ABSTRACT

Title of Dissertation: MARKETS, ELECTIONS, AND MICROBES: DATA-DRIVEN ALGORITHMS FROM THEORY TO PRACTICE

Brian Brubach
Doctor of Philosophy, 2020

Dissertation Directed by: Professor Aravind Srinivasan
Professor Mihai Pop
Department of Computer Science

Many modern problems in algorithms and optimization are driven by data which often carries with it an element of uncertainty. In this work, we conduct an investigation into algorithmic foundations and applications across three main areas.

The first area is online matching algorithms for e-commerce applications such as online sales and advertising. The importance of e-commerce in modern business cannot be overstated and even minor algorithmic improvements can have huge impacts. In online matching problems, we generally have a known offline set of goods or advertisements while users arrive online and allocations must be made immediately and irrevocably when a user arrives. However, in the real world, there is also uncertainty about a user's true interests and this can be modeled by considering matching problems in a graph with stochastic edges that only have a probability of existing. These edges can represent the probability of a user purchasing a product or clicking on an ad. Thus, we optimize over data which only provides an estimate

of what types of users will arrive and what they will prefer. We survey a broad landscape of problems in this area, gain a deeper understanding of the algorithmic challenges, and present algorithms with improved worst case performance

The second area is constrained clustering where we explore classical clustering problems with additional constraints on which data points should be clustered together. Utilizing these constraints is important for many clustering problems because they can be used to ensure fairness, exploit expert advice, or capture natural properties of the data. In simplest case, this can mean some pairs of points have "must-link" constraints requiring that that they must be clustered together. Moving into stochastic settings, we can describe more general pairwise constraints such as bounding the probability that two points are separated into different clusters. This lets us introduce a new notion of fairness for clustering and address stochastic problems such as semi-supervised learning with advice from imperfect experts. Here, we introduce new models of constrained clustering including new notions of fairness for clustering applications. Since these problems are NP-hard, we give approximation algorithms and in some cases conduct experiments to explore how the algorithms perform in practice. Finally, we look closely at the particular clustering problem of drawing election districts and show how constraining the clusters based on past voting data can interact with voter incentives.

The third area is string algorithms for bioinformatics and metagenomics specifically where the data deluge from next generation sequencing drives the necessity for new algorithms that are both fast and accurate. For metagenomic analysis, we present a tool for clustering a microbial marker gene, the 16S ribosomal RNA gene.

On the more theoretical side, we present a succinct application of the Method of the Four Russians to edit distance computation as well as new algorithms and bounds for the maximum duo-preservation string mapping (MPSM) problem.

MARKETS, ELECTIONS, AND MICROBES: DATA-DRIVEN ALGORITHMS FROM THEORY TO PRACTICE

by

Brian Brubach

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Advisory Committee:
Professor Aravind Srinivasan, Chair/Advisor
Professor Mihai Pop, Co-Advisor
Professor John P. Dickerson
Professor David Mount
Professor Lawrence Ausubel

Dedication

For Miles.

Acknowledgments

I have to thank a number of people for their support, brilliance, and generosity. I certainly could not have done this alone.

My undergraduate mentor, Rajiv Gandhi, was the first to put me on this path toward a PhD. He inspired my interest in algorithms and research. His deep mentorship was invaluable and I would not have even made it to graduate school without him.

During my PhD, I received support from a number of people. Most notably, my advisors, Aravind Srinivasan and Mihai Pop, who taught me a lot about research, their areas of expertise, and much more. Beyond their wisdom, I am grateful to have had two advisors who always put my best interest first and foremost in our relationship.

In addition, John P. Dickerson was another mentor in research and very encouraging and helpful in my faculty job search. Srinivas Aluru and his lab at Georgia tech hosted me for a summer which helped me expand into bioinformatics and more applied research in general. I also want to thank my other dissertation committee members David Mount and Lawrence Ausubel as well as preliminary proposal committee members Samir Khuller and Jordan Boyd-Graber. On the teaching side, Nelson Padua-Perez, Fawzi Emad, and Pedram Sadeghian supported me in my journey to communicate my work and become a better teacher. I have enjoyed fruitful

discussions with these folks and all of the other wonderful professors I have met on this journey. I hope to have many more.

Throughout this time, my fellow students were another crucial support system. I especially want to thank my collaborators Darshan Chakrabarti, Jay Ghurye, Nathaniel Grammel, Karthik Abinav Sankararaman, Leonidas Tsepenekas, Pan Xu, and Shawn Zhao who are all co-authors of work presented here. There are many others including those I have just begun to collaborate with, my lab mates and desk mates, and everyone who joined me in organizing events for the computer science department.

Finally, I want to thank family and friends at UMD and elsewhere. It would have been impossible to survive this without them. Of course, I give extra special thanks to my wife Heather and son Miles, who give me a reason to work hard and do my best.

The research presented here was supported in part by NSF Awards (CNS 1010789, CCF 1422569, and CCF-1749864), the NIH (grant R01-AI-100947 to Mihai Pop), and research awards from Adobe, Inc., Amazon, and Google.

Table of Contents

Chapter 1: Introduction

This manuscript will cover a broad range of topics and techniques. To start with, we give a summary of motivations and background for the major problems we study.

Much of this work has already been published, sometimes twice with a highly polished and expanded journal version appearing after the conference presentation. At several points in later chapters, we will reference those papers rather than duplicate them in their entirety, which would take over 500 pages in this double-spaced format.

1.1 Online and Stochastic Matching

Matching problems in all of their variations capture a large number of important problems in an abstract, foundational way. The basic structure of these problems is straightforward. We have a set and we want to make pairs of elements. Typically, we are guided by some kind of graph with edges describing the value or cost of matching a pair of elements. In real world applications, the elements can be people, products, tasks, etc.

As much of our world moves to the internet, an increasing number of these

applications occur in e-commerce and online marketplaces. Algorithms are utilized to match users with products or advertisements, match workers to tasks in the gig economy, or even match people to each other in the case of online dating sites. Due to the enormous scale of these implementations, even minor algorithmic improvements can have major effects on efficient resource allocation and revenue.

One major challenge facing all of these applications is uncertainty in the input. What type of user will arrive at a website next? How likely are two people to form a longterm relationship if they go on a date? Advances in machine learning and data science can often help us predict the probabilities of these events, but how can our algorithms leverage these forecasts?

To address this challenge, we focus on bipartite matching with two notions of uncertainty, one concerning the vertex set and the other concerning the edge set. In *bipartite matching*, the underlying graph is bipartite and we must form pairs containing an element from each of the two partitions. Models of *online bipartite matching* establish an *offline* partition which is known up front and an *online* partition which arrives one-by-one. Each arriving vertex must be matched or discarded before the next is revealed, and there are numerous models capturing what we know about future arrivals. Similarly, the problem of *stochastic matching* adds uncertainty to the edge set. Each edge is assigned a known, independent, and distinct probability of existing. We must then *probe* an edge to find out if it exists before adding it to the matching. This notion of stochastic edges has been used to capture diverse applications including pay-per-click advertising, dating websites, and kidney exchange markets.

Our work seeks to gain a better fundamental understanding of these complex problems in addition to providing algorithms with improved worst case performance. Section 2.1 summarizes our main contributions at a high level. Chapter 3 gives a detailed description of some of these contributions along with survey of the landscape of problems and related work.

1.2 Constrained Clustering

Clustering is a fundamental problem studied in wide array of fields including machine learning, operations research, data science, and bioinformatics. The goal is to partition a set in way that optimizes some objective and respects a collection of constraints. The purpose of this is often to learn something about a data set or make decisions about how to allocate resources.

As with the previous section, meeting the needs of real world clustering applications often necessitates incorporating some form of stochasticity. In machine learning, we can use probabilistic constraints to model uncertain experts or leverage the power of randomness to generate fair decisions. Likewise, facility location problems in operations research may require solutions that anticipate forecasted demand in addition to serving current client needs. For the metagenomics applications introduced in Section 1.4, we attempt to cluster noisy genomic data which has been subjected to both biological mutations and errors in the DNA sequencing process. In of these applications, we use data to drive our clustering decisions, but it is only an estimate of the ground truth or future events.

The first clustering work we present deals with adding pairwise constraints to both old and new metric clustering problems. Most of our focus will be on probabilistic or soft must-link constraints, pairs of points which should be placed in the same cluster. Sometimes these must-link pairs derive from querying experts who may have some bounded error rate and the requirement is to satisfy a given fraction of them. Other times, these types of probabilistic constraints can arise naturally from the data set. In the lane finding problem, two nearby GPS data points known to come from the same car will have a high chance of being from the same lane since cars rarely change lanes on average. We will view these fundamental NP-hard problems from a theoretical perspective and show new approximation algorithms.

Beyond optimization and computational efficiency, another question we can and should ask is whether our algorithms are fair. When the data points we cluster correspond to real people, evidence has shown that people are not automatically treated equitably. In response to this challenge, there is a growing movement surrounding research into fairness, accountability, and transparency in automated systems.

A major focus of research at the intersection of fairness and optimization is demonstrating what is possible from a computational perspective and how we can minimize the cost of fair practices. In a sense, we cannot guarantee a right for all individuals unless we can show that guaranteeing that right is tractable and in some cases, affordable as well. The concept of disparate impact in United States law allows for practices that adversely affect a protected class provided the negative impact is not intentional and the practices serve a business necessity. So achieving a

balance between optimization and fairness can be an essential component of pushing for more equitable practices.

Along these lines, we investigate how constrained clustering can support fairness in clustering by introducing the notions of pairwise fairness and community preservation. Pairwise fairness uppers bound the probability that any pair of nearby points in the metric space gets separated into different clusters while community preservation ensures that dense communities of points are not split into many clusters. These fairness constraints are motivated by scenarios where data points represent people who gain some benefit from being clustered together. One example of this, explored in detail in the next section, is drawing political districts. The gerrymandering practice of "fracturing" involves dividing communities of voters into different districts in order to disenfranchise them. Another example is preserving community cohesion in public school assignment where students benefit from attending the same school as their neighbors, but the neighborhood school model is rife with inequality. In all cases, we are faced with a stochastic problem because randomization is required to minimize the maximum negative impact while achieving reasonable optimization goals.

As with the matching problems discussed, our work aims to build a fundamental understanding of these problems and introduce relevant new models in addition to providing algorithms with improved worst case performance. Section 2.2 summarizes our main contributions at a high level. Chapter 4 gives a detailed description our contributions.

1.3 Elections, Redistricting, and Gerrymandering

One unique clustering problem that we give special attention to is the drawing of U.S. congressional districts. In this problem, voters within a state are partitioned into clusters called districts that each elect a congressional representative. A major focus of our work on fairness here is the challenge of how to combat partisan gerrymandering. Gerrymandering refers to drawing electoral district maps to manipulate the outcomes of elections. Partisan gerrymandering in particular occurs when political parties use this practice to gain an advantage over another party (e.g., winning more seats in the US House of Representatives). One path to addressing this issue is developing algorithmic tools for drawing fairer districts. Another is measuring and regulating unfair practices. Thus, this problem now sits at the intersection algorithms, fairness, law, mathematics, and machine learning.

When designing new techniques for drawing districts, there are many pitfalls including: prioritizing fairness to political parties over citizens, incompatibility with real world election law, inadvertent discrimination by algorithms that optimize purely mathematical objectives, and failure to account for downstream effects. One small step we take to avoid these pitfalls is the fair clustering work introduced in the previous section and elaborated on in Sections 2.2 and 4.2. In that work, we develop new notions of fairness in this context that prioritize the fair treatment individuals and communities of people. This is in line with how some US Supreme Court justices have argued for the unconstitutionality of certain types of gerrymandering.

On the subject of measuring and regulating partisan gerrymandering, there

has been of a flurry of exciting recent activity and progress. This has led to computer scientists serving as experts in court cases going all the way up to the US Supreme Court. One of the most promising metrics uses past voting data and random sampling of hypothetical district maps to identify a gerrymandered map by showing that it is an outlier. This argument has now been used successfully in state court cases leading Pennsylvania and North Carolina to redraw their district maps.

Our contributions to the study of gerrymandering measurement and regulation are summarized in Section 2.3 and detailed in Chapter 5. We look ahead at the downstream effects of the most successful recent approaches that rely on Markov Chain Monte Carlo sampling of the space of legal maps. We explore how methods that rely on past voting data for districting regulation can affect voter incentives and lead to strategic voting even in two party elections. To show this, we propose a game theoretic model that captures the process of iteratively drawing districts and voting while subject to anti-gerrymandering regulation. This also reveals how strategic voting or election tampering could circumvent regulation. Finally, we use our models to better understand issues discussed in a recent U.S. Supreme Court case that consider whether sampling approaches are a proxy for less desirable measures what it means for a district map to be fair to an individual.

1.4 Bioinformatic, Genomics, and Metagenomics

The deluge of genomic data drives a constant need for new bioinformatics tools and a deeper theoretical understanding of the underlying computational prob-

lems. Larger and larger datasets up to billions of DNA sequencing reads [1] demand ever more efficient analytical techniques. Plus, surprising challenges arise, such as a growing reference database of microbes (RefSeq) leading to fewer species level classifications [2]. These challenges raise new questions about what is computationally possible and how to tailor algorithms to biological problems.

Beyond the scalability issues of growing data sets, we again face issues of random events affecting our input data. Strings representing sequenced DNA have been subjected to both biological mutations and errors in the sequencing process. This motivates the study of how to measure the distance between two strings in terms of insertions, deletions, substitutions, and even rearrangements where entire substrings can be relocated.

On the positive side, high-throughput next generation sequencing and long-read third generation sequencing have changed the way we study biology and the computational problems they introduce are fascinating. One subfield which has blossomed on the backs of these technologies is *metagenomics*, analyzing environmental samples of genetic material. Metagenomics allows us to explore the vast array of bacteria that cannot be analyzed through traditional culturing approaches. It also provides a glimpse at the diversity and interactions within specific environments from the human gut to the depths of the ocean to the international space station. Beyond exploratory research, it has promising applications in clinical diagnostics, biodefense, and food safety [3].

The first work presented in Chapter 6 focuses on computational solutions to one specific application in metagenomics, the clustering of 16S rRNA genes. The

16S rRNA gene is a highly conserved ribosomal gene present in all known bacteria. The slow rate of evolution in this gene makes it an ideal marker gene for taxonomic studies. Although not as accurate as recent whole genome shotgun sequencing approaches, 16S rRNA gene analysis is faster, cheaper, easier, and remains widely used.

The remainder of Chapter 6 addresses more general string comparison problems. These problems including computing edit distance (Levenshtein distance) and the maximum duo-preservation string mapping are relevant to genomics, but studied from a theoretical computer science perspective with provable bounds on performance.

Chapter 2: Summary of Contributions

In this chapter, we briefly summarize our main contributions at a high level. Detailed descriptions of our contributions with formal statements of the problems, algorithms, bounds, and other technical contributions are reserved for the following chapters.

2.1 Matching under Uncertainty in Online and Stochastic Settings

Most approaches to online matching in both prior work and ours use a linear programming relaxation to upper bound the optimal offline solution. This serves as

a benchmark for analyzing the online algorithm and a guide for its decisions as well. In the model of known IID arrivals, we essentially start with a known distribution on which vertices will arrive online and we can actually solve a linear program using this distribution as input prior to the online phase of the algorithm.

However, constructing appropriate linear programs for these stochastic problems involves two major challenges that interact with each other. The first is how to get a tight bound on a stochastic problem when we are essentially relaxing it to a deterministic problem (optimizing a linear system of equations). The second is how to use a given linear program to guide our algorithm and/or analysis.

To address the first challenge, we characterize the ways in which linear programs from prior work are weak upper bounds and show how to tighten them. For the second challenge, we show how to attenuate and round linear program solutions to guide our algorithms to good worst case performance. We also present an approach which departs from prior work by using dynamic programming to guide the algorithm and relying on a novel exponentially-sized linear program only for the analysis.

We organize our summary of online matching contributions as well as the detailed results of Chapter 3 into two broad categories: problems which only have uncertainty in the vertex arrivals (Section 2.1.1) and problems which also have uncertainty in the edge realizations (Section 2.1.2).

2.1.1 Online Bipartite Matching

In online bipartite matching problems, one set of vertices is known while the other arrives one-by-one. Each arriving vertex must be matched or discarded before the next arrival is revealed. Our work in Section 3.3 focuses on the known IID arrival model. In the simplest form of this model, we are given a bipartite graph with an offline partition that is a fixed set of vertices and an online partition that represents a distribution on vertex types. Each arrival is sampled with replacement from the online partition according to a known, independent, and identically distributed distribution. We give the current best theoretical bounds for several variations of the online matching with known IID arrivals using a few key technical contributions.

Our approaches to these problems use tighter LP benchmarks than some prior work by adding constraints that closely account for the probability that a given vertex in the online partition may never arrive. More importantly, we show how to leverage solutions to these LPs to guide our online algorithms. The key techniques that achieve this are careful applications of correlated randomized rounding and modification of the initial LP solution to balance for the worst case.

