indicating the relative ineffectiveness of spectral clustering in GBM compared to the motif counting algorithm.

6.7 Summary and Future Work

This chapter studied the geometric block model, a generative model that is motivated by random geometric graphs. The proposed model is validated on Amazon co-purchase network and academic collaboration networks. We then explored the use of a simple triangle counting-based algorithm to recover the clusters and analyzed its effectiveness. Experiments on various real-world datasets justified the efficacy of the proposed techniques. Some of the key takeaways are:

- Geometric block model captures dependent edge formation which is motivated by random geometric graphs.
- Given a pair of nodes u and v such that $(u, v) \in E$, the chances that w forms a triangle with them is independent of x forming an edge with them. Due to this property, calculating the number of triangles formed by each edge helps to classify it as intra-cluster or inter-cluster.
- Motif-counting based algorithm is efficient and accurate over real-world networks.

As future work, we plan to explore the use of supervision to improve the recovery algorithm and extend these generative models for hierarchical clustering.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In this dissertation, we formalized the different facets of clustering and developed algorithms with specific focus towards robustness and scalability. While our techniques are general, we illustrate their applications for different aspects of data integration tasks. Specifically, we explore the applications of entity resolution, data summarization and community detection. We proved robustness guarantees of the proposed techniques under different noise models and validated their quality over various real-world datasets.

In Chapter 3, we developed an error correction toolkit to ensure high quality of clusters while optimizing for the number of queries to the oracle. Chapter 4 proposed a novel progressive blocking framework to improve scalability.

In Chapter 5, we introduced two different noise models for comparison queries: adversarial and probabilistic noise. We designed approximation algorithms to perform k-center clustering and agglomerative hierarchical clustering using a quadruplet comparison oracle.

In Chapter 6, we proposed a generative model to model the interactions between records. For this model, we designed a simple triangle counting based algorithm to recover the ground truth set of clusters.

7.1 Future Directions

This dissertation has raised several questions pertaining to the different facets of clustering. We now describe some research directions that we believe are the next steps to cater to the modern needs of data-based applications.

7.1.1 Alternate Forms of Supervision

In applications like entity resolution, the crowd-based oracle supervision provides noisy labels for pairs of records which are then corrected by our random graph-based algorithm. Most commonly, these oracle models use crowdsourcing along with active learning to train a classifier [34, 172]. This procedure is equivalent to a supervised learning task and techniques like weak supervision and transfer learning can help to effectively reduce the dependence of classifier training on crowd workers [168, 186, 90].

Knowledge graphs are known to contain well curated information about various real-world entities. These sources provide high-quality domain knowledge which can be used as supervision to circumvent the dependence on crowdworkers, thereby reducing the monetary cost. A future direction for effective design of oracle would be to exploit openly available domain knowledge along with transfer learning techniques to reduce the dependence on crowdworkers.

7.1.2 Entity Resolution over Data Markets and Federated Data Sources

Designing scalable techniques for integrating data from heterogeneous sources will be an increasingly important area for businesses and government. The availability of open data sources has increased the heterogeneity and ambiguity across sources. Data markets and federated data sources have raised the importance of privacy in data integration systems. One of the recent techniques [187] has explored privacy-aware techniques for entity resolution. Designing privacy preserving ways to leverage supervision for entity resolution is an interesting direction for further exploration.


| Record id | Title | Price | Picture | Type |
|-----------|---|-------|---|---------------|
| 1 | General Lighting, Soft White, 60W Equivalent A29 LED Light Bulb | 21.56 |  | General LED |
| 2 | Sylvania Ultra LED Night Chaser, 250W Equivalent, Replacement for Halogen Flood SpotLight | 20.64 |  | General LED |
| 3 | Kasa Smart Light Bulb, LED Smart WiFi Alexa Bulbs | 13.99 |  | Smart LED |
| 4 | Philips Hue White 4-Pack A19 LED Smart Bulb, Bluetooth Zigbee compatible | 49.99 |  | Smart LED |
| 5 | GE Lighting 38-watt Halogen Floodlight Bulb with Medium Base | 31.99 |  | Halogen light |
| 6 | Lithonia Lighting Mini Single-Head Flood Light 150-Watt Double Ended Quartz Halogen Lamp | 13.99 |  | Halogen light |
| 7 | Luxrite T8 Fluorescent Tube Light Bulb, Cool White | 29.99 |  | Tube light |
| 8 | Luxrite U Bend LED Tube Light, T8 T12, 18W (32WEquivalent), 2100 Lumens, Direct or Ballast Bypass | 24.99 |  | Tube light |
| 9 | F8T5/CW 8W T5 12" Cool White 4100k FluorescentLight Bulb | 9.99 |  | Tube light |
| 10 | Sony VAIO Core i7 17.3-Inch Laptop | 395 |  | Laptop |
| 11 | Sony VAIO EJ2 Series VPCEJ28FX/B 17.3-InchLaptop | 435 |  | Laptop |
| 12 | Sony vaio i7 laptop, 17 inch | 399 |  | Laptop |

Figure 7.1: Example collection of products. Records 1, 2 refer to Regular LED bulbs, 3, 4 to Smart LED bulbs, 5, 6 to halogen bulbs, 7, 8, 9 to tube lights and 10, 11, 12 to laptops.

7.1.3 Hierarchical Clustering with Supervision: Taxonomy Construction

In many applications, records (which refer to real-world entities) need to be organized in ways that capture entities and type relationships between entities. For example, a retail website that recognizes which records refer to the same product and organizes such products in the form of a product taxonomy can enable better search and recommendation. Consider a sample of products in Figure 7.1 collected from a retail website, and focus on records 6, 7, 8 and 9. A partial solution would be to identify that records 7 and 8 refer to the same entity – “Luxrite T8” tube light, while 6 and 9 refer to other products. A better solution would be to recognize that 9 - even if it refers to a different “F8T5/CW” tube light - is closer to 7 and 8 than

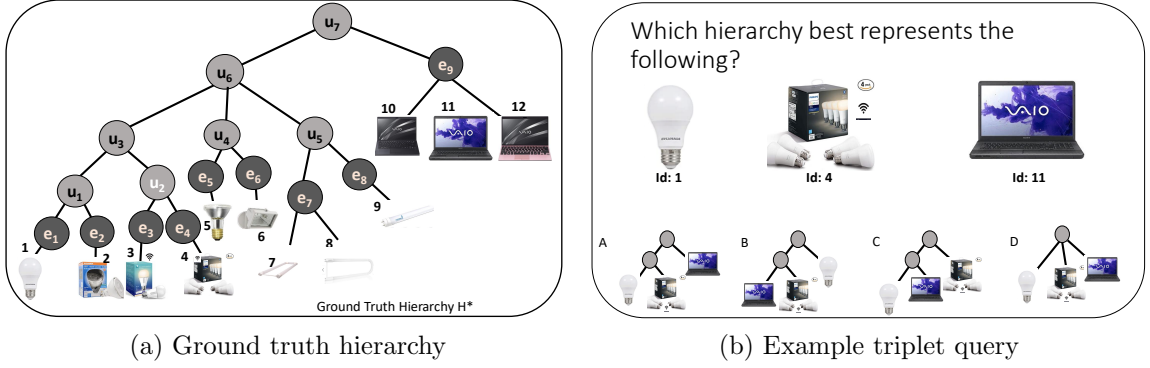


Figure 7.2: (a) Hierarchical relationships for products in Figure 7.1. Dark gray nodes like e_9 represent entities. E.g., records 10, 11 and 12 indeed refer to the same ‘Vaio laptop’ (denoted by e_9), sold by different vendors. Light gray nodes ($u_1 - u_7$) represent types. Deepest light gray nodes (u_1, u_2, u_4, u_5) which are parents of dark gray nodes, correspond directly to the “Type” column of Figure 7.1. E.g., u_2 corresponds to Smart LED. (b) Example of a triplet query about nodes 1, 4 and 11.

to 6, which refers to a halogen bulb which is of a different entity type. Even more, when considering other records such as 10, which refers to a laptop, we can identify that 6 is now closer to 7, 8 and 9 (which are all “lighting” items) than to 10 and so on, hierarchically. Figure 7.2a shows the entire set of relationships for the records in Figure 7.1. In this context, types can be thought of as clusters of entities, possibly included in other more general types, and entities can be thought of *sui generis* types consisting only of themselves. Reconstructing the entire set of hierarchical relationships can be challenging because types and entities may not be known a priori (i.e., some may need to be discovered during the process) and even the number of types or their size distribution may not be known; some types might have many entities in the dataset while others may be *niche* types with few entities.

As a future work, we plan to investigate a new *hierarchical ER* task, where records need to be clustered in a tree-like structure capturing entities as parents of leaf records and types from the level immediately above. In terms of supervision, we assume access to *oracles* that can answer the following types of queries.

- Binary optimal cluster query: “do records u and v refer to the same entity?”

- Comparison query: “which records among u , v and w are most similar?” (shown in Figure 7.2b).

The above query forms have been used separately for solving ER [194, 83, 87] and hierarchical clustering [79] but not for the proposed problem.

Limitations of using existing strategies. One approach to solve hierarchical ER could be to run ER first followed by hierarchical clustering (or vice versa). However, pipelining the two processes turns out to be sub-optimal. Let n be the number of records.

- Running a hierarchical clustering technique like [79] first and then post-processing the bottom level in order to detect entities can require $O(n^2)$ queries for non-binary hierarchies in the worst case, before even identifying the entities.
- Running an ER technique like [83] first and post-processing entities after that to detect types can be very efficient in case of large entities but can require $O(n^2)$ queries to identify small entity clusters, before even starting to process types.

Designing effective techniques to leverage both oracle models to recover the hierarchical representation of records is an interesting direction for future research.

7.1.4 Overlapping Clusters with Supervision

This dissertation focused on settings where the output clusters are disjoint. However, in some applications the ground truth clusters may be overlapping. For example, clustering group photos based on different individuals present in the photograph. Recent techniques have studied oracle models to recover overlapping clusters in the absence of pairwise similarity/probability values [45, 121]. Automated techniques to calculate similarity between such records can be noisy. Our ER techniques (Chapter 3) do not extend to these settings directly. The immediate next step is to study

supervision based techniques that can leverage similarity information between records to generate overlapping clusters.

7.1.5 Fair Clustering with Supervision

Using biased data in AI application development has had many disastrous consequences. For example, the COMPAS software used by the Wisconsin Supreme Court was found to erroneously consider a black defendant twice as likely as a white defendant to be a recidivist [8]. In another example, Amazon’s automated AI-based hiring tool infamously discriminated against women [2]. Given the societal impact of such applications, every effort of ensuring fairness in outcomes must be employed in the development process at every stage.

Clustering algorithms are known to be used for team formation, shortlisting resumes for hiring applications and many other applications that have societal impact [72]. Traditional clustering techniques are observed to produce biased results [55]. There has been a lot of research interest in developing fair techniques for clustering [55, 170, 36, 18, 35, 17]. However, all these techniques generate approximate solutions and are dependent on the input fairness constraints. Use of supervision to generate fair clusters has not been explored. We believe that supervision based techniques can also be helpful to reduce the dependence on accurate specification of fairness constraints and techniques for different fairness constraints [93].

7.1.6 Interpretable Clustering

Clustering results are expected to be inherently interpretable as the aim of clustering is to group similar nodes together. Many application domains are characterized by high-dimensional data and the interpretability may be diminished since no clear patterns may be easy to recognize for an end-user. There has been a growing interest in developing interpretable methods for machine learning based classifiers. However, there is limited prior research in improving interpretability of clusters [86, 171, 135].

Further, the increased importance of fairness along with other requirements like privacy and interpretability have justified the need to study multi-constraint clustering techniques. We believe that multi-constraint clustering is a challenging problem with a high potential of impact.

BIBLIOGRAPHY

- [1] <http://di2kg.dia.uniroma3.it/2019>.
- [2] Amazon scraps secret ai recruiting tool that showed bias against women. *ACM CACM*. <https://cacm.acm.org/news/231814-amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women/fulltext>.
- [3] Are ai hiring programs eliminating bias or making it worse? *Forbes*. <https://www.forbes.com/sites/nicolemartin1/2018/12/13/are-ai-hiring-programs-eliminating-bias-or-making-it-worse/#9456a7722b8f>.
- [4] Bad data costs the u.s. \$3 trillion per year. *Harvard Business Review*. <https://hbr.org/2016/09/bad-data-costs-the-u-s-3-trillion-per-year>.
- [5] Data never sleeps. *Domo*. <https://www.domo.com/solution/data-never-sleeps-6>.
- [6] Google vision api <https://cloud.google.com/vision>.
- [7] How to create a business case for data quality improvement. *Gartner*. <https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement/>.
- [8] How we analyzed the compas recidivism algorithm. *ProPublica*. <https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm>.
- [9] Making search easier. *Amazon Blog*. <https://blog.aboutamazon.com/innovation/making-search-easier>.
- [10] United states cities database.
- [11] Abbe, Emmanuel. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research* 18, 1 (2017), 6446–6531.
- [12] Abbe, Emmanuel, Bandeira, Afonso S., and Hall, Georgina. Exact recovery in the stochastic block model. *IEEE Trans. Information Theory* 62, 1 (2016), 471–487.