Our extension of the dependent rounding technique of [4] to suit these problems allows us to round an arbitrary fractional LP solution into a sparse integral multi-graph. This new graph violates the LP constraints, but satisfies several beneficial properties and can be used to guide the online algorithm. Our rounding approach also allows for the application of a modification procedure prior to rounding. Essentially, we massage the solution a bit to improve the worst case performance by

helping some LP variables and hurting others.

2.1.2 Stochastic Rewards and Patience

In the stochastic rewards model of online matching, the edges incident to an arriving vertex each have known and independent probabilities of existing. When we attempt to match an edge, we first probe it to find out if it exists. If it exists, we add it to the matching. Otherwise, we may continue probing additional edges until we have exhausted the patience of the online vertex. The patience of a vertex is an upper bound on the number of incident edges we may probe. Each online vertex may have an arbitrary known patience, but in some models, we assume all patience values are one.

Similar to the previous section, many existing approaches upper bound the optimal solution to these stochastic problems with a deterministic problem that has a simple linear programming relaxation. This is sometimes called the budgeted allocation LP. The main idea is to convert all probabilities of existence into deterministic fractional sizes and think of it as a classical packing problem. So an edge with probability 0.3 of existing is thought of as having size 0.3 and every vertex is like a container or knapsack with a capacity of 1.

Our work in Section 3.4 seeks to understand and address the limitations of these deterministic LP benchmarks. On the negative side, we introduce the concept of a stochasticity gap. Analogous to an integrality gap, the stochasticity gap of a linear program is a ratio comparing the performance of an optimal algorithm

for a stochastic problem to the objective value of a linear program which assumes deterministic input. We then prove stochasticity gaps showing that some online algorithms are tight or nearly tight with respect to their specific LP benchmark. This implies that LPs which upper bound the problem more tightly are need for improvement.

On the positive side, we present algorithms for online stochastic matching problems with good worst case performance. Some of these algorithms still use fairly standard linear programs, which we have proven stochasticity gaps for. However, we also design an algorithm which takes a wholly different approach to address the problem where vertices from an unknown graph arrive in an adversarial order.

In that work, we make key observations about the properties of an optimal probing algorithm. This leads to a dynamic programming algorithm which can solve the offline stochastic matching problem optimally on star graphs, the first optimal solution to any stochastic matching problem. Then, we can view each arriving online vertex as the center of a stochastic star graph and use the optimal star graph algorithm to guide our online decisions. To benchmark this approach, we devise a new exponentially-sized LP and compare the performance of our algorithm to the LP without actually solving it. Additionally, this approach is the first greedy algorithm presented for the online matching variant with stochastic rewards with patience constraints. We note this because greedy algorithms have been crucial to achieving good results in empirical studies of non-stochastic online matching problems [5].

2.2 Clustering Algorithms with Constraints and Fairness Guarantees

As with Chapter 4, we separate this summary of contributions into our work on constrained clustering and our work on fairness in clustering. In both cases, our main contributions include proposing new models and designing approximation algorithms for NP-hard problems. We note that in classical versions of centroid-based clustering problems such as k-center, k-median, etc., points are always assigned to the nearest center. However, in the problems we discuss, a point may be assigned to some other farther-away center in order to satisfying some additional constraints.

2.2.1 Constrained Clustering in Metric Spaces

One of the simplest models of pairwise constraints applied to a clustering problem is k-center with hard must-link constraints. We are given a list of pairs of points with each pair forming a constraint that those two points must be added to the same cluster. Our goal is to find a set of k centers and assignments to them that minimizes the maximum radius while ensuring that each pair of points is added to the same cluster. The soft must-link variant allows us to violate a given fraction of these constraints in order to further minimize the objective function.

Our work addresses hard and soft must-link constraints as well as two new models that we introduce: bounded separation probabilities and the constrained "no-k"-center problem. In bounded separation probabilities, pairs of points may be given an upper bound on the probability that they are separated into different clusters (a probability of 0 would represent a hard must-link constraint). The "no-

k"-center problem is just like the previously described constrained k-center problems except that we have no limit to the number of centers we can choose. Our goal is to choose any number of centers to minimize the maximum radius while respecting linkage constraints.

Most of these problems inherent NP-hardness from their unconstrained counterparts with the exception of "no-k"-center. Note that the unconstrained version of "no-k"-center admits a trivial solution (every point assigned to itself). Similarly, having only only must-link constraints without the objective to minimize the maximum radius (i.e., correlation clustering without negative edges) also has a trivial solution (place all points in a single cluster). Nevertheless, we prove that "no-k"-center with must-link constraints is in fact NP-hard.

Since we are dealing with intractable problems, we take an approximation algorithms approach. We present LP rounding algorithms that achieve bicriteria approximations with small constant approximations. The two criteria we approximate are the radius and a violation of the linkage constraints. We use a new LP formulation combined with a correlated rounding procedure from [6]. For the "no-k" problems, we also design a careful reassignment procedure after rounding the LP solution in order to find a feasible solution to the original problem. In addition, we are able to extend these results to related variants such as k-supplier and "no-k"-median.

2.2.2 Pairwise Fairness and Community Preservation

Our work on fairness in clustering introduces the new concepts pairwise fairness and community preservation. We complement this by presenting an algorithm that achieves fairness under those definitions while bounding the loss to the objective function. Our simple algorithm is easy to implement and can be used to augment any existing algorithm for the classical "unfair" k-center problem. Essentially, we take a set of clusters and grow them according to an exponential distribution, thus bounding the probability that the edge of a cluster will separate nearby points. In additions to theoretical results, we perform experiments on classical benchmark data sets from the optimization literature as well as a real data set commonly used to evaluate other work in the fairness space. These experiments illustrate how tweaking some parameters of our algorithm can lead to good trade-offs between optimization and fairness in practice.

2.3 Redistricting and Gerrymandering Regulation

The work presented in Chapter 5 looks closely at the practice of measuring and regulating gerrymandering using past voter data. Specifically we focus on approaches that sample (approximately) from the space of legal district maps and use past voting records to determine which maps are outliers in terms of estimated election outcomes. To analyze the effects of using these measurements in regulation from the perspective of social choice theory, we create a game to model the iterative process of alternately redrawing districts and voting. Using our new tools, we can

show how strategic voting can occur in a way that deviates from some common understandings such as the Gibbard-Satterthwaite theorem. We demonstrate this both theoretically and empirically. We also extend our experiments to respond to some questions in a recent U.S. Supreme Court cases where gerrymandering measurement was hotly debated.

Informally, the famous Gibbard-Satterthwaite theorem states the following for elections that choose a single winner deterministically. One of these three things must be true: there is only one voter (dictatorship), there are only two candidates/parties competing, or voters can be incentivized to vote strategically. Thus, we often think of a two party race as being immune to strategic voting. However, one key difference we show from the scenario addressed by Gibbard-Satterthwaite is that the step of drawing districts in the U.S. electoral system allows each voter to affect additional elections beyond the one they are voting in. This violates the assumption of a single winner that Gibbard-Satterthwaite relies on.

Our first contribution is devising a simple game that captures a series of elections between two parties with redistricting occurring in between rounds of voting. In our game, the majority party draws the districts, but is forbidden from drawing an "outlier map". Outlier maps are identified by looking at the space of all legal maps and the expected election outcomes given the last voting round. If a map produces an outcome that is rare, it is labeled an outlier. This allows us to show theoretically how strategic voting can allow a party to draw a more favorable map and therefore win more elections.

Using the insights from analyzing our game, we develop a heuristic for discov-

ering strategies We apply this heuristic to a huge set of voter configurations within a small toy problem as well as real North Carolina voting data. Our results reveal many scenarios where a strategy can be found although our real data experiments use a more restricted model to be able to process the large data set.

Beyond those main results, we address some questions from the oral arguments of Rucho v. Common Cause [7]. First, several justices suggest that sampling-based approaches to measuring gerrymandering are merely proxies for a proportionality test. However, we show that for some arrangements of voters, the measurement tools are the opposite of a proportionality test. Another line of questioning explored the following definition of fairness. Suppose most district maps place a person in a district that elects their preferred party/candidate. We show that no single map can give such a guarantee for all voters. This motivates the study of randomized methods that propose a fair distribution on maps as with the concepts of pairwise fairness and community preservation explored in our clustering work.

2.4 String Algorithms for Bioinformatics

The work presented in Chapter 6 overlaps with the areas of matching and clustering. However, it is unified by the unique properties of problems involving strings and genomic motivations. While gene sequence clustering in our model is a problem of constrained clustering in a metric space, our main contributions involve efficiently computing banded alignments between large groups of relatively short strings. Similarly, the maximum duo-preservation string mapping problem is

closely related to bipartite 4-uniform hypergraph matching, but has a very particular structure due to the fact that each partition is defined by a string.

2.4.1 16S rRNA Gene Clustering

The clustering of 16S ribosomal RNA genes is a useful step in understanding metagenomic samples. Two high-level approaches to this problem are *reference-based* where we utilize a database of known genomes and *de novo* where we cluster gene sequences into operational taxonomic units (OTUs) without consulting a database. We focus on de novo approaches which can better identify novel organisms and avoid biases in reference databases. The goal of most OTU clustering tools is to group sequences into clusters of bounded radius based on a sequence similarity function.

We have implemented a tool for clustering these genes using a similarity function based on edit distance (aka Levenshtein distance). Edit distance is a well-studied problem that counts the minimum number of insertions, deletions, and substitutions needed to transform on string into another. The best known theoretical algorithm for this problem uses a technique called the Method of the Four Russians, a trick for speeding up some dynamic programming algorithms. While this approach is not typically used to compute edit distance in practice, we borrow ideas from it to develop a tool that scales to handle large data sets.

The main bottleneck in scaling our clustering approach is computing the edit distance between one sequence and all other sequences individually. To alleviate this, our new data structure minimizes the amount of duplicated work by collapsing

similar prefixes and storing the outcomes of previous computations. Experiments show that our tool is scalable while produces high-quality clusters.

2.4.2 More Succinct Method of the Four Russians for Edit Distance

Following the work of the previous section, we developed some theoretical improvements to the space-efficiency. While this work is theoretical, space-efficiency is an important challenge face when implement the Method of the Four Russians in practice. Our work dramatically reduces the space needed at the expense of slightly slower running time and can be combined with an existing space reduction technique that is complementary.

2.4.3 The Maximum Duo-preservation String Mapping Problem

The maximum duo-preservation string mapping problem is an NP-hard and APX-hard problem comparing two strings which are permutations of each other. It is one of many ways to compute a distance between strings in terms of rearrangements and the complement to the well-studied minimum common string partition problem.

Over the course of two publications, we introduced a new matching-based approach to attack this problem [8,9]. This yielding the first approximation algorithm with a running time linear in the length of the strings for the unweighted variant of the problem with constant-sized alphabets and the current best approximation for the weighted version [8,9].

While most prior work has leveraged local search techniques, we introduce

a tractable intermediate problem based on matching triplets of characters. This intermediate problem is derived from a unique approach to making the conflict graph claw-free by both removing and adding constraints. The resulting tractable problem yields a solution the is closer to a feasible solution to original problem than if we had only removed constraints. Further, we can still show that the value of this solution is close that of an optimal solution to the original problem.

Chapter 3: Matching under Uncertainty

Following the ubiquity of the internet, e-commerce has become an enormous part of the economy. One representative application in this area is internet advertising where companies (e.g. Google, Facebook) generate revenue by matching advertisements to users. Another is online sales in two-sided matching markets (e.g. E-Bay, Amazon). In most cases, companies are repeatedly trying to optimize a large matching problem and even minor improvements can have massive effects on user satisfaction and revenue. We now give an introduction to the online matching abstractions of these problems and how uncertainty can be added to enhance the models.

3.1 Introducing $3 \times 3 \times 3 = 27$ Models

We discuss online matching across three arrival models, three types of objectives/weighting, and three variations on matching. In the most basic outline of the online matching setting, we have a known set of offline vertices U and a set of online vertices V. Over a series of rounds, a single vertex online v from V arrives online in some fashion and the edges incident on v are revealed. We must immediately and irrevocably match v to some vertex in U or choose not to match it before the next

online vertex arrives. For simplicity, we generally assume throughout this document that $|U| = |V| = n$ and the number of rounds in which an online vertex arrives is also n. In the context of e-commerce, we can think of the offline set as advertisements or items for sale and the online set as users arriving to a website.

The common objectives studied are unweighted, vertex-weighted, and edge-weighted. For *unweighted* graphs, our goal is to maximize the size of the matching. In the *vertex-weighted* case, each offline vertex $u \in U$ has a nonnegative weight w_u and we wish to maximize the weight of the offline vertices matched. Note that the online vertices typically do not have a weight associated with them in this model. Finally, in an *edge-weighted* graph, each edge $e = (u, v) \in U \times V$ has a nonnegative weight w_e and our objective is again to maximize the weight of the matching. In a practical sense, vertex weights can represent items being sold at a fixed cost (sometimes called posted prices) while edge weights can represent users willing to pay different amounts for the same item. One can see that the edge-weighted objective generalizes vertex-weighted, which in turn generalizes unweighted.

Incorporating stochasticity into online matching models allows us to capture real world situations where outcomes are uncertain, but we have some prediction about what will happen. There two common ways that stochasticity appears in these models. We outline them below and note that in the literature, the term "stochastic" is used rather loosely to describe any randomness in the models. Thus, we'll define specific terms for each element of randomness we consider.

The first introduction of randomness historically is in the arrival model. Online matching problems were originally studied under an *adversarial arrival* model where

the edges incident to each arriving vertex are chosen by some adversary. However, this can be too pessimistic to represent many e-commerce applications. Thus, other arrival models were developed. In random order arrival, an adversary can fix the underlying bipartite graph, but the online vertices arrive in a random order. In the *known IID arrival* model, the online vertices are sampled with replacement from a known independent and identically distributed distribution on vertex "types". The idea of this final model is that we are given a bipartite graph upfront where U is still the offline set, but now V represents known types of vertices that can arrive online. Each round, the online vertex is sampled uniformly with replacement from V. Thus, the same vertex $v \in V$ may arrive multiple times, but we treat each arrival as a separate vertex to be matched. We note that sampling uniformly from V is defined specifically as the integral arrival rates model and for simplicity, we only discuss this variation here. It is known that an algorithm for adversarial arrival will achieve the same performance or better in the other two models and an algorithm for random order will achieve the same performance or better with known IID arrivals.

The other way in which randomness appears is in the probability of edges existing. In the offline version of this problem, referred to as *stochastic matching*, each edge e has a known probability p_e of existing. Instead of simply matching an edge, we must first *probe* it to find out if it exists. With probability p_e it exists and is matched. Otherwise, it does not exist and we may not probe it again. In this work, we only consider the *probe-commit* model where an edge found to exist must be matched immediately and no further probes of its endpoints are allowed. In the online setting, this problem was introduced as "online matching with stochastic

rewards". An online vertex arrives and the p_e values of its neighborhood are revealed. We choose at most one neighbor u to probe, and if we are successful, the edge is matched as in classical matching. Otherwise, v is discarded and another online vertex arrives. In the case of known IID arrivals, each copy of the same vertex type that arrives is considered to be a distinct vertex with a distinct edge set, and thus, the same u may be probed by multiple copies of some v until one of them is successful. Naturally, stochastic matching generalizes the non-stochastic case since we could have all $p_e \in \{0, 1\}$.

An even more general version is stochastic matching with *patience* constraints (sometimes called *timeouts*). In this case, each online vertex $v \in V$ has patience t_v. This means we are allowed to probe at most t_v neighbors of v (we still must stop probing sooner if we encounter a match). This generalizes the previous model where each online vertex can be thought of as having $t_v = 1$. Both versions of stochastic edges capture the idea that we do not know for sure if a match will be successful. The probability p_e can represent the estimated likelihood of a user clicking on an ad (the pay-per-click revenue model) or purchasing a specific product. The patience may capture the number of ad slots available or the number of items a shopper will view before becoming bored and leaving the market. In the offline setting, stochastic edges have been used for diverse real world problems from kidney exchange to dating services, where patience refers to the literal amount of patience a person has to go on different dates before abandoning the service.

In keeping with the offline definition, we will use the terms "stochastic matching" and "stochastic rewards" interchangeably. We note that many works use "stochas-

tic matching" to refer to the known IID arrival model. However, we use the term "known IID arrival" specifically when referring to this model.

Observe that patience generalizes stochastic rewards and that both generalize the classical non-stochastic model. Another, more general model of stochasticity is presented in [10]. In their model, when a vertex v (viewed as a customer) arrives online, an online algorithm chooses a *set* S of potential matches for v (viewed as an offering of products to the customer). Each customer (online vertex) has a *general choice model* which specifies the probability of the customer purchasing each item when offered each possible set of product assortments S. We discuss this model in more detail in Section 3.4.1, but note that in this setting, a set of potential matches is chosen *all at once* rather than probed sequentially, with the outcome being determined by full set S (the offered product assortment).