- [13] Abbe, Emmanuel, and Sandon, Colin. Community detection in general stochastic block models: Fundamental limits and efficient algorithms for recovery. In *56th Annual Symposium on Foundations of Computer Science (FOCS)* (2015), IEEE, pp. 670–688.
- [14] Abbe, Emmanuel, and Sandon, Colin. Recovering communities in the general stochastic block model without knowing the parameters. In *Advances in Neural Information Processing Systems* (2015), pp. 676–684.
- [15] Adamic, Lada A, and Glance, Natalie. The political blogosphere and the 2004 us election: divided they blog. In *3rd international workshop on Link discovery* (2005), ACM, pp. 36–43.
- [16] Addanki, Raghavendra, Galhotra, Sainyam, and Saha, Barna. How to design robust algorithms using noisy comparison oracle. In *PVLDB* (2021).
- [17] Ahmadi, Saba, Galhotra, Sainyam, Saha, Barna, and Schwartz, Roy. Fair correlation clustering. *arXiv preprint arXiv:2002.03508* (2020).
- [18] Ahmadian, Sara, Epasto, Alessandro, Kumar, Ravi, and Mahdian, Mohammad. Clustering without over-representation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), pp. 267–275.
- [19] Ailon, Nir, Bhattacharya, Anup, Jaiswal, Ragesh, and Kumar, Amit. Approximate clustering with same-cluster queries. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)* (2018), vol. 94, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, p. 40.
- [20] Ajtai, Miklós, Feldman, Vitaly, Hassidim, Avinatan, and Nelson, Jelani. Sorting and selection with imprecise comparisons. In *International Colloquium on Automata, Languages, and Programming* (2009), Springer, pp. 37–48.
- [21] Alon, Noga, and Spencer, Joel H. *The probabilistic method*. John Wiley & Sons, 2004.
- [22] Altowim, Yasser, Kalashnikov, Dmitri V, and Mehrotra, Sharad. Progressive approach to relational entity resolution. *PVLDB* 7, 11 (2014), 999–1010.
- [23] Angwin, Julia, Larson, Jeff, Mattu, Surya, and Kirchner, Lauren. Machine bias. *ProPublica May 23* (2016). <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing>.
- [24] Arora, Akhil, Sinha, Sakshi, Kumar, Piyush, and Bhattacharya, Arnab. Hd-index: pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *Proceedings of the VLDB Endowment* 11, 8 (2018), 906–919.

- [25] Ashtiani, Hassan, Kushagra, Shrinu, and Ben-David, Shai. Clustering with same-cluster queries. *NIPS* (2016).
- [26] Avin, Chen, and Ercal, Gunes. On the cover time and mixing time of random geometric graphs. *Theoretical Computer Science* 380, 1-2 (2007), 2–22.
- [27] Awasthi, Pranjali, Balcan, Maria, and Voevodski, Konstantin. Local algorithms for interactive clustering. In *International Conference on Machine Learning* (2014), PMLR, pp. 550–558.
- [28] Bachrach, Yoram, Graepel, Thore, Kasneci, Gjergji, Kosinski, Michal, and Van Gael, Jurgen. Crowd iq: aggregating opinions to boost performance. In *AAMAS* (2012), pp. 535–542.
- [29] Bae, Juhee, Helldin, Tove, Riveiro, Maria, Nowaczyk, Sławomir, Bouguelia, Mohamed-Rafik, and Falkman, Göran. Interactive clustering: a comprehensive review. *ACM Computing Surveys (CSUR)* 53, 1 (2020), 1–39.
- [30] Balcan, Maria-Florina, and Blum, Avrim. Clustering with interactive feedback. In *International Conference on Algorithmic Learning Theory* (2008), Springer, pp. 316–328.
- [31] Ball, Brian, Karrer, Brian, and Newman, Mark EJ. Efficient and principled method for detecting communities in networks. *Physical Review E* 84, 3 (2011), 036103.
- [32] Bansal, Nikhil, Blum, Avrim, and Chawla, Shuchi. Correlation clustering. *Machine Learning* 56, 1-3 (2004), 89–113.
- [33] Basu, Sumit, Fisher, Danyel, Drucker, Steven, and Lu, Hao. Assisting users with clustering tasks by combining metric learning and classification. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2010), vol. 24.
- [34] Bellare, Kedar, Iyengar, Suresh, Parameswaran, Aditya G, and Rastogi, Vibhor. Active sampling for entity matching. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* (2012), pp. 1131–1139.
- [35] Bera, Suman, Chakrabarty, Deeparnab, Flores, Nicolas, and Negahbani, Maryam. Fair algorithms for clustering. In *Advances in Neural Information Processing Systems* (2019), pp. 4955–4966.
- [36] Bercea, Ioana O, Groß, Martin, Khuller, Samir, Kumar, Aounon, Rösner, Clemens, Schmidt, Daniel R, and Schmidt, Melanie. On the cost of essentially fair clusterings. *arXiv preprint arXiv:1811.10319* (2018).
- [37] Bettstetter, Christian. On the minimum node degree and connectivity of a wireless multihop network. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing* (2002), ACM, pp. 80–91.

- [38] Bilenko, Mikhail, Kamath, Beena, and Mooney, Raymond J. Adaptive blocking: Learning to scale up record linkage. In *ICDM* (2006).
- [39] Bitton, Dina, and DeWitt, David J. Duplicate record elimination in large data files. *ACM Transactions on database systems (TODS)* 8, 2 (1983), 255–265.
- [40] Bollobás, Béla. Random graphs. In *Modern Graph Theory*. Springer, 1998, pp. 215–252.
- [41] Boucheron, Stéphane, Lugosi, Gábor, and Bousquet, Olivier. Concentration inequalities.
- [42] Bragg, Jonathan, and Weld, Daniel S. Optimal testing for crowd workers. In *AAMAS* (2016).
- [43] Braverman, Mark, Mao, Jieming, and Weinberg, S Matthew. Parallel algorithms for select and partition with noisy comparisons. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing* (2016), pp. 851–862.
- [44] Braverman, Mark, and Mossel, Elchanan. Noisy sorting without resampling. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms* (2008), Society for Industrial and Applied Mathematics, pp. 268–276.
- [45] Bressan, Marco, Cesa-Bianchi, Nicolò, Lattanzi, Silvio, and Paudice, Andrea. Exact recovery of mangled clusters with same-cluster queries. *arXiv preprint arXiv:2006.04675* (2020).
- [46] Bressan, Marco, Cesa-Bianchi, Nicolò, Paudice, Andrea, and Vitale, Fabio. Correlation clustering with adaptive similarity queries. In *Advances in Neural Information Processing Systems* (2019), pp. 12510–12519.
- [47] Brown, Eli T, Liu, Jingjing, Brodley, Carla E, and Chang, Remco. Dis-function: Learning distance functions interactively. In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)* (2012), IEEE, pp. 83–92.
- [48] Bubeck, Sébastien, Ding, Jian, Eldan, Ronen, and Rácz, Miklós Z. Testing for high-dimensional geometry in random graphs. *Random Structures & Algorithms* (2016).
- [49] Cabrereros, Irineo, Abbe, Emmanuel, and Tsirigos, Aristotelis. Detecting community structures in hi-c genomic data. In *2016 Annual Conference on Information Science and Systems (CISS)* (2016), IEEE, pp. 584–589.
- [50] Chai, Chengliang, Li, Guoliang, Li, Jian, Deng, Dong, and Feng, Jianhua. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD Conference* (2016), pp. 969–984.

- [51] Chatziafratis, Vaggos, Niazadeh, Rad, and Charikar, Moses. Hierarchical clustering with structural constraints. *arXiv preprint arXiv:1805.09476* (2018).
- [52] Chen, Jingchun, and Yuan, Bo. Detecting functional modules in the yeast protein–protein interaction network. *Bioinformatics* 22, 18 (2006), 2283–2290.
- [53] Chen, Lei, Lee, Dongwon, and Milo, Tova. Data-driven crowdsourcing: Management, mining, and applications. In *2015 IEEE 31st International Conference on Data Engineering* (2015), IEEE, pp. 1527–1529.
- [54] Chien, I, Pan, Chao, and Milenkovic, Olga. Query k-means clustering and the double dixie cup problem. In *Advances in Neural Information Processing Systems* (2018), pp. 6649–6658.
- [55] Chierichetti, Flavio, Kumar, Ravi, Lattanzi, Silvio, and Vassilvitskii, Sergei. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5029–5037.
- [56] Chin, Peter, Rao, Anup, and Vu, Van. Stochastic block model and community detection in the sparse graphs: A spectral algorithm with optimal rate of recovery. *arXiv:1501.05021* (2015).
- [57] Chittilappilly, Anand Inasu, Chen, Lei, and Amer-Yahia, Sihem. A survey of general-purpose crowdsourcing techniques. *IEEE Transactions on Knowledge and Data Engineering* 28, 9 (2016), 2246–2266.
- [58] Choudhury, Tuhinangshu, Shah, Dhruti, and Karamchandani, Nikhil. Top-m clustering with a noisy oracle. In *2019 National Conference on Communications (NCC)* (2019), IEEE, pp. 1–6.
- [59] Christen, Peter. Febrl-: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *KDD* (2008), pp. 1065–1068.
- [60] Christen, Peter. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering* 24, 9 (2011), 1537–1555.
- [61] Christen, Peter, Churches, Tim, and Hegland, Markus. Febrl—a parallel open source data linkage system. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2004), Springer, pp. 638–647.
- [62] Chuang, Jason, and Hsu, Daniel J. Human-centered interactive clustering for data analysis. In *Conference on Neural Information Processing Systems (NIPS). Workshop on Human-Propelled Machine Learning* (2014).
- [63] Ciceri, Eleonora, Fraternali, Piero, Martinenghi, Davide, and Tagliasacchi, Marco. Crowdsourcing for top-k query processing over uncertain data. *IEEE Transactions on Knowledge and Data Engineering* 28, 1 (2015), 41–53.

- [64] Clauset, Aaron, Newman, Mark EJ, and Moore, Cristopher. Finding community structure in very large networks. *Physical review E* 70, 6 (2004), 066111.
- [65] Cline, Melissa S, Smoot, Michael, Cerami, Ethan, Kuchinsky, Allan, Landys, Neri, Workman, Chris, Christmas, Rowan, Avila-Campilo, Iliana, Creech, Michael, Gross, Benjamin, et al. Integration of biological networks and gene expression data using cytoscape. *Nature protocols* 2, 10 (2007), 2366.
- [66] Coden, Anni, Danilevsky, Marina, Gruhl, Daniel, Kato, Linda, and Nagarajan, Meena. A method to accelerate human in the loop clustering. In *Proceedings of the 2017 SIAM International Conference on Data Mining* (2017), SIAM, pp. 237–245.
- [67] dal Bianco, Guilherme, Gonçalves, Marcos André, and Duarte, Denio. Bloss: Effective meta-blocking with almost no effort. *Information Systems* 75 (2018).
- [68] Dalvi, Niles, Dasgupta, Anirban, Kumar, Ravi, and Rastogi, Vibhor. Aggregating crowdsourced binary ratings. In *WWW* (2013), pp. 285–294.
- [69] Das, Sanjib, GC, Paul Suganthan, Doan, AnHai, Naughton, Jeffrey F, Krishnan, Ganesh, Deep, Rohit, Arcaute, Esteban, Raghavendra, Vijay, and Park, Youngchoon. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD* (2017).
- [70] Davidson, Susan, Khanna, Sanjeev, Milo, Tova, and Roy, Sudeepa. Top-k and clustering with noisy comparisons. *ACM Trans. Database Syst.* 39, 4 (Dec. 2015).
- [71] Decelle, Aurelien, Krzakala, Florent, Moore, Cristopher, and Zdeborová, Lenka. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E* 84, 6 (2011), 066106.
- [72] Deepak, P. Whither fair clustering. In *AI for Social Good Workshop. Harvard CRCS* (2020).
- [73] Devroye, Luc, György, András, Lugosi, Gábor, Udina, Frederic, et al. High-dimensional random geometric graphs and their clique number. *Electronic Journal of Probability* 16 (2011), 2481–2508.
- [74] Dong, Xin Luna, and Srivastava, Divesh. Big data integration. *Synthesis Lectures on Data Management* 7, 1 (2015), 1–198.
- [75] Driver, Harold Edson, and Kroeber, Alfred Louis. *Quantitative expression of cultural relationships*, vol. 31. Berkeley: University of California Press, 1932.
- [76] Dushkin, Eyal, and Milo, Tova. Top-k sorting under partial order information. In *Proceedings of the 2018 International Conference on Management of Data* (2018), pp. 1007–1019.