Competitive Ratio. For all of these models, theoretical analysis is done using the common notion of a *competive ratio* for online algorithms. This is the ratio of the expected performance of an online algorithm to the expected performance of an optimal offline algorithm. More formally, this is defined as $\frac{\mathbb{E}[\text{ALG}]}{\mathbb{E}[\text{OPT}]}$. Here, $\mathbb{E}[\text{ALG}]$ is the expected performance of our online algorithm with respect to the random online vertex arrivals and any internal randomness the algorithm may use as well as random edge realizations for the stochastic rewards variants. Similarly, $\mathbb{E}[\text{OPT}]$ is the expected performance of an optimal offline matching algorithm which knows the random vertex arrivals in advance. In the case of stochastic rewards, we compare to an optimal offline stochastic matching algorithm which can probe edges in any order, but does not know the outcomes of these probes and is subject to patience constraints

on the online set. We note that for the stochastic matching problems, we do not have polynomial time algorithms for the offline problems. So we generally compare to an upper bound on the offline optimal solution as in the typical approach to bounding approximation algorithms. While it may seem pessimistic to benchmark an online algorithm against an offline one, we note that empirical average case analysis can yield competitive ratios over 0.9 and in some cases approaching 1 [5].

3.2 Related Work

Figure 3.1 summarizes the state-of-the-art for the 27 models defined in the introduction along with indications of where we have provided improved bounds. The book by Mehta [11] gives a detailed (slightly outdated) overview of an even wider landscape of problems in this area.

Adversarial Arrival. The study of online matching began with the seminal work of Karp, Vazirani, Vazirani [12]. They gave an optimal online algorithm called Ranking that achieved a ratio of $1 - 1/e$ for unweighted online matching with adversarial arrivals and showed that bound was tight. The vertex-weighted version of this problem was introduced by Aggarwal, Goel, Karande, and Mehta [13] who showed that a hybrid of Ranking and the greedy algorithm yielded a tight ratio of $1 - 1/e$.

The unweighted problem with stochastic rewards was introduced by Mehta and Panigrahi [14]. They stated that the simple greedy algorithm achieves 0.5. For the special case where all edge probabilities p_e are equal and vanishingly small,

Adversarial	Unweighted	Vertex-weighted	Edge-weighted
Non-stochastic	0.632 [12] (tight)	0.632 [13] (tight)	–
Stochastic rewards	0.5 [14] (0.62 [14] → **?**)	? → **0.5** [15]	–
Patience	? → **0.5** [15]	? → **0.5** [15]	–

Random order	Unweighted	Vertex-weighted	Edge-weighted
Non-stochastic	0.696 [16]	0.6534 [17]	$1/e$ [18]
Stochastic rewards	0.5 [14]	? → **0.5** [15]	?
Patience	? → **0.5** [15]	? → **0.5** [15]	?

Known IID	Unweighted	Vertex-weighted	Edge-weighted
Non-stochastic	→ **0.7299** [19]	→ **0.7299** [19]	→ **0.705** [20]
Stochastic rewards	0.5 [14] → **0.623** [19]	? → **0.623** [19]	? → **0.623** [19]
Patience	0.46 [21] → **0.5** [15]	0.46 [21] → **0.5** [15]	? → **0.46** [21]

Figure 3.1: Landscape of online matching results with upper bounds for some problems in parenthesis. The bolded results with arrows show contributions of our work. Question marks denote problems where no prior bound was known. In the case of the hardness result for unweighted matching with stochastic rewards and adversarial arrivals, we argue in Section 3.4.4 that the definition of competitive ratio under which that hardness result was proven is too pessimistic. If a result follows immediately from the work of a paper, we cite that paper even if the specific result was not mentioned.

they showed that the Balance algorithm achieves 0.567. They further show that using their definition of optimal, no algorithm can achieve a ratio better than 0.621 (strictly less than $1-1/e$) for this problem. However, we argue that this result arises from a different definition of competitive ratio which is too pessimistic. Therefore, we claim this hardness result does not hold under the common definition of competitive ratio for this problem. Later, Mehta, Waggoner, and Zadimoghaddam [22] studied the problem where p_e can be unequal, but are still vanishingly small and showed a 0.534 ratio. However, 0.5 remains best known for the case of arbitrary, unequal probabilities.

Random Order Arrival. For unweighted random order arrival, Mahdian and Yan [16] showed that Ranking achieves 0.696. Just recently, Huang et al [17] showed the first algorithm to beat $1-1/e$ for the vertex-weighted problem with random arrivals, achieving 0.6534. Other work addressing random order arrival includes [18, 23, 24]. There is also a hardness of 5/6 due to Goel and Mehta [25].

Known IID Arrival. The vertex-weighted and unweighted settings have many results starting with Feldman, Mehta, Mirrokni and Muthukrishnan [26] who first beat $1-1/e$ with a competitive ratio of 0.67 for the unweighted problem. This was improved by Manshadi, Gharan, and Saberi [27] to 0.705 with an adaptive algorithm. They also showed that even in the unweighted variant with integral arrival rates, no algorithm can achieve a ratio better than $1-e^{-2} \approx 0.86$. Finally, Jaillet and Lu [28] presented an adaptive algorithm which used a clever LP to achieve 0.725 and $1-2e^{-2} \approx 0.729$ for the vertex-weighted and unweighted problems, respectively. This was improved to 0.7299 for both problems in our work [20].

For edge weights, Haeupler, Mirrokni, Zadimoghaddam [29] were the first to beat $1 - 1/e$ by achieving a competitive ratio of 0.667. They use a *discounted LP* with tighter constraints than the basic matching LP and they employ the *power of two choices* by constructing two matchings offline to guide their online algorithm. This was improved to 0.705 in our prior work [20].

For edge weights and stochastic rewards, we presented an algorithm achieving $1 - 1/e$ which is tight for any algorithm based on the natural LP used on this problem [20]. Bansal *et al.* [30] introduced the problem of online stochastic matching with timeouts (patience) and gave the first constant factor competitive ratio of 0.12. This was later improved to 0.24 by Adamczyk *et al.* [31]. Most recently, we showed 0.46 [21]. As stated above, the original motivation for patience came from the patience constraints in the offline stochastic matching problem. This offline problem was first introduced by Chen *et al.* [32] and later studied by Bansal *et al.* [30], Adamczyk *et al.* [31], and Baveja *et al.* [33]. A generalization to packing problems was studied by Gupta and Nagarajan [34].

Other Related Work. Beyond the arrival models described above, online matching is studied under other variants such as unknown distributions and known adversarial distributions. With unknown distributions, an item is sampled in each round from a fixed, but unknown distribution. If the sampling distributions are required to be the same during each round, it is called unknown I.I.D. [35, 36]; otherwise, it is called adversarial stochastic input [35]. As for known adversarial distributions, in each round an item is sampled from a known distribution, which is allowed to change over time [37, 38]. The edge-weighted setting has been studied in

the adversarial model by Feldman, Korula, Mirrokni and Muthukrishnan [39], where they consider an additional relaxation of "free-disposal".

Devanur *et al* [36] gave an algorithm which achieves a ratio of $1 - k!/(k^k e^k)$ for the Adwords problem in the Unknown I.I.D. arrival model with knowledge of the optimal budget utilization and when the bid-to-budget ratios are at most $1/k$, where k is some positive integer. Alaei *et al.* [37] considered the Prophet-Inequality Matching problem, in which v arrives from a distinct (known) distribution $\mathcal{D}_t$, in each round t. They gave a $1 - 1/\sqrt{k+3}$ competitive algorithm, where k is the minimum capacity of u. In [21] we introduced the online stochastic matching with two-sided timeouts problem where the offline vertices have patience constraints as well and showed a ratio of 0.3.

The b-matching Variant. One may further consider the case where we may allow offline vertices to be matched multiple times. This captures the notion that we allow ads to be presented to multiple users (in the non-stochastic case) or clicked by multiple users (in the stochastic case). Similarly, in e-commerce applications, this corresponds to having multiple items of the same type which can be sold to multiple users. This is captured by the notion of *b-matching*, as studied in [40]. In this generalization, each offline vertex $u \in U$ has a *capacity*, b_u, and we allow u to be matched up to b_u times. Standard online matching can be seen as a special case of this problem where $b_u = 1$ for all $u \in U$. Note that we can extend results for the classical matching problem to that of b-matching by making b_u copies of each offline vertex u. Note that the capacities, which restrict the number of times a vertex may be *successfully* matched, are different from patience constraints, which restrict

the number of *attempts* each vertex has to be matched (that is, patience constraints count the number of failed attempts at a match, while capacities only care about the number of successful matches). The online b-matching problem was first introduced in [40], which considered the *unweighted, non-stochastic* setting in the adversarial model, and presented an optimal $1-\frac{1}{(1+1/b)^b}$-competitive algorithm for the case where all offline vertices have capacity at least b (note that for large b, this approaches $1-1/e$). For the Known IID arrival model with *stochastic rewards* and *edge weights*, Brubach *et al.* [20] showed that the competitive ratio is at least $1-b^{-1/2+\epsilon}-O(e^{-b^{2\epsilon}/3})$ for any $\epsilon > 0$ (note that, contrary to the adversarial model, this ratio approaches 1 for large b). While our 0.5-competitive algorithm (Theorem 7) extends to the b-matching case (by duplicating vertices as described above), we leave as an open problem whether this can be improved; we note however that the result of [40] provides an upper bound of $1 - 1/e$ for large b.

Further Generalizations. In [41], Meir et al. consider a deterministic model in which the online vertices are rational agents who make matching choices: They will choose the offline vertex which maximizes their utility (defined as the difference between their preference valuation of the choice and the posted price of the choice). The problem is then to design a mechanism for setting the posted prices of each alternative so as to maximize the social welfare (the sum of the valuations of all the agents final choices). Our model differs significantly in that matching decisions are made by the algorithm rather than by agents, edge rewards are stochastic, and the goal is to maximize the expected total weight (profit) of the matching rather than the expected welfare. While [41] models problems such as a parking mechanism with

the goal of maximizing the benefit of all agents, our setting models problems such as e-commerce and online advertising. The *Prophet Inequality Matching* problem [37] may be viewed as a variant of *edge-weighted* b-matching with an arrival model similar to Known IID, but in which each stage of the online arrivals may have a *different* (though still independent) known distribution. Alaei et al. [37] also considered the *Budgeted* Prophet Inequality Matching problem, where offline vertices instead have budgets limiting the total amount of *weight* that may be allocated to them, rather than the *number* of vertices (note that in the special case of vertex weights, the budgeted version is equivalent to the version with capacities). We note that this variant does not consider stochastic rewards or patience constraints.

3.3 Online Matching with Known I.I.D Arrivals

We devote this section to the non-stochastic online matching problem with known I.I.D. arrivals. The focus will be on the main technical contributions and our algorithm for the edge-weighted problem. The full details of this work including algorithms for the vertex-weighted, unweighted, and stochastic rewards problems are presented in [19].

3.3.1 Preliminaries and Notation

In *edge-weighted online matching with known I.I.D. arrivals*, we are given a bipartite graph $G = (U, V, E)$. The offline set U is fixed from the start while the online set V represents the online vertex types we sample I.I.D. from. Each edge

$e \in E$ is associated with a weight w_e. This represents the *input graph*. The vertices v arrive online and are drawn with replacement from an I.I.D. distribution on V. Each arriving vertex must be matched or discard before the next arrives, and our goal is to construct a maximum weight matching. For each $v \in V$, we are given an *arrival rate* r_v, which is the expected number of times v will arrive. The results presented in this section focus on the integral arrival rates setting where all $r_v \in \mathbb{Z}^+$. For reasons described in [29], we can further assume without loss of generality that each v has $r_v = 1$ under the assumption of integral arrival rates. In particular, a vertex type v with an integral arrival rate $k > 1$, can be split into k different vertex types each with an arrival rate of 1. In this case, we have that $|V| = n$ where n is the total number of online rounds.

Asymptotic assumption and notation. We assume n is large and analyze algorithms as n goes to infinity: e.g., if $x \leq 1 - (1 - 2/n)^n$, we write this as "$x \leq 1 - 1/\mathsf{e}^2$" instead of the more-accurate "$x \leq 1 - 1/\mathsf{e}^2 + o(1)$". These suppressed $o(1)$ terms will subtract at most $o(1)$ from our competitive ratios. We use e for Euler's constant in contrast with e which denotes an edge and WS to refer to the worst case instance for an algorithm.

3.3.2 LP Benchmark

As in prior work (e.g., [11]), we use the following LP to upper bound the optimal offline expected performance and also use it to guide our algorithm. There is a variable f_e for each edge. Let $\partial(u)$ be the set of edges adjacent to a vertex

$u \in U$ and $f_u = \sum_{e \in \partial(u)} f_e$. Define $\partial(v)$ and f_v similarly for $v \in V$. Recall that w_e is the weight of an edge e. Constraint (3.4) is used in [27] and [29]. We add Constraint (3.5) to further tighten the LP upper bound on the optimal solution and avoid certain bad cases that arise in the unweighted and vertex-weighted problems, but this constraint is not needed for our edge-weighted algorithm.

$$\text{maximize} \quad \sum_{e \in E} f_e w_e \tag{3.1}$$

$$\text{subject to} \quad \sum_{e \in \partial(u)} f_e \leq 1 \qquad \forall u \in U \tag{3.2}$$

$$\sum_{e \in \partial(v)} f_e \leq 1 \qquad \forall v \in V \tag{3.3}$$

$$0 \leq f_e \leq 1 - 1/\mathsf{e} \qquad \forall e \in E \tag{3.4}$$

$$f_e + f_{e'} \leq 1 - 1/\mathsf{e}^2 \qquad \forall e, e' \in \partial(u), \forall u \in U \tag{3.5}$$

Lemma 1. *Let* OPT *denote the total weight obtained by the best offline algorithm. Let* $\mathbf{f}^*$ *denote the optimal solution to the above* LP. *Then* $\sum_{e \in E} f_e^* w_e \geq \mathbb{E}[\text{OPT}]$.

Proof. Let Y_e denote the indicator random variable for the event that edge $e \in E$ is matched in the optimal solution for a given arrival sequence $\mathcal{A}$. Let $y_e := \mathbb{E}_{\mathcal{A}}[Y_e]$ for every edge $e \in E$. We will now argue that the vector $\vec{y} := (y_e)_{e \in E}$ is a feasible solution to the LP.

Consider a vertex $u \in U$. We have that $\sum_{e \in \partial(u)} Y_e \leq 1$. Taking expectations on both sides and using the linearity of expectation we have $\sum_{e \in \partial(u)} y_e \leq 1$. This shows that $\vec{y}$ is feasible for constraint (3.2). Let R_v denote the random variable

for the number of times a vertex $v \in V$ arrived in a given arrival sequence $\mathcal{A}$. Then we have, for every $v \in V$, $\sum_{e \in \partial(v)} Y_e \leq R_v$. From the integral arrival rates assumption, $\mathbb{E}_{\mathcal{A}}[R_v] = 1$ for every $v \in V$. Thus, from linearity of expectation we obtain $\sum_{e \in \partial(v)} y_e \leq 1$. This shows that $\vec{y}$ is feasible for constraint (3.3).

For any edge $e = (u, v)$, let $\mathbb{I}[R_v > 0]$ be an indicator for the event that a vertex $v \in V$ arrives at least once in the T rounds. Thus, for any arrival sequence $\mathcal{A}$, we have $Y_e \leq \mathbb{I}[R_v > 0]$. Taking expectations on both sides we get $y_e \leq \mathbb{E}_{\mathcal{A}}[\mathbb{I}[R_v > 0]$. The probability that a vertex v never arrives in T rounds is $\left(1 - \frac{1}{T}\right)^T \leq 1/\mathsf{e}$. Thus, $\mathbb{E}_{\mathcal{A}}[\mathbb{I}[R_v > 0] \leq 1 - 1/\mathsf{e}$. This shows that $\vec{y}$ is feasible for constraint (3.4).

Now, consider two edges $e, e' \in \partial(u)$ for some $u \in U$. Let $e = (u, v)$ and $e' = (u, v')$ and as before let $\mathbb{I}[R_v > 0]$ and $\mathbb{I}[R_{v'} > 0]$ denote the indicators for the events that v, v' arrive at least once in the T rounds, respectively. For any arrival sequence $\mathcal{A}$, we have that $Y_e + Y_{e'} \leq \mathbb{I}[R_v > 0] \wedge \mathbb{I}[R_{v'} > 0]$. Taking expectations on both sides, we get $y_e + y_{e'} \leq \mathbb{E}_{\mathcal{A}}[\mathbb{I}[R_v > 0] \wedge \mathbb{I}[R_{v'} > 0]]$. The probability that both v and v' never arrive in the T rounds is then given by $\left(1 - \frac{2}{T}\right)^T \leq \frac{1}{\mathsf{e}^2}$. Thus, we get $y_e + y_{e'} \leq 1 - \frac{1}{\mathsf{e}^2}$, which shows that $\vec{y}$ is feasible for constraint 3.5.