- [77] Dyer, Martin E., and Frieze, Alan M. The solution of some random np-hard problems in polynomial expected time. *Journal of Algorithms* 10, 4 (1989), 451–489.
- [78] Elmagarmid, Ahmed K, Ipeirotis, Panagiotis G, and Verykios, Vassilios S. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.* 19, 1 (2007), 1–16.
- [79] Emamjomeh-Zadeh, Ehsan, and Kempe, David. Adaptive hierarchical clustering using ordinal queries. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms* (2018), SIAM, pp. 415–429.
- [80] Fan, Wenfei, Jia, Xibei, Li, Jianzhong, and Ma, Shuai. Reasoning about record matching rules. *Proceedings of the VLDB Endowment* 2, 1 (2009), 407–418.
- [81] Feige, Uriel, Raghavan, Prabhakar, Peleg, David, and Upfal, Eli. Computing with noisy information. *SIAM Journal on Computing* 23, 5 (1994), 1001–1018.
- [82] Fellegi, Ivan P, and Sunter, Alan B. A theory for record linkage. *Journal of the American Statistical Association* 64, 328 (1969), 1183–1210.
- [83] Firmani, Donatella, Saha, Barna, and Srivastava, Divesh. Online entity resolution using an oracle. *PVLDB* 9, 5 (2016), 384–395.
- [84] Fisher, Jeffrey, Christen, Peter, and Wang, Qing. Active learning based entity resolution using markov logic. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2016), Springer, pp. 338–349.
- [85] Fortunato, Santo. Community detection in graphs. *Physics reports* 486, 3-5 (2010), 75–174.
- [86] Frost, Nave, Moshkovitz, Michal, and Rashtchian, Cyrus. Exkmc: Expanding explainable k -means clustering. *arXiv preprint arXiv:2006.02399* (2020).
- [87] Galhotra, Sainyam, Firmani, Donatella, Saha, Barna, and Srivastava, Divesh. Robust entity resolution using random graphs. In *Proceedings of the 2018 International Conference on Management of Data* (2018), pp. 3–18.
- [88] Galhotra, Sainyam, Firmani, Donatella, Saha, Barna, and Srivastava, Divesh. Beer: Blocking for effective entity resolution. In *SIGMOD* (2021).
- [89] Galhotra, Sainyam, Firmani, Donatella, Saha, Barna, and Srivastava, Divesh. Efficient and effective er with progressive blocking. In *VLDB Journal* (2021).
- [90] Galhotra, Sainyam, Golshan, Behzad, and Tan, Wang-Chiew. Adaptive rule discovery for labeling text data. *arXiv preprint arXiv:2005.06133* (2020).
- [91] Galhotra, Sainyam, Mazumdar, Arya, Pal, Soumyabrata, and Saha, Barna. Connectivity in random annulus graphs and the geometric block model. *CoRR abs/1804.05013* (2018).

- [92] Galhotra, Sainyam, Mazumdar, Arya, Pal, Soumyabrata, and Saha, Barna. The geometric block model. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).
- [93] Galhotra, Sainyam, Saisubramanian, Sandhya, and Zilberstein, Shlomo. Learning to generate fair clusters from demonstrations. *arXiv preprint arXiv:2102.03977* (2021).
- [94] Geissmann, Barbara, Leucci, Stefano, Liu, Chih-Hung, and Penna, Paolo. Sorting with recurrent comparison errors. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)* (2017), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- [95] Geissmann, Barbara, Leucci, Stefano, Liu, Chih-Hung, and Penna, Paolo. Optimal sorting with persistent comparison errors. In *27th Annual European Symposium on Algorithms (ESA 2019)* (2019), vol. 144, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, p. 49.
- [96] Geissmann, Barbara, Leucci, Stefano, Liu, Chih-Hung, and Penna, Paolo. Optimal dislocation with persistent errors in subquadratic time. *Theory of Computing Systems* 64, 3 (2020), 508–521.
- [97] Getoor, Lise, and Machanavajjhala, Ashwin. Entity resolution: theory, practice & open challenges. *PVLDB* 5, 12 (2012), 2018–2019.
- [98] Ghosh, Arpita, Kale, Satyen, and McAfee, Preston. Who moderates the moderators?: crowdsourcing abuse detection in user-generated content. In *EC* (2011), pp. 167–176.
- [99] Ghoshdastidar, Debarghya, Perrot, Michaël, and von Luxburg, Ulrike. Foundations of comparison-based hierarchical clustering. In *Advances in Neural Information Processing Systems* (2019), pp. 7454–7464.
- [100] Girdhar, Yogesh, and Dudek, Gregory. Efficient on-line data summarization using extremum summaries. In *2012 IEEE International Conference on Robotics and Automation* (2012), IEEE, pp. 3490–3496.
- [101] Girvan, Michelle, and Newman, Mark EJ. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99, 12 (2002), 7821–7826.
- [102] Goel, Ashish, Rai, Sanatan, and Krishnamachari, Bhaskar. Monotone properties of random geometric graphs have sharp thresholds. *Annals of Applied Probability* (2005), 2535–2552.
- [103] Gokhale, Chaitanya, Das, Sanjib, Doan, AnHai, Naughton, Jeffrey F, Rampalli, Narasimhan, Shavlik, Jude, and Zhu, Xiaojin. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD Conference* (2014), pp. 601–612.

- [104] Goldenberg, Anna, Zheng, Alice X, Fienberg, Stephen E, and Airoldi, Edoardo M. A survey of statistical network models.
- [105] Gonzalez, Teofilo F. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science* 38 (1985), 293–306.
- [106] Gravano, Luis, Ipeirotis, Panagiotis G, Jagadish, Hosagrahar Visvesvaraya, Koudas, Nick, Muthukrishnan, Shanmugaelayut, and Srivastava, Divesh. Approximate string joins in a database (almost) for free. In *VLDB* (2001), pp. 491–500.
- [107] Green Larsen, Kasper, Mitzenmacher, Michael, and Tsourakakis, Charalampos. Clustering with a faulty oracle. In *Proceedings of The Web Conference 2020* (New York, NY, USA, 2020), WWW '20, Association for Computing Machinery, p. 2831–2834.
- [108] Griffin, Gregory, Holub, Alex, and Perona, Pietro. Caltech-256 object category dataset.
- [109] Gruenheid, Anja, Dong, Xin Luna, and Srivastava, Divesh. Incremental record linkage. *PVLDB* 7, 9 (2014), 697–708.
- [110] Gruenheid, Anja, Nushi, Besmira, Kraska, Tim, Gatterbauer, Wolfgang, and Kossmann, Donald. Fault-tolerant entity resolution with the crowd. *arXiv preprint arXiv:1512.00537* (2015).
- [111] Guo, Stephen, Parameswaran, Aditya, and Garcia-Molina, Hector. So who won? dynamic max discovery with the crowd. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), pp. 385–396.
- [112] Gupta, Piyush, and Kumar, Panganamala R. Critical power for asymptotic connectivity. In *37th IEEE Conference on Decision and Control* (1998), vol. 1, IEEE, pp. 1106–1110.
- [113] Haenggi, Martin, Andrews, Jeffrey G, Baccelli, François, Dousse, Olivier, and Franceschetti, Massimo. Stochastic geometry and random graphs for the analysis and design of wireless networks. *IEEE Journal on Selected Areas in Communications* 27, 7 (2009).
- [114] Hajek, Bruce, Wu, Yihong, and Xu, Jiaming. Achieving exact cluster recovery threshold via semidefinite programming. *IEEE Transactions on Information Theory* 62, 5 (2016), 2788–2797.
- [115] Hajek, Bruce E., Wu, Yihong, and Xu, Jiaming. Computational lower bounds for community detection on random graphs. In *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015* (2015), pp. 899–928.

- [116] He, Ruining, and McAuley, Julian. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web* (2016), pp. 507–517.
- [117] Hernández, Mauricio A, and Stolfo, Salvatore J. The merge/purge problem for large databases. In *ACM Sigmod Record* (1995), vol. 24, ACM, pp. 127–138.
- [118] Hernández, Mauricio A, and Stolfo, Salvatore J. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data mining and knowledge discovery* 2, 1 (1998), 9–37.
- [119] Holland, Paul W, Laskey, Kathryn Blackmond, and Leinhardt, Samuel. Stochastic blockmodels: First steps. *Social networks* 5, 2 (1983), 109–137.
- [120] Hopkins, Max, Kane, Daniel, Lovett, Shachar, and Mahajan, Gaurav. Noise-tolerant, reliable active classification with comparison queries. *arXiv preprint arXiv:2001.05497* (2020).
- [121] Huleihel, Wasim, Mazumdar, Arya, Médard, Muriel, and Pal, Soumyabrata. Same-cluster querying for overlapping clusters. *arXiv preprint arXiv:1910.12490* (2019).
- [122] Ilvento, Christina. Metric learning for individual fairness. *arXiv preprint arXiv:1906.00250* (2019).
- [123] Jeffery, Shawn R, Franklin, Michael J, and Halevy, Alon Y. Pay-as-you-go user feedback for dataspace systems. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (2008), pp. 847–860.
- [124] Jeffery, Shawn R, Sun, Liwen, DeLand, Matt, Pendar, Nick, Barber, Rick, and Galdi, Andrew. Arnold: Declarative crowd-machine data integration. In *CIDR* (2013).
- [125] Karger, David R, Oh, Sewoong, and Shah, Devavrat. Iterative learning for reliable crowdsourcing systems. In *NIPS* (2011), pp. 1953–1961.
- [126] Kazemi, Ehsan, Chen, Lin, Dasgupta, Sanjoy, and Karbasi, Amin. Comparison based learning from weak oracles. *arXiv preprint arXiv:1802.06942* (2018).
- [127] Kim, Taewan, and Ghosh, Joydeep. Relaxed oracles for semi-supervised clustering. *arXiv preprint arXiv:1711.07433* (2017).
- [128] Kim, Taewan, and Ghosh, Joydeep. Semi-supervised active clustering with weak oracles. *arXiv preprint arXiv:1709.03202* (2017).
- [129] Klein, Rolf, Penninger, Rainer, Sohler, Christian, and Woodruff, David P. Tolerant algorithms. In *European Symposium on Algorithms* (2011), Springer, pp. 736–747.

- [130] Kleindessner, Matthäus, and Awasthi, Pranjali. Crowdsourcing with arbitrary adversaries. In *International Conference on Machine Learning* (2018), PMLR, pp. 2708–2717.
- [131] Kleindessner, Matthäus, Awasthi, Pranjali, and Morgenstern, Jamie. Fair k-center clustering for data summarization. In *International Conference on Machine Learning* (2019), pp. 3448–3457.
- [132] Konda, Pradap, Das, Sanjib, Suganthan GC, Paul, Doan, AnHai, Ardalan, Adel, Ballard, Jeffrey R, Li, Han, Panahi, Fatemah, Zhang, Haojun, Naughton, Jeff, et al. Magellan: Toward building entity matching management systems. *PVLDB* 9, 12 (2016), 1197–1208.
- [133] Kou, Ngai Meng, Li, Yan, Wang, Hao, U, Leong Hou, and Gong, Zhiguo. Crowdsourced top-k queries by confidence-aware pairwise judgments. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), pp. 1415–1430.
- [134] Krause, Jonathan, Stark, Michael, Deng, Jia, and Fei-Fei, Li. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)* (2013).
- [135] Laber, Eduardo, and Murtinho, Lucas. On the price of explainability for some clustering problems. *arXiv preprint arXiv:2101.01576* (2021).
- [136] Lei, Jing, Rinaldo, Alessandro, et al. Consistency of spectral clustering in stochastic block models. *The Annals of Statistics* 43, 1 (2015), 215–237.
- [137] Leskovec, Jure, Adamic, Lada A, and Huberman, Bernardo A. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)* 1, 1 (2007), 5.
- [138] Leskovec, Jure, Lang, Kevin J, Dasgupta, Anirban, and Mahoney, Michael W. Statistical properties of community structure in large social and information networks. In *17th international conference on World Wide Web* (2008), ACM, pp. 695–704.
- [139] Linden, Greg, Smith, Brent, and York, Jeremy. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.
- [140] Liu, Wei, Xue, Gui-Rong, Huang, Shen, and Yu, Yong. Interactive chinese search results clustering for personalization. In *International Conference on Web-Age Information Management* (2005), Springer, pp. 676–681.
- [141] Manning, Christopher D, Manning, Christopher D, and Schütze, Hinrich. *Foundations of statistical natural language processing*. 1999.