The expected weight of the optimal solution is $\mathbb{E}_{\mathcal{A}}[\sum_{e \in E} w_e Y_e]$ which from linearity of expectation gives $\sum_{e \in E} w_e y_e$. Since $\vec{y}$ is a feasible solution we have that the optimal value to LP is at least as large as the expected offline optimal solution. □

Given Lemma 1, we can compare the performance of our algorithm to this LP. Suppose that $\vec{f}^*$ is the optimal solution to the above LP. We prove the following

lemma which shows that it suffices to analyze the competitive ratio *edge-wise*.

Lemma 2. *If* $\min_{e \in E, f_e^* > 0} \frac{\Pr[e \text{ is included in the matching}]}{f_e^*} \geq \alpha$, *then this implies that the competitive ratio is at least* α.

Proof. From linearity of expectation we have that

$$\begin{aligned}
\mathbb{E}[\text{ALG}] &= \sum_{e \in E} \Pr[e \text{ is included in the matching}] \\
&\geq \alpha \sum_{e \in E} f_e^* \\
&\geq \alpha \mathbb{E}[\text{OPT}].
\end{aligned}$$

□

In what follows, we only compute a lower-bound on the probability that any edge $e \in E$ is included in the final matching (we call this quantity the *competitive ratio of edge* e) which implies a lower-bound on the overall competitive ratio.

We note that the work of [27] does not use an LP to upper-bound the optimal value of the offline instance. Instead, they use Monte-Carlo simulations wherein they simulate the arrival sequence and compute the vector $\vec{f}$ by approximating (via Monte-Carlo simulation) the probability of matching an edge e in the offline optimal solution. We do not use a similar approach for our problems for three important reasons. (1) For the *weighted* variants, namely the edge and vertex-weighted versions, the number of samples depends on the maximum value of the weight, making it expensive. (2) In the unweighted version, the running time of the sampling based algorithm is $O(|E|^2 n^4)$; on the other hand, we show in Section 3.3.5

that the LP based algorithm can be solved much faster, $\tilde{O}(|E|^2)$ time in the worst case and even faster than that in practice. (3) For the stochastic rewards setting, the offline problem is not known to be polynomial-time solvable, which is required for [27] since they rely on solving instances of the offline problem on simulated graphs. [42] show that under the assumption of constant p *and* OPT $= \omega(1/p)$, we can obtain a $(1-\epsilon)$-approximation to the optimal solution. However, these assumptions are too strong to be used in our setting.

3.3.3 Overview of Edge-weighted Algorithm and Contributions

The previous best result due to [29] for the edge-weighted problem was 0.667. They used two matchings, M_1 and M_2, from the offline graph to guide the online algorithm and leverage the *power of two choices*. When a vertex v arrives for the first time, it can be matched to its neighbor in M_1 and on its second arrival it can be matched to its neighbor in M_2. However, these two matchings may not be edge disjoint, leaving some arriving vertices with only one choice (meaning a second arrival of the same vertex type is guaranteed to go unmatched). In fact, choosing two guiding matchings that maximize both the edge weights and the number of disjoint edges is a major challenge that arises in applying the *power of two choices* to this setting.

When the same edge (u, v) is included in both matchings M_1 and M_2, the copy of (u, v) in M_2 can offer no benefit and a second arrival of v is wasted. To use an example from related work, Haeupler *et al.* [29] choose two matchings in

the following way. M_1 is attained by solving an LP with constraints (3.2), (3.3), and (3.4) and randomly rounding to an integral solution. M_2 is constructed by finding a maximum-weight matching and removing any edges which have already been included in M_1. A key element of their proof is showing that the probability of an edge being removed from M_2 is at most $1 - 1/\mathsf{e} \approx 0.63$.

Our approach is to construct two or three matchings together in a correlated manner to reduce the probability that some edge is included in multiple matchings. We show a general technique to construct an ordered set of k matchings where k is an easily adjustable parameter. For $k = 2$, we show that the probability of an edge appearing in both M_1 and M_2 is at most $1 - 2/\mathsf{e} \approx 0.26$.

For the algorithms presented, we first solve an LP on the input graph. We then round the LP solution vector to a sparse integral vector and use this vector to construct a randomly ordered set of matchings which will guide our algorithm during the online phase. We start in Section 3.3.7 with a simple warm-up algorithm which uses a set of two matchings as a guide to achieve a 0.688 competitive ratio, improving the best known result for this problem. We follow it up in Section 3.3.9 with a slight variation that improves the ratio to 0.7. We refer the reader to our paper [19] for a more complex 0.705-competitive algorithm which relies on a convex combination of a 3-matching algorithm and a separate *pseudo-matching* algorithm.

3.3.4 Overview of Vertex-weighted Algorithm and Contributions

Here, we briefly summarize our work on the vertex-weighted problem, but refer to our paper [19] for the full algorithm and analysis which involves extensive cases. The previous best results for this problem due to [28] for the vertex-weighted and unweighted problems were 0.725 and $1 - 2\mathsf{e}^{-2} \approx 0.729$, respectively. They used a clever LP which guaranteed they could find a solution wherein each edge variable was assigned a value in $\{0, 1/3, 2/3\}$ as opposed to an arbitrary fractional value. This property, which we call a $\{0, 1/3, 2/3\}$ solution, was required by their adaptive online algorithm. However, their special LP was a slightly weaker upper bound on the optimal solution than the LP we describe in Section 3.3.2.

Another key challenge encountered by [28] was that solutions to their special LP could lead to length-four cycles of type C_1 shown in Figure 3.2. In fact, they used this case to show that no algorithm could perform better than $1 - 2\mathsf{e}^{-2} \approx 0.7293$ using their LP as an upper bound. They mentioned that tighter LP constraints such as (3.4) and (3.5) in the LP from Section 3.3.2 could avoid this bottleneck, but did not propose a technique to use them. Note that the $\{0, 1/3, 2/3\}$ solution produced by their specific LP was an essential component of their *Random List* algorithm.

To address this challenge, we show a randomized rounding algorithm to construct a similar, simplified $\{0, 1/3, 2/3\}$ vector from the solution to a stronger benchmark LP. This allows for the inclusion of additional constraints, most importantly constraint (3.5). Using our rounding algorithm combined with tighter constraints, we can upper-bound the probability of a vertex appearing in the cycle type C_1 from

Figure 3.2 at $2 - 3/\mathsf{e} \approx 0.89$. By constant, cycles of type C_1 occur deterministically in [28].

Additionally, we note briefly that there are other length four cycles with different variable weights, defined as types C_2 and C_3 (See Figure 3.2). These cycles are also problematic, but we show how to deterministically break them without creating any new cycles of type C_1 (This can happen if the cycle breaking is not done carefully). Finally, we describe an algorithm which utilizes these techniques to improve previous results in both the vertex-weighted and unweighted settings.

For this problem, we first solve the LP in Section 3.3.2 on the input graph and use the technique in Section 3.3.6 to obtain a sparse fractional vector. We then utilize a randomized online algorithm (similar to the one in [28]), which uses the sparse fractional vector as a guide, to achieve a competitive ratio of 0.7299.

Previously, there was a gap between the best unweighted algorithm with a ratio of $1 - 2\mathsf{e}^{-2}$ due to [28] and the negative result of $1 - \mathsf{e}^{-2}$ due to [27]. We take a step toward closing this gap by showing that an algorithm can achieve $0.7299 > 1 - 2\mathsf{e}^{-2}$ for both the unweighted and vertex-weighted variants with integral arrival rates. In doing so, we make progess on Open Questions 3 and 4 from the book [11]. [1]

3.3.5 Running Time of the Algorithms

In this section, we discuss the implementation details of our algorithms. All of our algorithms solve an LP in the pre-processing step. The dimension of the LP

[1] Open Questions 3 and 4 state the following: "In general, close the gap between the upper and lower bounds. In some sense, the ratio of $1 - 2\mathsf{e}^{-2}$ achieved in [28] for the integral case, is a nice 'round' number, and one may suspect that it is the correct answer."

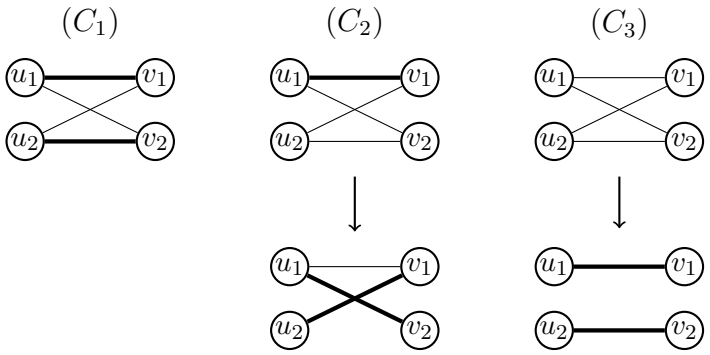


Figure 3.2: Challenge for the vertex-weighted problem. These are the three possible types of cycles of length 4 after applying our rounding approach $\mathsf{DR}[\mathbf{f}, 3]$ to the LP solution for the vertex-weighted problem. Thin edges have $f_e = 1/3$ and thick edges have $f_e = 2/3$. Cycle type C_1 is the source of the negative result described by Jaillet and Lu [28]. It results from the edge variable assignments in their special LP. This structure and variable assignment leads to a gap of $1 - 2\mathsf{e}^{-2}$ between the LP solution and the best possible solution of any online algorithm. The arrows show how cycle types C_2 and C_3 are broken by our algorithm while type C_1 is just avoided with bounded probability.

is determined by the constraint matrix which consists of $O(|E|^2 + |U| + |V|)$ rows and $O(|E|)$ columns. However, note that the number of non-zero entries in this matrix is of the order $O(|E|^2)$ because each edge is subject to $O(|E|)$ constraints primarily due to LP constraint 3.5. Some recent work (e.g., [43]) shows that such sparse programs can be solved in time $\tilde{O}(|E|^2)$ using interior point methods (which are known to perform well in practice). This sparsity in the LP is the reason we can solve large instances of the problem. The second critical step in pre-processing is to perform randomized rounding. Note that we have $O(|E|)$ variables and that in each step of the randomized rounding due to [4], they incur a running time of $O(|E|)$. Hence the total running time to obtain a rounded solution is of the order $O(|E|^2)$. Additionally, both of these operations are part of the pre-processing step. In the online phase, the edge-weighted algorithm presented here incurs a per-time-step running time of $O(1)$. Other algorithms from [19] have online steps that require

at most $O(|U|)$ for the stochastic rewards case (in fact, a smarter implementation using binary search runs as fast as $O(\log |U|)$ time) and $O(1)$ for the vertex-weighted algorithm.

3.3.6 LP Rounding Technique $\mathsf{DR}[\mathbf{f}, k]$

For the algorithms in this section, we first solve the benchmark LP in Section 3.3.2 for the input instance to get a fractional solution vector $\mathbf{f}$. We then round $\mathbf{f}$ to an integral solution $\mathbf{F}$ using a two step process we call $\mathsf{DR}[\mathbf{f}, k]$. The first step is to multiply $\mathbf{f}$ by k. The second step is to apply the dependent rounding techniques of Gandhi, Khuller, Parthasarathy, and Srinivasan [4] to this new vector. For this manuscript, we focus on $k = 2$.

While dependent rounding is typically applied to values between 0 and 1, the useful properties extend naturally to our case in which kf_e may be greater than 1 for some edge e. To understand this process, it is easiest to imagine splitting each kf_e into two edges with the integer value $f'_e = \lfloor kf_e \rfloor$ and fractional value $f''_e = kf_e - \lfloor kf_e \rfloor$. The former will remain unchanged by the dependent rounding since it is already an integer while the latter will be rounded to 1 with probability f''_e and 0 otherwise. Our final value F_e would be the sum of those two rounded values. The two properties of dependent rounding we use are:

1. **Marginal distribution:** For every edge e, let $p_e = kf_e - \lfloor kf_e \rfloor$. Then, $\Pr[F_e = \lceil kf_e \rceil] = p_e$ and $\Pr[F_e = \lfloor kf_e \rfloor] = 1 - p_e$.

2. **Degree-preservation:** For any vertex $w \in U \cup V$, let its fractional de-

gree kf_w be $\sum_{e \in \partial(w)} kf_e$ and integral degree be the random variable $F_w = \sum_{e \in \partial(w)} F_e$. Then $F_w \in \{\lfloor kf_w \rfloor, \lceil kf_w \rceil\}$.

These properties are guaranteed by directly applying the analysis of [4] to the decomposed solution vector as described above.

3.3.7 Warm-up: 0.688-competitive Algorithm

As a warm-up, we describe a simple algorithm which achieves a competitive ratio of 0.688 and introduces the key ideas in our approach. We begin by solving the LP in Section 3.3.2 to get a fractional solution vector $\mathbf{f}$ and applying $\mathsf{DR}[\mathbf{f}, 2]$ as described in Section 3.3.6 to get an integral vector $\mathbf{F}$. We construct a bipartite graph $G_{\mathbf{F}}$ with F_e copies of each edge e. Note that $G_{\mathbf{F}}$ will be a multigraph with max degree 2 since for all $w \in U \cup V$, $F_w \leq \lceil 2f_w \rceil \leq 2$. Thus, we can decompose it into two matchings using a greedy algorithm and *Hall's Theorem*. The exact choice of the two matchings is not critical to the algorithm as long as the union contains all edges in $G_{\mathbf{F}}$. Finally, we randomly permute the two matchings into an ordered pair of matchings, $[M_1, M_2]$. These matchings serve as a guide for the online phase of the algorithm, similar to [29]. The entire warm-up algorithm for the edge-weighted model, denoted by EW_0, is summarized in Algorithm 1.

3.3.8 Analysis of Warm Up Algorithm EW_0

We show that EW_0 (Algorithm 1) achieves a competitive ratio of 0.688. Let $[M_1, M_2]$ be our randomly ordered pair of matchings. Note that there might exist

Algorithm 1: [EW_0]

1. Construct and solve the benchmark LP in Section 3.3.2 for the input instance.
2. Let $\mathbf{f}$ be an optimal fractional solution vector. Call $\mathsf{DR}[\mathbf{f}, 2]$ to get a random integral vector $\mathbf{F}$.
3. Create the graph $G_{\mathbf{F}}$ with F_e copies of each edge $e \in E$ and decompose it into two matchings as described in text.
4. Randomly permute the matchings to get a *random ordered* pair of matchings, say $[M_1, M_2]$.
5. When a vertex v arrives for the first time, attempt to match v to u_1 if $(u_1, v) \in M_1$; when v arrives for the second time, attempt to match v to u_2 if $(u_2, v) \in M_2$.
6. When a vertex v arrives for the third time or more, do nothing in that step.

some edge e which appears in both matchings due to having $f_e > 1/2$, which could be rounded up to $F_e = 1$ in the rounding step. Therefore, we consider three types of edges. We say an edge e is of type ψ_1, denoted by $e \in \psi_1$, if and only if e appears *only* in M_1. Similarly $e \in \psi_2$, if and only if e appears *only* in M_2. Finally, $e \in \psi_b$, if and only if e appears in *both* M_1 and M_2. Let P_1, P_2, and P_b be the probabilities of getting matched for $e \in \psi_1$, $e \in \psi_2$, and $e \in \psi_b$, respectively. According to the result in Haeupler *et al.* [29], Lemma 3 bounds these probabilities.

Lemma 3 (Proof details in section 3 of [29])**.** *For any two matchings, M_1 and M_2, steps (5) and (6) in Algorithm 1 imply that we have (1) $P_1 > 0.5808$; (2) $P_2 > 0.14849$; and (3) $P_b > 0.6321$.*

We can use Lemma 3 to prove that the warm-up algorithm EW_0 achieves a ratio of 0.688 by examining the probability that a given edge becomes type ψ_1, ψ_2, or ψ_b after rounding the LP solution to two matchings and randomly ordering the matchings.

Analysis of EW_0. Consider the following two cases.

- **Case 1:** $0 \leq f_e \leq 1/2$: By the marginal distribution property of dependent rounding, there can be at most one copy of e in $G_{\mathbf{F}}$ and the probability of including e in $G_{\mathbf{F}}$ is $2f_e$. Since an edge in $G_{\mathbf{F}}$ can appear in either M_1 or M_2 with equal probability $1/2$, we have $\Pr[e \in \psi_1] = \Pr[e \in \psi_2] = f_e$. Thus, the ratio is $(f_e P_1 + f_e P_2)/f_e = P_1 + P_2 = 0.729$.

- **Case 2:** $1/2 < f_e \leq 1 - \frac{1}{\mathsf{e}}$: Similarly, by marginal distribution, $\Pr[e \in \psi_b] = \Pr[F_e = \lceil 2f_e \rceil] = 2f_e - \lfloor 2f_e \rfloor = 2f_e - 1$. It follows that $\Pr[e \in \psi_1] = \Pr[e \in \psi_2] = (1/2)(1-(2f_e-1)) = 1-f_e$. Thus, the ratio is (noting that the first term is from case 1 while the second term is from case 2) $((1-f_e)(P_1+P_2)+(2f_e-1)P_b)/f_e \geq 0.688$, where the worst case is for an edge e with $f_e = 1 - \frac{1}{\mathsf{e}}$. □

3.3.9 Improved Algorithm: 0.7-competitive Algorithm

In this section, we describe an improvement upon the previous warm-up algorithm to get a competitive ratio of 0.7. We start by making an observation about the performance of the warm-up algorithm. After solving the LP, let edges with $f_e > 1/2$ be called *large* and edges with $f_e \leq 1/2$ be called *small*. Let L and S, be the sets of large and small edges, respectively. Notice that in the previous analysis, small edges achieved a much higher competitive ratio of 0.729 versus 0.688 for large edges. This is primarily due to the fact that we may get two copies of a large edge in $G_{\mathbf{F}}$. In this case, the copy in M_1 has a better chance of being matched, since

there is no edge which can "block" it (i.e. an edge with the same offline neighbor that gets matched first), but the copy that is in M_2 has no chance of being matched.

To correct this imbalance, we make an additional modification to the f_e values *before* applying $\mathsf{DR}[\mathbf{f}, k]$. The rest of the algorithm is exactly the same. Let η be a parameter to be optimized in the analysis. For all large edges $\ell \in L$ such that $f^*_\ell > 1/2$, we set $\tilde{f}^*_\ell(\ell) = f^*_\ell + \eta$. For all small edges $s \in S$ which are adjacent to some large edge, let $\ell \in L$ be the largest edge adjacent to s such that $f^*_\ell > 1/2$. Note that it is possible for s to have two large neighbors, one at each endpoint, but we only care about the larger of the two. We set $\tilde{f}^*_s = f^*_s \left(\frac{1-\tilde{f}^*_\ell}{1-f^*_\ell}\right)$.

In other words, we increase the values of large edges while ensuring that for all vertices $w \in U \cup V$, $f_w \leq 1$ by reducing the values of neighboring small edges proportional to their original values. Note that it is not possible for two large edges to be adjacent since they must both have $f_e > 1/2$, but the sum of two adjacent edges can be at most 1. For all other small edges which are not adjacent to any large edges, we leave their values unchanged. We then apply $\mathsf{DR}[\mathbf{f}, 2]$ to this new vector, multiplying by 2 and applying dependent rounding as before.

Analysis

Let $\eta = 0.0142$ be our optimized parameter for the new algorithm.

Theorem 1. *For edge-weighted online stochastic matching with integral arrival rates,* $\mathsf{EW}(0.0142)$ *achieves a competitive ratio of at least* 0.7.

Proof. As in the warm-up analysis, we'll consider large and small edges separately

- **Scenario 1:** $0 \leq f_s^* \leq \frac{1}{2}$:

  Here we have two cases

  - **Case 1:** s is not adjacent to any large edges.

    In this case, the analysis is the same as Case 1 in the warm-up analysis. Thus, the probability that edge s is added to the matching is $0.729 f_e^*$.

  - **Case 2:** s is adjacent to some large edge ℓ.

    For this case, let f_ℓ^* be the value of the largest neighboring edge in the original LP solution. Then the probability that edge s is added to the matching is

    $$f_s^* \left(\frac{1 - (f_\ell^* + \eta)}{1 - f_\ell^*} \right) (0.1484 + 0.5803).$$

    This follows from Lemma 3; in particular, the first two terms are the result of how we set $\tilde{f}_s$ in the algorithm, while the two numbers, 0.1484 and 0.5803, are the probabilities that s is matched when it is in M_2 and M_1, respectively. Note that for $f_\ell^* \in [0, 1)$ this is a decreasing function in f_ℓ^*. So the worst case is when $f_\ell^* = 1 - \frac{1}{\mathsf{e}}$ (due to third constraint in the LP (3.4)) Thus, the probability that edge s is added to the matching is

    $$f_s^* \left(\frac{1 - (1 - \frac{1}{\mathsf{e}} + \eta)}{1 - (1 - \frac{1}{\mathsf{e}})} \right) (0.1484 + 0.5803).$$

    Since $\eta = 0.0142$, this evaluates to,

    $$0.701 f_s^*. \tag{3.6}$$

- $\frac{1}{2} < f_\ell^* \leq 1 - \frac{1}{\mathsf{e}}$: Here, the probability that ℓ is added to the matching is, $[1 - (f_\ell^*(\ell) + \eta)][P_1 + P_2] + [2(f_\ell^* + \eta) - 1]P_b$. This can re-arranged to obtain

$$(P_1 + P_2)(1 - \eta) + (2\eta - 1)P_b + f_\ell^*[2P_b - P_1 - P_2]. \tag{3.7}$$

Since $\eta = 0.0142$ using Lemma 3 we have $(P_1+P_2)(1-\eta)+(2\eta-1)P_b = 0.1048$. Similarly, using Lemma 3 we have $2P_b - P_1 - P_2 = 0.535$. Thus, Eq. (3.7) simplifies to,

$$0.1048 + f_\ell^* 0.535 \tag{3.8}$$

We can write Eq. (3.8) as $f_\ell^*[0.1048/f_\ell^* + 0.535]$. Note that $\frac{1}{2} < f_\ell^* \leq 1 - \frac{1}{\mathsf{e}}$. Thus, Eq. (3.8) can be lower-bounded by

$$0.701 f_\ell^*. \tag{3.9}$$

Thus combining Eq. (3.6) and (3.9) with Lemma 2 we get a competitive ratio of 0.7.

We now show that the chosen value of $\eta = 0.0142$ ensures that both $\tilde{f}_\ell^*$ and $\tilde{f}_s^*$ are less than 1 *after* modification. Since $f_\ell^* \leq 1 - \frac{1}{\mathsf{e}}$ we have that $f_\ell^* + \eta \leq 1 - \frac{1}{\mathsf{e}} + 0.0142 \leq 1$. Note that $f_\ell^* \geq 1/2$. Hence, the modified value $\tilde{f}_s^*$ is always less than or equal to the original value, since $\left(\frac{1-(f_\ell^*+\eta)}{1-f_\ell^*}\right)$ is decreasing in the range $f_\ell^* \in [1/2, 1 - \frac{1}{\mathsf{e}}]$ and has a value less than 0.98 at $f_\ell^* = 1/2$. □

3.3.10 The Integral Arrival Rates Assumption

As mentioned in the preliminaries, we make the simplifying assumption that the arrival rates $r_v = 1$ for every online vertex $v \in V$. Our algorithms and analysis crucially rely on this assumption. Specifically, our algorithm finds two matchings in the offline graph and uses them to guide the online matching process. In doing so, it assumes that each edge in those matchings is incident to an online vertex with an arrival rate of 1. Without this assumption, two key problems arise. First, Lemma 3, which bounds the probability that each edge gets matched, is no longer true as all of the analysis in the proof relies critically on the integral arrival rates assumption. When arrival rates are arbitrary, Lemma 3 does not hold. Consider an edge $e = (u, v)$ either in M_1 or M_2 with $r_v = 1/n$ for example, where n is the total number of online rounds. We observe that e will be matched with a probability no larger than the probability that v arrives at least once, which is $1-(1-1/n^2)^n \sim 1/n$.

Second, the algorithm described above can have arbitrarily bad performance when the arrival rates are less than 1. This algorithm will find two matchings in the offline graph and only attempt to match edges in those matchings. However, note that when a vertex has a small arrival rate (e.g. $\frac{1}{n}$), it is unlikely to arrive at all during the online process. It is possible to construct examples where the edges added to our two matchings after our rounding procedure will be incident on online vertices that are unlikely to arrive. Thus, our online algorithm would match almost no edges while the optimal offline algorithm could find a large value matching among the vertices that actually arrived.

3.4 Stochastic Rewards and Stochastic Matching with Patience

A common scenario in e-commerce is the online sale of unique goods due to the ability to reach niche markets via the internet (e.g., eBay, etc.). Typical products include rare books, trading cards, art, crafts, and memorabilia. We will use this as a motivating example in describing our setting. However, our problem can also model job search/hiring, assigning workers to tasks, online advertising, and other online matching problems.

In e-commerce, this may model the probability that a customer will purchase a given item. In online advertising, this corresponds to the *pay-per-click* model, in which ad revenue is only earned when a user clicks on an ad (the probabilities may be inferred or estimated based on historical data of the user). See [11] for further discussion and models.

In both the e-commerce and advertising settings, we only discover if a customer or user would purchase an item or click an ad *after* it has been presented to them and they have done so (or not): that is, we cannot later choose to "revoke" the item offer or ad placement. This situation exemplifies the *probe-commit* model: if a stochastic edge is probed and found to exist, it must be matched irrevocably. In the most basic *stochastic rewards* setting, we are allowed to probe at most one edge adjacent to each arriving vertex while offline vertices may have many edges probed until they are matched and become unavailable [14, 20]. Think of a single banner ad on a website for example. However, in this work we consider a further generalization called *patience constraints* (also known as timeouts in the literature) where an online

vertex v has a known patience t_v and we may probe up to t_v neighbors (stopping early if it is successfully matched) [30, 31, 44]. This corresponds to a user browsing multiple items until they either find something to buy or lose patience and exit the marketplace. Alternatively, this may correspond to the number of ad *impressions* a user may be shown while browsing a website or mobile app, and thus, the number of opportunities to show an ad that the user ultimately clicks. In our model, we make the standard assumption that offline vertices have unlimited patience.

Note that although this problem is "patience-constrained", it is actually more general than the classical online matching problem or the stochastic rewards variant [14], since the latter two essentially have patience values of 1 *for online vertices, while patience can be arbitrary in the "patience-constrained" problem.*

3.4.1 Additional Detail on Related Work in this Setting

In this work, we consider the setting with *vertex weights*, *stochastic edges* in the *probe-commit* model, and *patience constraints*. In what follows we review some related works which are more closely tied to this setting.

The work of [10] considers a model in which online vertices represent customers and offline vertices represent products, and a merchant wishes to offer products to consumers so as to maximize profit. This setting differs from our own in that the merchant offers a collection of several products all at once. The customer then either chooses to purchase some product (and in fact, may purchase multiple products at once), based on products offered to her, or chooses to purchase nothing. By contrast,

in our model the algorithm (the "merchant" in our setting) attempts one match at a time, stopping when a successful match occurs or the number of unsuccessful attempts equals the patience constraint.

In the setting of [10], each customer v has a "general choice model" $\phi_v(S, u)$ that specifies the probability that customer v purchases item u when offered the set S of items. More generally, since the model considers that v may purchase more that one item, $\phi_v(S, S')$ is used to denote the probability that v will purchase exactly the items S' when offered S (and then $\phi_v(S, u)$ is defined to be $\sum_{S':u\in S'} \phi_v(S, S')$). It is assumed that the customer will only purchase products that were offered to her as part of the assortment S (that is, $\phi_v(S, S') = 0$ if $S' \not\subseteq S$).

The algorithm they propose for their model can be viewed as a greedy algorithm which presents an online-arriving customer v with the set S that maximizes the expected profit of the items v purchases. Doing so would guarantee a competitive ratio of at least 0.5, though this maximization step is not necessarily solvable in polynomial time for arbitrary choice models (they present only a specific family of choice models for which this step can be solved in polynomial time).

Their results do not immediately extend to our setting, as their stochastic model is somewhat different. Extending their results to our setting requires a reduction from our sequential probing with the probe-commit model to this *all-at-once* model by construction of appropriate choice models ϕ. Further, such a reduction would not necessarily yield a polynomial-time result without also designing an algorithm for solving the aforementioned maximization in polynomial time.

One contribution of the present work is Algorithm 2, which indeed can be

viewed as greedily maximizing the expected weight (or profit) of v's match (or purchase). However, without also constructing a reduction from our sequential probing model to this all-at-once model, the result of [10] does not extend to give a competitive ratio of 0.5 for our problem. Rather, in the present work, we present clean, self-contained analyses of Algorithm 2 and Algorithm 3 to achieve a competitive ratio of 0.5 for our problem without relying on the results of [10] and without the need for a messy or complicated reduction.

Another work closely related to our model is that of [45], which considers a model very similar to ours, with stochastic rewards and vertex weights. They do not consider arbitrary patience constraints (i.e, they consider only the special case where $t_v = 1$ for every online vertex v). They present a $(1 - 1/e)$-competitive algorithm for the special case of *decomposable probabilities*: that is, the case where $p_{u,v} = p_u p_v$ for every edge u, v. They further show their algorithm is $(1 - 1/e)$-competitive for the case of *vanishing probabilities*, where $p_{u,v} \to 0$ for all u, v. They do not consider the more general setting of patience constraints.

The recent work of [46] studies an offline version of the problem, wherein all of the vertices are offline, but the edges are stochastic. For the *probe-commit* model, they achieve a competitive ratio of $1 - 1/e$. They also consider a variant called the *price of information* model (as opposed to probe-commit), in which each edge e has a price π_e and the goal is to output a matching M which maximizes $\sum_{e \in M} w_e - \sum_{e \in Q} \pi_e$, where Q is the set of all probed edges and $W(M)$ is the total /weight of the matching produced. Their techniques, like ours, utilize a non-standard LP to upper bound the weight of an optimal matching.

3.4.2 Our Contributions

Here, we give a rough outline of this section and our contributions to online matching problems with stochastic edges. Most of the work in this section focuses on a deeper understanding of the stochastic matching problem and presents an algorithm for adversarial arrivals, all of which appears in [15]. However, we also briefly mention some results for online matching with stochastic rewards [19] and online stochastic matching with patience [21], both in the known I.I.D arrival model.

Clarified and Unified Competitive Ratio Definition

Our first contribution in Section 3.4.4 is to argue for a unified definition of competitive ratio for online matching problems with stochastic rewards. We give the following definition which aligns with the prior work of [20,30,31,44], but differs crucially from [14].

Definition 2 (Competitive Ratio for Online Matching with Stochastic Rewards)**.** *This competitive ratio is defined as the ratio of an online algorithm's solution to the solution of an optimal algorithm for the corresponding offline* stochastic *matching problem.*

An important consequence of this definition, stated in Observation 3, is that the hardness result shown in [14] does not apply under Definition 2. The definition of competitive ratio in [14] compares the online algorithm to the solution of the Budgeted Allocation LP (equivalent to the LP in Section 3.4.5 with all $t_v = 1$) rather

than to the offline stochastic matching problem. It is known that the Budgeted Allocation LP upper bounds the offline stochastic matching problem. Thus, the positive results of [14] are unaffected by Definition 2 since their LP formulation still serves to *upper bound* the optimal offline stochastic matching solution under Definition 2.

Observation 3. *The hardness result of [14], upper bounding the competitive ratio of online matching with stochastic rewards under adversarial arrivals at* $0.621 < 1-1/e$*, does not apply under Definition 2 of the competitive ratio for this problem.*

We further show in Section 3.4.4 that definition 2 allows for a more granular comparison between online algorithms. The significance of Observation 3 is that it reopens the question of whether a tight $1 - 1/e$ bound on the competitive ratio can be achieved in the stochastic rewards with adversarial arrivals setting.

LP Stochasticity Gap

In the process of discussing the competitive ratio, we show that the standard LP formulation for the stochastic matching problem with patience (timeout) constraints [30,31,44] is a fairly weak upper bound on the optimal solution. We call this a *stochasticity gap* (defined formally in Section 3.4.6) of the LP relaxation, analogous to the familiar concept of an integrality gap. Theorem 4 states that the natural LP (LP 3.10 in Section 3.4.5) for the offline stochastic matching problem with patience (timeout) constraints has a stochasticity gap of at most 0.544. Similarly, we have shown in [44] that the standard LP for the more specific problem of online matching

with stochastic rewards (equivalent to the Budgeted Allocation LP) has a $1 - 1/e$ stochasticity gap.

Theorem 4. *There exists an instance of the offline bipartite stochastic matching with patience problem where LP 3.10 (Section 3.4.5) has a stochasticity gap of at most* 0.544.

This implies that the 0.46 competitive ratio we achieved in [44] for the online stochastic matching problem with patience (timeout) constraints and known IID arrivals is somewhat tight with respect to the LP used in that paper to upper bound the optimal solution and guide the online algorithm. In other words, that *online* algorithm achieves 0.46 compared to the LP solution while no *offline* algorithm can perform better than 0.544 with respect to same the LP solution. Thus, serious improvements to that problem will only be possible with a tighter upper bound and only measurable under Definition 2 as we will see in Section 3.4.4.

Optimal Offline Stochastic Matching on Star Graphs

In Section 3.4.8, we introduce a dynamic programming algorithm that solves offline stochastic matching on star graphs optimally (the best-known approximations for offline stochastic matching with patience constraints in bipartite graphs and general graphs are 0.35 [31] and 0.31 [33], respectively). This algorithm will be used as a subroutine in our online algorithm, where we will view an arriving online vertex as the center of a star graph, and its optimality — stated as Theorem 5 — is used in our analysis.

Theorem 5. *There exists an algorithm for the offline stochastic matching with patience problem on vertex-weighted star graphs that finds the optimal probing strategy in $O(n^2 t_v)$ time where n is the number of vertices and t_v is the patience of the center vertex.*

Note that $O(n^2 t_v)$ is at most $O(n^3)$ since the patience of a vertex t_v is at most n, the number of vertices in the entire graph.