- [142] Mason, Blake, Tripathy, Ardhendu, and Nowak, Robert. Learning nearest neighbor graphs from noisy distance samples. In *Advances in Neural Information Processing Systems* (2019), pp. 9586–9596.
- [143] Mazumdar, Arya, and Saha, Barna. Clustering via crowdsourcing. *CoRR abs/1604.01839* (2016).
- [144] Mazumdar, Arya, and Saha, Barna. Clustering with noisy queries. In *Advances in Neural Information Processing Systems* (2017), pp. 5788–5799.
- [145] Mazumdar, Arya, and Saha, Barna. Query complexity of clustering with side information. In *Advances in Neural Information Processing Systems* (2017), pp. 4682–4693.
- [146] McCallum, Andrew, 2004. <http://www.cs.umass.edu/~mcallum/data/cora-refs.tar.gz>.
- [147] McCallum, Andrew, Nigam, Kamal, and Ungar, Lyle H. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of ACM SIGKDD international conference on Knowledge discovery and data mining* (2000), pp. 169–178.
- [148] McNeill, N, Kardes, Hakan, and Borthwick, Andrew. Dynamic record blocking: efficient linking of massive databases in mapreduce. Citeseer.
- [149] Milenkovic, Jovan. How many ecommerce sites are there in 2020? <https://kommandotech.com/statistics/how-many-ecommerce-sites-are-there/>.
- [150] Mossel, Elchanan, Neeman, Joe, and Sly, Allan. Consistency thresholds for the planted bisection model. In *47th Annual ACM Symposium on Theory of Computing* (2015), ACM, pp. 69–75.
- [151] Mudgal, Sidharth, Li, Han, Rekatsinas, Theodoros, Doan, AnHai, Park, Youngchoon, Krishnan, Ganesh, Deep, Rohit, Arcaute, Esteban, and Raghavendra, Vijay. Deep learning for entity matching: A design space exploration. In *SIGMOD* (2018).
- [152] Okabe, Masayuki, and Yamada, Seiji. An interactive tool for human active learning in constrained clustering. *Journal of Emerging Technologies in Web Intelligence* 3, 1 (2011), 20–27.
- [153] Olson, Parmy. The algorithm that beats your bank manager. *CNN Money March 15* (2011). <http://www.forbes.com/sites/parmyolson/2011/03/15/the-algorithm-that-beats-your-bank-manager/#cd84e4f77ca8>.
- [154] Papadakis, G., Koutrika, G., Palpanas, T., and Nejdl, W. Meta-blocking: Taking entity resolution to the next level. *TKDE* 26 (2014).

- [155] Papadakis, George, Alexiou, George, Papastefanatos, George, and Koutrika, Georgia. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *PVLDB* 9, 4 (2015), 312–323.
- [156] Papadakis, George, Ioannou, Ekaterini, Palpanas, Themis, Niederee, Claudia, and Nejdl, Wolfgang. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Transactions on Knowledge and Data Engineering* 25, 12 (2012), 2665–2682.
- [157] Papadakis, George, Mandilaras, George, Gagliardelli, Luca, Simonini, Giovanni, Thanos, Emmanouil, Giannakopoulos, George, Bergamaschi, Sonia, Palpanas, Themis, and Koubarakis, Manolis. Three-dimensional entity resolution with jedai. *Information Systems* 93 (2020), 101565.
- [158] Papadakis, George, Papastefanatos, George, and Koutrika, Georgia. Supervised meta-blocking. *PVLDB* 7 (2014).
- [159] Papadakis, George, Svirsky, Jonathan, Gal, Avigdor, and Palpanas, Themis. Comparative analysis of approximate blocking techniques for entity resolution. *Proceedings of the VLDB Endowment* 9, 9 (2016), 684–695.
- [160] Papadakis, George, Tsekouras, Leonidas, Thanos, Emmanouil, Giannakopoulos, George, Palpanas, Themis, and Koubarakis, Manolis. The return of jedai: end-to-end entity resolution for structured and semi-structured data. *PVLDB* 11, 12 (2018), 1950–1953.
- [161] Papenbrock, Thorsten, Heise, Arvid, and Naumann, Felix. Progressive duplicate detection. *Knowledge and Data Engineering, IEEE Transactions on* 27, 5 (2015), 1316–1329.
- [162] Penrose, Mathew. *Random geometric graphs*. No. 5. Oxford University Press, 2003.
- [163] Penrose, Mathew D. On a continuum percolation model. *Advances in applied probability* 23, 03 (1991), 536–556.
- [164] Polychronopoulos, Vassilis, De Alfaro, Luca, Davis, James, Garcia-Molina, Hector, and Polyzotis, Neoklis. Human-powered top-k lists. In *WebDB* (2013), pp. 25–30.
- [165] Prelec, Dražen, Seung, H Sebastian, and McCoy, John. A solution to the single-question crowd wisdom problem. *Nature* 541, 7638 (2017), 532–535.
- [166] Prolific. <https://www.prolific.co/>.
- [167] Qian, Kun, Popa, Lucian, and Sen, Prithviraj. Active learning for large-scale entity resolution. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (2017), pp. 1379–1388.