Greedy Algorithms for Vertex-weighted Online Matching with Stochastic Rewards and Patience Constraints

In Section 3.4.9, we start by showing that an obvious, naive greedy algorithm for this problem can be arbitrarily bad as stated in Theorem 6.

Theorem 6. *Probing neighbors $u \in U$ of an arriving vertex v in non-ascending order of expected weight ($w_u p_{u,v}$) leads to a worst case competitive ratio of $O(1/n)$ where n is the number of offline vertices in the underlying graph.*

We then demonstrate how to achieve a 0.5 competitive ratio (best possible for a deterministic algorithm under adversarial arrivals) by locally optimizing for each arriving vertex and its neighborhood using the dynamic programing algorithm for star graphs from Section 3.4.8. We formalize this in Theorem 7.

Theorem 7. *There exists an algorithm which achieves a 0.5 competitive ratio for the vertex-weighted online matching problem with stochastic rewards and patience in the adversarial arrival model.*

Since the model in Theorem 7 is quite general and adversarial, the result extends to the unweighted problem as well as random order and known IID arrival models as stated in Corollary 1.

Corollary 1. *There exists an algorithm which achieves a* 0.5 *competitive ratio for the vertex-weighted and unweighted online stochastic matching with patience problems in the adversarial, random order, and known IID arrival models.*

We note further that the performance of our algorithm is tight with respect to greedy algorithms which optimize the performance of each arriving vertex locally instead of attempting to make globally optimal decisions across all arrivals.

While greedy matching algorithms generally have poorer worst case performance theoretically, Figure 4.1 illustrates that our result is currently the best known for a number of natural online matching problems. We also stress that greedy algorithms can be useful in practice and it is important to establish the difference between our greedy algorithm and the naive greedy approach, which one might be tempted to implement. The recent empirical work of [5] for non-stochastic online matching under known IID arrival gives evidence that simple greedy algorithms perform well on this problem in practice. They further observe that the theoretically superior algorithms of [20, 26–28, 47] can be augmented with greedy choices to add additional adaptivity which improves empirical performance. Indeed, many of their best results come from these greedy-augmented algorithms. Thus, greedy algorithms can play an important role in practical solutions to this problem and it is useful to understand their behavior.

Finally, since we are not subject to the hardness result of [14] under Definition 2, one might ask if the problem actually gets *easier* when restricted to vanishingly small edge probabilities (or even just probabilities strictly less than 1). For example, the algorithms proposed for the stochastic rewards problem with adversarial arrivals in [14] and [22] achieve their best results for vanishing probabilities. Further, standard adversarial inputs where a matched edge "blocks" a future potential match do not present the same bounds when the "blocking" edge has a low expected value before being realized. In one step toward addressing this question, we show in Theorem 8 that the hardness result of 1/2 for greedy algorithms extends to the special case of stochastic rewards with small edge probabilities. We use "SimpleGreedy" to refer to the algorithm which always probes an arbitrary available neighbor if one exists. We note that [14] showed that "SimpleGreedy" achieves a ratio of at least 1/2 in the stochastic rewards with adversarial arrivals setting.

Theorem 8. *There exists a family of unweighted graphs under stochastic rewards (online vertices with patience of 1) and adversarial arrivals for which* SimpleGreedy *achieves a competitive ratio of at most* $1/2$ *even when all edges have uniform probability* $p = O(1/n)$.

3.4.3 Preliminaries and Notation

We use $G = (U, V, E)$ to denote the bipartite graph with vertex set $U \cup V$ and edge set $E \subseteq U \times V$. For a given bipartite graph $G = (U, V, E)$, let $U = \{u_1, \ldots, u_m\}$ represent offline vertices and $V = \{v_1, \ldots, v_n\}$ represent online vertices. Let w_i

denote the *weight* of offline vertex $u_i \in U$. We assume, wlog, that $w_1 \geq w_2 \geq \ldots w_m$. For each edge $(u_i, v_j) \in U \times V$, let $p_{i,j}$ denote the given probability that edge (u_i, v_j) exists when probed. We also use $p_{u,v}$ for the given probability of edge (u, v) when indices i and j are not required. For simplicity, we may assume G is the *complete* bipartite graph with $E = U \times V$ by allowing $p_{u,v} = 0$ for nonexistent edges. Thus from here on, when we refer to an edge (u, v) as *incident* to a vertex v, or to u being *adjacent* to or a *neighbor* of v, we mean that $p_{u,v} > 0$ (i.e., that edge (u, v) has a positive probability of existence). We are further given a *patience* value t_j for each online vertex $v_j \in V$ that signifies the number of times we are allowed to probe different edges incident on v_j when it arrives. Note that each edge may be probed at most once and if it exists, we must match it and stop probing (probe-commit model).

Strictly speaking, G specifies a *probability distribution* on input graphs, the true realization of which is initially unknown. We denote this *realization* graph by $\tilde{G} = (U, V, \tilde{E})$, where $\tilde{E}$ consists only of edges which exist when probed.

We consider the online vertices arriving in *stages*. Specifically, we may assume (without loss of generality) that the online vertices arrive in the order $v_1, v_2, \ldots, v_n$ and number the stages 1 through n correspondingly. When the online vertex $v_k \in V$ arrives at stage k, we attempt to match it to an available offline vertex. We are allowed to probe edges incident to v_k one-by-one, stopping as soon as an edge (u_i, v_k) is found to exist, at which point the edge is included in the matching and we receive a reward of w_i. We are allowed to probe a maximum of t_k edges; if t_k edges are probed and none of the edges exist, then vertex v_k remains unmatched and we receive no

reward. If we successfully match v_k to u_i, we say that w_i is the *value* or *reward* of v_k's match; if v_k remains unmatched, we say it has a value or reward of 0.

3.4.4 Unifying and Clarifying the Competitive Ratio

A key argument in this section is that we should unify and clarify the definition of competitive ratio for the problem of online matching with stochastic rewards. Currently, the most common definition [20, 30, 31, 44] compares an online algorithm to the offline optimal for the corresponding offline stochastic matching problem introduced in [30]. However, in [14], they compare an online algorithm to a specific non-stochastic offline packing problem called Budgeted Allocation. Budgeted Allocation is equivalent to LP 3.10 in Section 3.4.5, but with all $t_v = 1$ since it addresses the stochastic rewards problem without patience. We note that the Budgeted Allocation LP and its natural extension to patience constraints (LP 3.10) both *upper bound* the corresponding offline stochastic matching problems. However, we argue that these LPs should not be used as tight bounds to prove hardness results as with Budgeted Allocation in [14].

We use CR_{stoch} to refer to the first definition (Definition 2) since it compares to a stochastic offline problem (offline stochastic matching) and CR_{non} to refer to the definition from [14], which compares to a non-stochastic offline problem (Budgeted Allocation). In this section, we advocate for CR_{stoch} as the canonical definition of competitive ratio for online matching problems with stochastic rewards. Use of the term "competitive ratio" in the analysis of future sections refers to CR_{stoch}.

We give the following reasons for choosing CR_{stoch}:

1. CR_{stoch} is more in line with the standard concept of competitive ratio. We compare an online algorithm for a problem to the offline optimal for that same problem.

2. It enables finer grained comparison between algorithms. We will describe an example below where an online algorithm which is optimal under the CR_{non} definition can be improved upon under CR_{stoch}.

3. In Section 3.4.6, we define the concept of a stochasticity gap to capture the gap between the offline stochastic matching problem and LPs such as Budgeted Allocation. For the problem with patience constraints, we show in Section 3.4.7 that this gap is quite large.

In [14], they argue against three specific potential definitions of competitive ratio. We agree with those arguments. However, we further argue that their definition, CR_{non}, is too pessimistic. To support point (2) above, we note that in [19], they show an algorithm for online matching with stochastic rewards and known IID arrivals that achieves a competitive ratio of $1 - 1/e$ (under both CR_{stoch} and CR_{non}). Under CR_{non}, this result would be tight with no further improvement possible. However, under CR_{stoch} (the definition used in [19]), they also show that for the case of uniform constant edge probabilities, a competitive ratio of 0.702 is possible using a more constrained LP to guide the algorithm. This finer granularity of analysis supports the develop of improved algorithms that would be impossible to see under CR_{non}.

In Section 3.4.5, we describe the natural LP which is often used as an upper bound on the offline optimal under CR_{stoch} and as the definition of offline optimal under CR_{non} (called Budgeted Allocation in [14]). Then, in Sections 3.4.6 and 3.4.7, we show how there is a large gap between the solution to this LP and the optimal offline stochastic matching solution. This implies that under CR_{non}, no online algorithm for stochastic matching with patience could achieve a competitive ratio better than 0.544 even under the more tractable known IID arrival model. We believe this is far too pessimistic and conjecture that algorithms which exceed that ratio under CR_{stoch} will be found in the future.

3.4.5 Standard Linear Programing Relaxation

Below is a natural extention of the "standard" LP formulation (e.g. as in both [20, 30, 31, 44] and [14]) to vertex weights and non-unit patience values.

$$\text{maximize} \sum_{u \in U} \sum_{v \in V} x_{u,v} p_{u,v} w_u \tag{3.10}$$

$$\text{subject to} \sum_{v \in V} x_{u,v} p_{u,v} \leq 1, \qquad \forall u \in U \tag{3.10a}$$

$$\sum_{u \in U} x_{u,v} p_{u,v} \leq 1, \qquad \forall v \in V \tag{3.10b}$$

$$\sum_{u \in U} x_{u,v} \leq t_v, \qquad \forall v \in V \tag{3.10c}$$

$$0 \leq x_{u,v} \leq 1, \qquad \forall u \in U, v \in V \tag{3.10d}$$

We note that the Budgeted Allocation problem LP of Mehta and Panigrahi [14]

uses different notation, but is equivalent if all t_v are set to 1. The notation we use here is in keeping with [20, 30, 31, 44]. Some formulations such as [20, 44] use edge weights instead of vertex weights in the objective or consider additional patience constraints on the offline vertices (two-sided timeouts in [44]).

In most cases, such an LP is used to upper bound the optimal solution. The first two constraints are non-stochastic relaxations of the matching constraint. The third constraint enforces the patience constraint that a vertex v can be probed at most t_v times. However, this is not a tight bound on the optimal solution as we will see in Section 3.4.6 which defines the concept of a stochasticity gap between an optimal stochastic matching algorithm and the optimal solution to LP 3.10. Later, in Section 3.4.12, we present a new LP formulation (LP 3.12) that introduces additional new constraints. Being more constrained, this new LP provides a tighter bound on OPT, although it is open whether it achieves a provably better stochasticity gap than LP 3.10.

3.4.6 Stochasticity Gap

In keeping with our mission, we now formalize the definition of a *stochasticity gap* for matching problems. The term was first used casually in this context in [44] without a rigorous definition. They showed that there is a gap of at least $1 - 1/e$ between the LP solution and an optimal algorithm's solution. In other words,

$$\frac{\mathbb{E}[\text{OPT}]}{\text{LPOPT}} \leq 1 - 1/e$$

For the offline problem, this gap arises with a star graph on n vertices with each edge e having $p_e = 1/n$ and the center vertex having unlimited patience (or equivalently $t_v = n$). To create a similar problem instance for online matching with stochastic rewards, let the center of the star be the offline set (with unlimited patience) and the remaining vertices be the online set. In both cases, the LP can assign 1 to all variables to get LPOPT $= 1$ while $\mathbb{E}[\text{OPT}] = 1 - 1/e$ for large n since there is a $(1 - 1/n)^n = 1/e$ probability of no edge existing. We give a more general definition below which captures this concept.

Definition 9 (Stochasticity Gap)**.** *The ratio of the optimal algorithmic solution for a stochastic packing problem to the optimal solution of a linear programming relaxation which treats probabilities as deterministic fractional size coefficients.*

3.4.7 A Larger Stochasticity Gap

As stated in our problem definitions, online matching with stochastic rewards and patience constraints (aka online matching with timeouts) generalizes the online matching with stochastic rewards problem. It allows each online vertex v to probe up to t_v neighbors when it arrives instead of just one.

To see a larger gap for this problem, consider the unweighted, complete bipartite graph $K_{n,n}$ (n vertices in each partition) with $p_e = 1/n$ for all edges e and unlimited patience (or equivalently $t_v = n$) on all vertices. In this case, LP 3.10 has value of n, achieved by assigning a value of 1 to each variable. However, we can upper bound $\mathbb{E}[\text{OPT}]$ with the expected size of the maximum matching in the

realization graph G of all edges that actually exist. In other words, imagine we probed every edge and sought a maximum matching in the graph of edges that were found to exist.

To do this, we first mention the following result due to [48] for random graphs. Note, however, that Theorem 14 of [48] is slightly more general, providing a bound on the size of the independent set for $p = c/n$. Lemma 4 states the special case for $c = 1$.

Lemma 4 ([48]). *Let G be a random bipartite graph with both partitions of size n and where each edge exists independently with probability $p = 1/n$. Let γ be the solution to the equation $\gamma = e^{-\gamma}$. Then, the largest independent set of G has size $n(2\gamma + \gamma^2))[1 + o(1)]$ with probability $1 - o(1)$.*

The proof of Theorem 4 can be derived from Lemma 4 as follows. Lemma 4 implies that, almost surely, a minimum vertex cover for G has size (asymptotically, as $n \to \infty$) $2n - n(2\gamma + \gamma^2) \approx 0.544n$, and by Kőnig's Theorem this is equivalent to the size of the maximum matching in G (see also [49, 50]). It follows, then, that no online or offline algorithm can achieve an expected matching size greater than $\approx 0.544n$ on this graph. This shows a stochasticity gap of at least $0.544n/n = 0.544$.

Thus, Linear Program 3.10 can overestimate the true optimal value quite drastically considering that the stochasticity gap of LP 3.10 is an upper bound on the competitive ratio for any algorithm using that LP to upper bound OPT and we hope for a competitive ratio higher than 0.544 for many online matching problems. In particular, the algorithm in [44] achieved a 0.46 competitive ratio for the edge-

weighted, known IID arrivals variant using LP 3.10 as an upper bound (as well as to guide the algorithm). We can see here that this ratio cannot be improved beyond 0.544 even in unweighted graphs without using a tighter upper bound.

Another problem with defining the competitive ratio as $\mathrm{CR}_{\mathrm{non}}$ is that it would imply that no online algorithm can achieve a ratio better than 0.544. However, using the definition $\mathrm{CR}_{\mathrm{stoch}}$ opens up the possibility of using a tighter bound on the offline optimal solution than LP 3.10.

3.4.8 Optimal Offline Stochastic Matching Strategy for Star Graphs

Here, we prove Theorem 5 by describing a dynamic programming algorithm for solving edge-weighted offline stochastic matching with patience on star graphs—that is, bipartite graphs $G = (U, V, E)$ in which V consists of a single vertex. Note that in this case, the edge-weighted and vertex-weighted problems are equivalent and the only patience constraint we need to consider is on the center vertex.

To see the relationship between this problem and online matching, observe that when an online vertex v arrives, we are given the star graph of v and its offline neighbors in G. A greedy algorithm seeks a probing strategy which maximizes the expected weight of a matching in that star graph. Consequently, observe that offline matching on a star graph is equivalent to online matching with a single online vertex and a single online stage.

Let S be a star graph with center vertex v, and let $U = S \setminus v$ denote the set of v's neighbors in S. Suppose $U = \{u_1, \ldots, u_m\}$ and $V = \{v\}$. The optimal strategy

for matching v is the one which maximizes the expected value of v's match. We show a dynamic programming approach which finds this optimal solution. Let p_i denote the probability that edge (u_i, v) exists when probed.

A crucial observation is that any optimal probing strategy will probe in non-increasing order of edge/vertex weight, regardless of the edge probabilities. Intuitively, this results in matching v to its highest weight neighbor with an edge that actually exists among the subset of U that is being probed. Claim 10 states this more formally.

Claim 10. *Let $U' \subseteq U$ be a subset of the offline vertices. Consider querying all edges in the set $\{(u, v) \mid u \in U'\}$ according to some ordering. Ordering the edges by decreasing weight maximizes the expected weight of v's match (with respect to all other orderings of U').*

Given Claim 10, we may restrict our probing strategies to those which probe edges in non-increasing order of weight. For ease of exposition, let U be sorted in non-increasing order of weight such that $w_{u_1} \geq w_{u_2} \geq \ldots \geq w_{u_m}$. For our dynamic program, we define $f(i, t)$ to be the maximum possible expected value of any decreasing-weight probing strategy that is allowed t probes and probes edge (u_i, v) first. The full definition of f is given in (3.11).

$$f(i,t) = \begin{cases} p_{u_i,v} w_{u_i} & \text{if } t = 1 \\ p_{u_i,v} w_{u_i} + (1 - p_{u_i,v}) \max\limits_{j>i} f(j, t-1) & \text{if } t > 1 \end{cases} \tag{3.11}$$

The following lemma states that this formulation does indeed provide the

expected weight of the optimal probing strategy.

Lemma 5. *For a patience of t_v, the value $\max_i f(i, t_v)$ is equal to the expected value of the optimal probing algorithm.*

Proof. We begin by showing that for all $t \geq 1$ and for all i, the value $f(i, t)$ equals the maximum possible expected matching weight with t probes, if we first probe edge (u_i, v) and proceed probing edges in decreasing weight. We proceed by induction on t. Clearly, this holds for $f(i, 1)$ (for all i). Now suppose that for all j, $f(j, t-1)$ equals the maximum possible expected matching weight with $t-1$ probes, if we first probe edge (u_j, v) (and subsequently probe edges of lower weight).

If we probe (u_i, v) first, we achieve an expected matching size of $p_{u_i,v} w_{u_i} + (1 - p_{u_i,v})\mathbf{E}$, where $\mathbf{E}$ is the expected matching weight achieved by the remaining probes. By the inductive hypothesis, this is maximized (over all vertices whose weight is less than w_{u_i}) by $\max_j f(j, t-1)$.

It follows from the above that given a patience t_v, the value $\max_i f(i, t_v)$ represents the maximum possible expected matching weight if we are restricted to probing edges in order of decreasing weight. However, it follows from Claim 10 that such a strategy is also optimal over all possible probing orders, and thus $\max_i f(i, t_v)$ is the expected value of an optimal probing algorithm. □

Clearly, we can construct a table storing all values of $f(i, t)$ with $i \leq m$ and $t \leq t_v$ using at most $O(mt_v)$ space. Computing each cell of the table requires at most $O(m)$ time to find $\max_{j>i} f(j, t-1)$ resulting in at most $O(m^2 t_v)$ time. Since t_v must be less than m, the time and space are guaranteed to be polynomial in the

size of the input. This procedure is stated explicitly in Algorithm 2.

Algorithm 2: Given a star graph with center v and patience t_v, solve Dynamic Program 3.11 and compute the optimal probing strategy

```
Function StarDP(v, t_v, p, w):
    for i := 1 to m do
        W[i, 1] ← p_{u_i,v} w_{u_i}
    for t := 2 to t_v do
        for i := 1 to m do
            j* ← arg max_{j>i} W[j, t − 1]
            W[i, t] ← p_{u_i,v} w_{u_i} + (1 − p_{u_i}) W[j*, t − 1]
            V[i, t] ← j*
    i*_0 ← max_i W[i, t_v]
    for t := 1 to t_v do
        i*_t ← V[i*_{t−1}, t_v − t + 1]
    return (i*_0, i*_1, ..., i*_{t_v−1})
```

Proof of Theorem 5. This follows immediately from Lemma 5, and the observation that Algorithm 2 solves Dynamic Program 3.11 in time $O(m^2 t_v)$, producing a probing strategy with expected matching weight equal to $\max_i f(i, t_v)$. □

3.4.9 Greedy Algorithms For Online Stochastic Matching

While greedy algorithms can provide powerful heuristics for online matching problems [5], it is not obvious how to behave greedily in the presence of vertex weights, stochastic edges, and patience constraints. We first illustrate how a naive greedy approach fails. We then show how to optimally probe a star graph. Finally, we analyze this greedy algorithm to bound its competitive ratio at 0.5 which is tight for worst case analysis of greedy algorithms for online matching.

3.4.10 A Naive Greedy Approach that Fails

One natural idea which may appear to generalize the common greedy approaches to similar problems is to sort the neighbors of an arriving vertex v in non-increasing order of *expected weight*, $w_u p_{uv}$. However, the competitive ratio of this approach can be arbitrarily bad as described in Theorem 6.

Proof. Consider the following underlying bipartite graph. Let the offline vertex set U contain one vertex u' of weight 1 and n vertices $u_1, u_2, \ldots, u_n$ of weight n, where n is some very large number. Let there be exactly one online vertex v with an edge of probability $p_{u'v} = 1$ to the vertex u' with weight 1 and edges of probability $p_{u_i v} = 1/(n+1)$ to the remaining offline vertices u_i for $i \in \{1, 2, \ldots, n\}$. Let v have a patience $t_v = n + 1$, meaning that it can probe as many neighbors as we want until it is matched or we run out of neighbors to probe. Note that we can always add "dummy" vertices (vertices with no neighbors) to the online set if we want to capture the setting where both the online and offline sets are large.

The strategy of sorting by expected weight will first probe the edge (u', v) because it has the largest expected weight of 1 while the other edges have an expected weight of $n/(n+1) < 1$. Since the edge (u', v) has probability 1 of existing and we are in the probe-commit model, this would match v to u' deterministically, earning a weight of 1. However, the optimal algorithm would probe u' *last* (probing first the vertices $u_1, u_2, \ldots, u_n$), earning an expected weight of $(1 - 1/e)n + 1/e$. □

Thus, to properly generalize greedy approaches to this setting, we need to fix

a probing order that maximizes the expected weight achieved by an arriving vertex. In Section 3.4.8, we show how to do this using dynamic programming.

3.4.11 A 0.5-Competitve Online Algorithm

We present a greedy algorithm which achieves a 0.5-approximation for online matching with vertex weights, stochastic rewards, and patience constraints in the adversarial arrival model. In this setting, the vertices of V arrive in an online fashion. If vertex v is the kth vertex to arrive online, we say v *arrives at time* k.

Algorithm 3: Use Dynamic Program 3.11 to greedily match arriving vertices

```
Function DPGreedy(U, V, p, w):
    for Online arriving vertex v ∈ V, with patience t_v do
        (i*_0, ..., i*_{t_v - 1}) ← StarDP(v, t_v, p, w)
        for t := 0 to t_v − 1 do
            Probe edge (u_{i*_t}, v)
```

The algorithm is as follows. When a vertex v_k arrives at time k, let U' denote the set of offline vertices which are still unmatched. Solve the Dynamic Program 3.11 on the star subgraph $S_{U',v_k} = (U', \{v\}, U' \times \{v\})$. Probe edges in the order given by the dynamic program (v is matched to the first vertex u for which the edge (u, v) exists when probed). This procedure is stated explicitly in Algorithm 3.

Let $\text{ALG}(G)$ denote the expected size of the matching produced by this algorithm on the graph G. Let $\text{OPT}(G)$ denote the expected size of the matching produced by an optimal *offline* algorithm. Our main result is given by Theorem 11.

Theorem 11. *For any bipartite graph* G, $\frac{\text{ALG}(G)}{\text{OPT}(G)} \geq 0.5$.

3.4.12 A New LP with a Tighter Upper Bound on OPT

We formulate a new LP by adding a new constraint to LP 3.10. This new LP gives a tighter upper bound on the offline optimal solution, OPT(G). Note that our algorithm does not need to explicitly solve this new LP. We simply use it in our analysis.

This new constraint is motivated by the observation that the dynamic program (3.11) is optimal for star graphs. Intuitively, for any subgraph G' of G, the optimal solution for G' cannot match more edges in expectation than OPT(G). We can represent this as a set of constraints which restrict the LP to ensure that the expected number of matched vertices on any star subgraph of G does not exceed the optimal value given by the dynamic program (3.11) for that same star subgraph. This is captured in the new constraint, (3.12d), in the linear program 3.12 below. In this constraint, we slightly abuse notation and write OPT(U', v) to denote OPT(G') for a star graph $G' = (U', \{v\}, U' \times \{v\})$. Recall that $OPT(U', v)$ is given by the

dynamic program (3.11).

$$\text{maximize} \sum_{u \in U} \sum_{v \in V} x_{u,v} p_{u,v} w_u \tag{3.12}$$

$$\text{subject to} \sum_{v \in V} x_{u,v} p_{u,v} \leq 1, \qquad \forall u \in U \tag{3.12a}$$

$$\sum_{u \in U} x_{u,v} p_{u,v} \leq 1, \qquad \forall v \in V \tag{3.12b}$$

$$\sum_{u \in U} x_{u,v} \leq t_v, \qquad \forall v \in V \tag{3.12c}$$

$$\sum_{u \in U'} x_{u,v} p_{u,v} w_u \leq \text{OPT}(U', v), \qquad \forall U' \subseteq U, v \in V \tag{3.12d}$$

$$0 \leq x_{u,v} \leq 1, \qquad \forall u \in U, v \in V \tag{3.12e}$$

Let $\text{LPOPT}(G)$ denote the value of Linear Program 3.12 on the graph G. Lemma 6 states that this new LP is still a valid upper bound on the optimal solution.

Lemma 6. *For any bipartite graph G, $\text{LPOPT}(G) \geq \text{OPT}(G)$.*

Proof. Consider an adaptive offline algorithm which is optimal. Let $x_{u,v}$ be the probability that this strategy probes edge (u, v). For any vertex $u \in U$, the probability that u is successfully matched is at most $\sum_{v \in V} x_{u,v} p_{u,v} \leq 1$, and similarly for the probability of successfully matching any online vertex $v \in V$. Thus, this assignment satisfies constraints (3.12a) and (3.12b). By the definition of OPT, we cannot probe more than t_v edges incident on an online vertex v. So constraint (3.12c) is satisfied.

Finally, we argue that the new constraint (3.12d) is satisfied by this assignment. Suppose instead there is some vertex $v' \in V$ and some $U' \subseteq U$ for which

$\sum_{u \in U'} x_{u,v'} p_{u,v'} w_u > \text{OPT}(U', v')$. Then, we can define a new offline probing strategy on the star graph $(U', \{v'\}, U' \times \{v'\})$ which simply simulates our original algorithm on G and probes only those edges which are in $U' \times \{v'\}$. This achieves an expected matching weight on the star graph of at least $\sum_{u \in U'} x_{u,v'} p_{u,v'} w_u > \text{OPT}(U', v')$, but this contradicts the fact that $\text{OPT}(U', v')$ is the optimal expected matching weight for the star graph. Thus, this assignment must satisfy constraint (3.12d). It follows that LP 3.12 must have objective value at least as large as the expected matching weight of the optimal offline algorithm. □

3.4.13 Analysis of the DP-based Greedy Algorithm

We will bound the performance of our greedy algorithm relative to the solution of Linear Program 3.12. Lemma 6 then implies that this bounds the competitive ratio. In particular, the following lemma, along with Lemma 6, implies Theorem 11.

Lemma 7. *For any bipartite graph G,* $\text{ALG}(G) \geq 0.5\,\text{LPOPT}(G)$.

Proof. For the sake of analysis, suppose we have solved LP 3.12 on the graph G. Let

$$c(x) = \sum_{u \in U} \sum_{v \in V} x_{u,v} p_{u,v} w_u$$

be the value of the objective function for x and let

$$c_v(x) = \sum_{u \in U} x_{u,v} p_{u,v} w_u$$

be the value "achieved" by a given online vertex v with $c(x) = \sum_{v \in V} c_v(x)$. Let

x^* be the optimal assignment given by LP (3.12), for the graph G. So $c(x^*) = \sum_{v \in V} c_v(x^*) = \text{LPOPT}(G)$.

We will make the following charging argument. Imagine that when a vertex v is matched to some $u \in U$, we assign $0.5w_u$ to v and for all $v' \in V$ (including v itself) we assign $0.5x_{u,v'}p_{u,v'}w_u$ to v'. Note we have assigned at most w_u weight in total since $\sum_{v' \in V} 0.5x_{u,v'}p_{u,v'}w_u \leq 0.5w_u$ due to LP constraint (3.12a).

Let w_v for online vertex $v \in V$ be equal to the weight w_u of the offline vertex u which is matched to v or 0 if v is unmatched at the end of the arrivals. Let $U_m \subseteq U$ be the set of offline vertices which are matched at the end of the arrivals. We define

$$c_v(\text{ALG}) = 0.5w_v + 0.5 \sum_{u \in U_m} x_{u,v}p_{u,v}w_u$$

as the weight assigned to v in our imaginary assignment. By the linearity of expectation

$$\text{ALG}(G) = \mathbb{E}\left[\sum_{v \in V} c_v(\text{ALG})\right] = \sum_{v \in V} \mathbb{E}[c_v(\text{ALG})]$$

Thus, to complete the proof, we must show that

$$\sum_{v \in V} \mathbb{E}[c_v(\text{ALG})] \geq \frac{1}{2}\,\text{LPOPT}(G)$$

Consider an online vertex v arriving at time k. Let $U_v \subseteq U$ be the set of vertices available (unmatched) when v arrives and $U_{-v} = U \setminus U_v$ be the set of vertices which are already matched when v arrives. Note that when v arrives, it has already been assigned a value of $0.5\sum_{u \in U_{-v}} x_{u,v}p_{u,v}w_u$. After attempting to match

v to U_v according to DP (3.11), we have assigned an expected value to v of at least $0.5\,\mathrm{OPT}(U_v, v) + 0.5\sum_{u\in U_{-v}} x_{u,v}p_{u,v}w_u$.

Thus, we have

$$\begin{aligned}
\sum_{v\in V}\mathbb{E}[c_v(\mathrm{ALG})] &\geq \sum_{v\in V}\sum_{U_v\subseteq U}\Pr[U_v]\left(0.5\,\mathrm{OPT}(U_v, v) + 0.5\sum_{u\in U_{-v}} x_{u,v}p_{u,v}w_u\right)\\
&\geq \sum_{v\in V}\sum_{U_v\subseteq U}\Pr[U_v]\left(0.5\sum_{u\in U_v} x_{u,v}p_{u,v}w_u + 0.5\sum_{u\in U_{-v}} x_{u,v}p_{u,v}w_u\right)\\
&\geq 0.5\sum_{v\in V}\sum_{U_v\subseteq U}\Pr[U_v]\sum_{u\in U} x_{u,v}p_{u,v}w_u\\
&= 0.5\sum_{v\in V}\sum_{u\in U} x_{u,v}p_{u,v}w_u\\
&= \frac{1}{2}\,\mathrm{LPOPT}(G)
\end{aligned}$$

□

Lemma 7 now implies the main result, a $\frac{1}{2}$-competitive ratio.

Proof of Theorem 11. By Lemmas 6 and 7, we have

$$\mathrm{ALG}(G) \geq 0.5\,\mathrm{LPOPT}(G) \geq 0.5\,\mathrm{OPT}(G)$$

□

3.4.14 A 1/2 Upper Bound for Greedy Under Stochastic Rewards

In [14], Mehta and Panigrahi showed that in the unweighted *Stochastic Rewards* (patience of 1 for online vertices) problem, any algorithm which is "oppor-

tunistic" achieves a competitive ratio of 1/2. As per [14], an *opportunistic* algorithm for the *Stochastic Rewards* setting is an algorithm which always attempts to probe an edge incident to an online arriving vertex $v \in V$ if one exists. They show that any opportunistic algorithm achieves a competitive ratio of at least 1/2.

The most simple opportunistic algorithm is the one which, when $v \in V$ arrives online, chooses a neighbor $u \in U$ of v arbitrarily and probes the edge (u, v). We call this algorithm "SimpleGreedy". The result of [14] shows that SimpleGreedy achieves a competitive ratio of at least 1/2. Theorem 8, proven below, shows that this is tight even when restricted to small, uniform p.

Proof. Let k be a fixed positive integer constant. Let $U = U_0 \cup U_n$, where U_0 and $U_n = \{u_1, \ldots, u_n\}$ are disjoint, and $|U_0| = k$. Let $V = V_0 \cup V_n$ where V_0 and $V_n = \{v_1, \ldots, v_n\}$ are disjoint, and $|V_0| = kn^2$. Let $E = E_0 \cup E_n$ where $E_0 = U_0 \times V$ and $E_n = \{(u_i, v_i) \mid i \in [n]\}$.[2] Let $p = k/n$.

For the bipartite graph $G(U, V; E)$, an offline algorithm can achieve a matching of expected size at least $2k$ by first probing edges $(u, v) \in U_0 \times V_0$ until all edges are probed or the maximum possible successful matches, k, is achieved. This strategy achieves k successful matches among these edges in expectation. Then, the offline optimal will probe all edges of E_n in any order, achieving an expected number of successful matches of k. The total expected size of the achieved matching is then $2k$.

On the other hand, an adversary in the online setting may expose all vertices of V_n before any of the vertices of V_0 to the greedy algorithm. SimpleGreedy choosing

[2] We use the notation $[n] = \{1, 2, \ldots, n\}$

arbitrarily may in the worst case choose to probe edges of E_0 first, preventing some vertices of V_0 from being matched later. We consider the case where SimpleGreedy chooses an edge $(u, v_i) \in E_0$ for each online vertex $v_i \in V_n$ if any $u \in U_0$ is available at v_i's arrival. We calculate the expected size of the matching produced by this strategy.