- [168] Ratner, Alexander, Bach, Stephen H, Ehrenberg, Henry, Fries, Jason, Wu, Sen, and Ré, Christopher. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases* (2017), vol. 11, NIH Public Access, p. 269.
- [169] Rohe, Karl, Chatterjee, Sourav, Yu, Bin, et al. Spectral clustering and the high-dimensional stochastic blockmodel. *The Annals of Statistics* 39, 4 (2011), 1878–1915.
- [170] Rösner, Clemens, and Schmidt, Melanie. Privacy preserving clustering with constraints. *arXiv preprint arXiv:1802.02497* (2018).
- [171] Saisubramanian, Sandhya, Galhotra, Sainyam, and Zilberstein, Shlomo. Balancing the tradeoff between clustering value and interpretability. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (2020), pp. 351–357.
- [172] Sarawagi, Sunita, and Bhamidipaty, Anuradha. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (2002), pp. 269–278.
- [173] Schütze, Hinrich, Manning, Christopher D, and Raghavan, Prabhakar. Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference* (2008), p. 260.
- [174] Shi, Jianbo, and Malik, Jitendra. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000), 888–905.
- [175] Sibson, Robin. Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal* 16, 1 (1973), 30–34.
- [176] Simonini, Giovanni, Bergamaschi, Sonia, and Jagadish, HV. Blast: a loosely schema-aware meta-blocking approach for entity resolution. *PVLDB* 9, 12 (2016).
- [177] Simonini, Giovanni, Papadakis, George, Palpanas, Themis, and Bergamaschi, Sonia. Schema-agnostic progressive entity resolution. *IEEE Transactions on Knowledge and Data Engineering* 31, 6 (2018), 1208–1221.
- [178] Sørlie, Therese, Perou, Charles M, Tibshirani, Robert, Aas, Turid, Geisler, Stephanie, Johnsen, Hilde, Hastie, Trevor, Eisen, Michael B, Van De Rijn, Matt, Jeffrey, Stefanie S, et al. Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proceedings of the National Academy of Sciences* 98, 19 (2001), 10869–10874.
- [179] Strickland, Eliza. Doc bot preps for the O.R. *IEEE Spectrum* 53, 6 (June 2016), 32–60.

- [180] Tamuz, Omer, Liu, Ce, Belongie, Serge, Shamir, Ohad, and Kalai, Adam Tauman. Adaptively learning the crowd kernel. In *Proceedings of the 28th International Conference on International Conference on Machine Learning* (2011), pp. 673–680.
- [181] Ukkonen, Antti. Crowdsourced correlation clustering with relative distance comparisons. In *2017 IEEE International Conference on Data Mining (ICDM)* (2017), IEEE, pp. 1117–1122.
- [182] <https://appen.com/>. Appen.
- [183] <https://modal-python.readthedocs.io/en/latest/>. modal library.
- [184] <https://scikit-learn.org/stable/>. Scikit-learn.
- [185] <https://www.mturk.com/>. Amazon mechanical turk.
- [186] Varma, Paroma, and Ré, Christopher. Snuba: automating weak supervision to label training data. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases* (2018), vol. 12, NIH Public Access, p. 223.
- [187] Vatsalan, Dinusha, Sehili, Ziad, Christen, Peter, and Rahm, Erhard. Privacy-preserving record linkage for big data: Current approaches and research challenges. In *Handbook of Big Data Technologies*. Springer, 2017, pp. 851–895.
- [188] Vazirani, Vijay V. *Approximation algorithms*. Springer Science & Business Media, 2013.
- [189] Venetis, Petros, Garcia-Molina, Hector, Huang, Kerui, and Polyzotis, Neoklis. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web* (2012), pp. 989–998.
- [190] Verdugo, Victor. Skyline computation with noisy comparisons. In *Combinatorial Algorithms: 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8–10, 2020, Proceedings*, Springer, p. 289.
- [191] Verroios, V., and Garcia-Molina, H. Entity resolution with crowd errors. In *ICDE Conference* (April 2015), pp. 219–230.
- [192] Verroios, Vasilis, Garcia-Molina, Hector, and Papakonstantinou, Y. Waldo: An adaptive human interface for crowd entity resolution. In *SIGMOD Conference* (2017).
- [193] Verroios, Vasilis, Garcia-Molina, Hector, and Papakonstantinou, Yannis. Waldo: An adaptive human interface for crowd entity resolution. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), pp. 1133–1148.
- [194] Vesdapunt, Norases, Bellare, Kedar, and Dalvi, Niles. Crowdsourcing algorithms for entity resolution. *PVLDB* 7, 12 (2014), 1071–1082.

- [195] Vinayak, Ramya Korlakai, and Hassibi, Babak. Crowdsourced clustering: Querying edges vs triangles. In *Advances in Neural Information Processing Systems* (2016), pp. 1316–1324.
- [196] Wang, Jiannan, Kraska, Tim, Franklin, Michael J, and Feng, Jianhua. Crowder: Crowdsourcing entity resolution. *PVLDB* 5, 11 (2012), 1483–1494.
- [197] Wang, Jiannan, Li, Guoliang, Kraska, Tim, Franklin, Michael J, and Feng, Jianhua. Leveraging transitive relations for crowdsourced joins. In *SIGMOD Conference* (2013), pp. 229–240.
- [198] Welinder, Peter, Branson, Steve, Belongie, Serge, and Perona, Pietro. The multidimensional wisdom of crowds. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems* (USA, 2010), NIPS’10, Curran Associates Inc., pp. 2424–2432.
- [199] Welinder, Peter, Branson, Steve, Perona, Pietro, and Belongie, Serge. The multidimensional wisdom of crowds. *Neural Information Processing Systems*.
- [200] Whang, Steven Euijong, and Garcia-Molina, Hector. Incremental entity resolution on rules and data. *The VLDB Journal* 23, 1 (Feb. 2014).
- [201] Whang, Steven Euijong, Lofgren, Peter, and Garcia-Molina, Hector. Question selection for crowd entity resolution. *PVLDB* 6, 6 (2013), 349–360.
- [202] Whang, Steven Euijong, Marmaros, David, and Garcia-Molina, Hector. Pay-as-you-go entity resolution. *Knowledge and Data Engineering, IEEE Transactions on* 25, 5 (2013), 1111–1124.
- [203] Whang, Steven Euijong, Menestrina, David, Koutrika, Georgia, Theobald, Martin, and Garcia-Molina, Hector. Entity resolution with iterative blocking. In *SIGMOD Conference* (2009), ACM, pp. 219–232.
- [204] Williamson, David P, and Shmoys, David B. *The design of approximation algorithms*. Cambridge university press, 2011.
- [205] Winkler, William E. Overview of record linkage and current research directions. In *Bureau of the Census* (2006), Citeseer.
- [206] Yang, Jaewon, and Leskovec, Jure. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (2015), 181–213.
- [207] Yang, Jingru, Fan, Ju, Wei, Zhewei, Li, Guoliang, Liu, Tongyu, and Du, Xiaoyong. Cost-effective data annotation using game-based crowdsourcing. *Proceedings of the VLDB Endowment* 12, 1 (2018), 57–70.

- [208] Zhang, Chao, Tao, Fangbo, Chen, Xiusi, Shen, Jiaming, Jiang, Meng, Sadler, Brian, Vanni, Michelle, and Han, Jiawei. Taxogen: Unsupervised topic taxonomy construction by adaptive term embedding and clustering. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018), pp. 2701–2709.