Let M be a random variable corresponding to the size of the final matching, and let M_0 and M_n be random variables corresponding to the number of matched vertices in V_0 and V_n, respectively. Then, the expected size of the matching is

$$\mathbb{E}[M] = \mathbb{E}[M_n] + \mathbb{E}[M_0] = k + \mathbb{E}[M_0].$$

We now consider $\mathbb{E}[M_0]$. If $l < k$ vertices of V_n are matched successfully, then when the vertices of V_0 arrive online, there will only be $k - l$ vertices of U_0 remaining to be matched. Since $|V_0| = kn^2$, greedy will almost surely match all of them successfully. Thus, we get (as $n \to \infty$)

$$\begin{aligned}
\mathbb{E}[M_0] &= \sum_{l=0}^{k-1} \binom{n}{l} (1-k/n)^{n-l} (k/n)^l (k-l) \sim \sum_{l=0}^{k-1} (k-l) \frac{n^l}{l!} e^{-k(n-l)/n} \frac{k^l}{n^l} \\
&\sim \sum_{l=0}^{k-1} (k-l) \frac{e^{-k}k^l}{l!} = k \sum_{l=0}^{k-1} \frac{e^{-k}k^l}{l!} - \sum_{l=1}^{k-1} \frac{e^{-k}k^l}{(l-1)!} \\
&= k \left[\sum_{l=0}^{k-1} \frac{e^{-k}k^l}{l!} - \sum_{i=0}^{k-2} \frac{e^{-k}k^l}{l!} \right] = k \cdot \frac{e^{-k}k^{k-1}}{(k-1)!} = k \cdot \frac{e^{-k}k^k}{k!}.
\end{aligned}$$

Finally, we observe that for large k, $e^{-k}k^k/k! \sim (2\pi k)^{-1/2}$, due to Stirling's formula.

Thus, we get a competitive ratio of

$$\frac{\mathbb{E}[M]}{\mathrm{OPT}} = \lim_{k\to\infty} \frac{k + \sqrt{k/(2\pi)}}{2k} = \frac{1}{2} \tag{3.13}$$

□

With a more intricate argument, we can also show that the same upper-bound of 1/2 even holds for the "random greedy" algorithm, where an online vertex gets matched to a *random* available neighbor (if any).

Chapter 4: Constrained Clustering

In this chapter, we present our work on various constrained clustering problems. Section 4.1 gives approximation algorithms for several models of constrained clustering including some new models we introduce. We also prove NP-hardness for one of the new models that does not directly inherit hardness from known problems. Then, in Section 4.2, we introduce the notions of pairwise fairness and community preservation for the k-center problem. We give an approximation algorithm for these new problems and perform experiments to show how the trade-off between fairness and optimization can be adjusted in practice.

4.1 Metric Clustering with Pairwise Constraints

Two of the most famous and well-studied clustering problems in areas ranging from combinatorial optimization to operations research to machine learning are k-center and k-median. We are given a set of points V in some metric space represented by a distance function $d : V \times V \mapsto \mathbb{R}^+$. The goal is to select at most k points to serve as *centers* and assign each of the remaining points to one of these centers. In *k-center*, the objective is to minimize the maximum distance of any point to its assigned center, while in *k-median*, we aim to minimize the average assignment

distance. We can view the chosen centers as defining clusters, where each cluster includes the points assigned to the corresponding center.

Pairwise constraints represent additional requirements or knowledge about whether or not specific pairs of points belong together and can be completely unrelated to the underlying metric space. Such constraints arise naturally in a variety of applications (see Section 4.1.2 for examples). In this work, we consider four types of pairwise constraints: bounded separation probabilities, hard must-link constraints, soft must-link constraints, and cannot-link constraints. With *bounded separation probabilities (BSP)*, pairs of points $e = \{u, v\}$ are given a value $p_e \in [0, 1]$ and we must find a randomized clustering where the probability of assigning u and v to different clusters is at most p_e. *Hard must-link constraints* define pairs of points which must be assigned to the same cluster in any valid clustering. This is a special case of bounded separation probabilities where each p_e is either 0 or 1. *Soft must-link constraints* are defined through a set S that contains pairs of points, each pair indicating a hard must-link constraint. However, in any feasible solution, we are allowed to violate at most a certain fraction of the constraints in S in expectation. Hence, given a number $\psi \in [0, 1]$ as input, our randomized clustering should satisfy at least $(1 - \psi)|S|$ of the pairs in S in expectation. As with BSP, we can see that hard must-link constraints are also a special case of soft must-link constraints where $\psi = 0$. When we allow multiple sets of soft constraints, as we do in our model, then soft constraints also generalize BSP (we can add each BSP constraint to its own soft set). Finally, *cannot-link constraints* state that pairs of points should be separated and are the natural complement of must-link constraints. Incorporating any of the

above constraints into k-center or k-median generalizes the original problem, and so the new variants remain NP-hard.

Our goal here is to study the above mentioned constraints in the classical settings of k-center and k-median. Along the way, we propose a related problem where there is no limit on the number of centers (we refer to this as the "no-k" version). While the classical settings are trivial to solve with unlimited centers, the problem becomes challenging again when pairwise constraints are added. We formally prove that through an NP-hardness reduction. We also show how our algorithmic framework can be extended to three broader problems, namely knapsack-center, matroid-center, and k-supplier. For all constraint cases, except cannot-link constraints, we provide algorithms with small constant factor approximations. On the negative side, we show that finding a solution that minimizes a violation function of the cannot-link constraints is NP-hard.

4.1.1 Formal Problem Definitions

In this section we formally define the main problems we study. In all settings, we are given a set of points V with $n = |V|$. Let $d(u, v) \geq 0$ represent the distance between any $u, v \in V$. Distances obey the triangle inequality, i.e., for $u, v, w \in V$ we have $d(u, v) \leq d(u, w) + d(w, v)$. We are interested in choosing a set of centers $C \subseteq V$ and an assignment $\phi : V \mapsto C$ of points to chosen centers that would satisfy each problem's specific requirements. Note that, due to pairwise constraints, we do not simply assign each point to its nearest center as is common in classical variants.

k-center with stochastic pairwise constraints. Here, we are given an integer $k > 0$ denoting the maximum number of centers that can be selected. For the bounded separation probabilities, we are given a set $P \subseteq V \times V$ of pairs of points $e = \{u, v\} \in P$, where $u, v \in V$ have a distinct separation probability of $p_e \in [0, 1]$. This value indicates the maximum probability with which we are allowed to separate u and v in any feasible solution. For soft must-link constraints, we are given a family of sets $S = \{S_1, S_2, \ldots\}$ with $S_i \subseteq V \times V$ and a family of fractions $\psi = \{\psi_1, \psi_2, \ldots\}$ with $\psi_i \in [0, 1]$. Each S_i represents a soft set of must-link pairs of points, with the requirement that we are allowed to separate at most $\psi_i |S_i|$ of them in expectation. Unlike classical k-center, we naturally require a randomized assignment to accommodate the stochastic constraints. We seek the minimum R for which there exists a set of centers $C \subseteq V$, with $|C| \leq k$, and an efficiently-samplable probability distribution over mappings $V \to C$, such that for a mapping ϕ sampled from this distribution, we have the following.

- $\sum_{u \in C} Pr[\phi(v) = u] = 1$, for every $v \in V$ (each $v \in V$ must be assigned to some cluster)
- $Pr[d(v, \phi(v)) \leq R] = 1$ for every $v \in V$ (enforcing the radius R)
- $Pr[\phi(u) = u] = 1$, for every $u \in C$ (centers should be assigned to the cluster they define)
- $Pr[\phi(v) \neq \phi(w)] \leq p_e$, $\forall e = \{v, w\} \in P$ (bounded separation probabilities)
- $\forall S_i \in S : \sum_{\{v,w\} \in S_i} Pr[\phi(v) \neq \phi(w)] \leq \psi_i |S_i|$ (soft must-link constraints)

No-k-center with stochastic pairwise constraints. This variant is exactly the same as the problem described above, with one key difference: we have no cardinality constraint on the set of chosen centers C. This means that we are allowed to pick as many centers as we want while obeying pairwise constraints and minimizing the radius.

k-center with hard must-link constraints. This setting is again similar to k-center with stochastic pairwise constraints, but with a few differences. First of all, there is no soft must-link set S. All constraints are captured by the set $P \subseteq V \times V$ of pairs of points $e = \{u, v\} \in P$. However, if $\{u, v\}$ is such a pair of P then these two points must deterministically be placed in the same cluster. In other words, $\phi(u) = \phi(v)$ with probability one.

k-median with hard must-link constraints. For this setting, we use the standard assumption used in the k-median literature, that distinguishes the points that are clients and require service from those that can serve as facilities. Hence, we have a set of points V, a set of facilities F, and a set $P \subseteq V \times V$ capturing the must-link constraints. We want a subset $F' \subseteq F$, with $|F'| \leq k$, and an assignment $\phi : V \mapsto F'$ of the points in V to the centers chosen in F', such that $\forall \{v, w\} \in P : \ \phi(v) = \phi(w)$ and $\sum_{v \in V} d(v, \phi(v))$ is minimized.

k-center with generalized cannot-link constraints. Here we are given a parameter $k > 0$, indicating the maximum number of centers we can open, a target radius R, a set of cannot-link constraints P and an arbitrary non-negative function $f(P, \phi)$ that penalizes unsatisfiable cannot-link constraints under the assignment $\phi : V \mapsto C$, where C the set of chosen centers. The goal is to open a set of centers

C with $|C| \leq k$, and to come up with an assignment function $\phi : V \mapsto C$, such that $\forall v \in V : d(v, \phi(v)) \leq R$ and $f(P, \phi)$ is minimized. We say that a cannot-link constraint $e = \{v, w\} \in P$ is violated when $\phi(v) = \phi(w)$. The only requirement we impose on f is that when no constraint is violated, its value should be 0.

Knapsack-center with stochastic pairwise constraints. Each point $u \in V$ has a non-negative cost $c_u \geq 0$, and we are given a global budget B. In this case, we would like to find a set of centers C such that $\sum_{u \in C} c_u \leq B$. The pairwise requirements remain the same as in k-center with stochastic pairwise constraints.

Matroid-center with stochastic pairwise constraints. Here, the input also includes a matroid $\mathcal{M}(V, I)$, where V is the set of points and $I \subseteq 2^V$ contains the independent sets of $\mathcal{M}$. We would like to find a set of centers C such that $C \in I$. Once more, the pairwise requirements remain the same as in k-center with stochastic pairwise constraints.

k-supplier with stochastic pairwise constraints. Here the input points are given in the form of two disjoint sets F and V. The goal is to find a set of centers $C \subseteq F$, with $|C| \leq k$, such that the maximum distance of any point in V to its assigned center in F is minimized. When incorporating pairwise constraints in this model, we assume that they are only defined between points of V and again their definition is the same as in k-center with stochastic pairwise constraints.

No-k-supplier with stochastic pairwise constraints. This problem is exactly the same as k-supplier with stochastic pairwise constraints, but this time we have no cardinality constraint on the set C.

No-k-median with stochastic pairwise constraints. This variant differs

from no-k-center in that here we are trying to and minimize the average distance of points to their assigned centers and not the maximum radius.

4.1.2 Motivations

Since we are introducing some new problems and variants, we take a moment here to elaborate on the motivations for these problems.

Fairness. In the area of fairness, we wish to avoid clusterings which perpetuate biases in society. Here, points may be job candidates clustered by a recruitment service to determine which job to advertise to them. The *aware* approach to fair classification introduced in the seminal work of [51] assumes that we have access to an additional metric, separate from the feature space, which captures the true "similarity" between points (or some approximation of it). This similarity metric may be quite different from the feature space (e.g., due to redundant encodings of features such as race) and they argue for the notion of "treating similar candidates similarly." In this setting, our framework can guarantee that the probability of any two points u and v being separated into different clusters is bounded by a function of their similarity. In addition, the ability to accept arbitrary probabilities lets us tune this function based on our confidence in the similarity metric.

Semi-supervised learning. Another common example of must-link and/or cannot-link constraints is in the area of semi-supervised learning [52, 53]. Here, we may assume some pairs of points have been annotated (e.g., by human experts) with additional information about their similarity [54] or some data points may

be labeled [55, 56] allowing pairwise relationships to be inferred. We then have to incorporate those extra requirements in our algorithmic setting. In many real-world applications, must-link and cannot-link constraints will be included together. However, we consider them separately here to show that must-link constraints alone are more tractable and admit algorithms with worst case guarantees.[1] In addition, we explore soft must-link constraints where the labeler generating the constraints is assumed to make some bounded number of errors and our model allows for multiple labelers with differing accuracies (e.g., from crowdsourcing labels) [57, 58].

OTU Clustering. The field of metagenomics involves analyzing environmental samples of genetic material to explore the vast array of bacteria that cannot be analyzed through traditional culturing approaches. A common practice in the study of these microbial communities that we explore in detail in Section 6.1 is the *de novo* clustering of genetic sequences (e.g., 16S rRNA marker gene sequences) into Operational Taxonomic Units (OTUs) [59, 60] that ideally correspond to clusters of closely related organisms. One of the most ubiquitous approaches to this problem involves taking a fixed radius (e.g., 97% similarity based on string alignment [61]) and outputting a set of center sequences, such that all points are assigned to a center within the given radius [59, 62]. In this case, we do not know the number of clusters a priori, but we may be able to generate many pairwise constraints based on a distance/similarity threshold as in [60] or reference databases of known sequences. Thus, the "no-k" variant of our problem is appropriate for this setting where the

[1] We refer the reader to Section 3.6 of [52] for a conjecture that must-link constraints may be more valuable than cannot-link constraints.

number k should be discovered, but radius and pairwise information is known or estimated. Other work in this area has considered *conspecific probability*, a given probability that two different sequences belong to the same species (easily translated to a BSP) and *adverse triplets*, sets of must-link constraints that cannot all be satisfied simultaneously (an appropriate scenario for soft must-links) [63].

Event planning. When planning multiple events serving a large geographic area, pairwise constraints naturally supplement the commonly used k-center or k-supplier objectives. For example, a dating service hosting speed dating events may choose event locations to minimize client travel distance while assigning pairs of people who are predicted to be a good match to the same event. In this case, the compatibility of two people could reasonably be encoded as either BSP or must-link constraints depending on our goals. In other cases, pairwise constraints may be added between points which are nearby each other in the distance metric to facilitate carpooling or busing [64].

Security placement. In security applications such as Stackelberg Security Games (SSGs), a defender allocates resources so as to protect targets from an adersary. Recent work in security games [65–67] and green security games [68] extends traditionally-discrete models into continuous action spaces for the defenders and/or attackers. In such applications, one resource might defend a certain radius, and for points corresponding to past attacks by the same adversary (in the case of security games) or noisy readings of an animal's GPS tracker over time (in the case of green security games), it may be beneficial to protect/cover them using the same resource. This can easily be expressed in terms of BSPs or hard must-link constraints.

Lane finding. Another example of must-link constraints is the lane finding problem [69,70], where we refine digital maps to the lane level using data from GPS receivers in cars. The goal is to learn the lane boundaries, essentially assigning each data point to a lane cluster. In this case, must-link constraints can arise from the data rather than from some external labeler. A common assumption is that cars usually stay in one lane. Thus, two sequential data points transmitted from the same car can be assigned a must-link constraint. While some work treats this as a hard constraint, assuming no lane changes [69] or that lane changes are labeled [70], our model allows for soft constraints wherein each car is allowed some bounded number of lane changes (i.e., violated must-links between sequential pairs). We further note that while some prior work uses the k-means objective (with centers being lines as opposed to points), radius-based clustering is a natural fit due to the fixed width of lanes.

Network Design. Router placement is another area where minimizing the radius can be balanced against pairwise constraints. Power level of transmission is related to cluster radius and it is desirable to keep it small, especially if routers are in remote areas running on battery. At the same time, the routers themselves could communicate using a low bandwidth network increasing latency for two points that are not assigned to the same router. If we have identified pairs of points which communicate frequently, then we seek a clustering which aims to minimize the radius while assigning as many high frequency pairs as possible to the same cluster.

Combining different types of pairwise constraints. While prior work on the examples above typically uses only one of the constraint types considered here

(sometimes combined with cannot-link constraints in heuristic approaches), we note that it is natural and useful to combine them in many cases. For example, in the fairness setting discussed above, we may generate must-link constraints from some labeling of points while incorporating bounded separation probabilities to guarantee that we respect the similarity metric.

4.1.3 Contributions to Radius-based Constrained Clustering

Here, we summarize the contributions for these problems the will be described in detail in this document. All of the results in Sections 4.1.4 through 4.1.8 are from an unpublished manuscript that is joint work with John P. Dickerson, Samir Khuller, Aravind Srinivasan, and Leonidas Tsepenekas. We describe three major areas of contribution below and briefly mention additional results in the next section.

BSP and soft must-link constraints. The first element of our contribution is the introduction of the novel BSP and soft must-link constraints to the problems defined in Section 4.1.1. These types of constraints model natural applications and demand randomized clusterings which turns out to be more challenging compared to the standard deterministic requirements found in the literature. The authors in [71] also consider a separation constraint, where the separation is deterministic and depends on the underlying distance between two points. Our BSP constraints allow room for more flexible solutions due to their stochastic nature, and also capture more general separation scenarios, since the BSP values can be arbitrarily chosen.

General framework for approximation algorithms. Regarding technical

results, we provide a summary in Table 4.1 of our bicriteria approximations. The first element of each tuple is the approximation ratio achieved for the clustering objective, while the second is the ratio by which we may violate the pairwise constraints under consideration (e.g., achieve a separation probability of $2p_e$ instead of p_e).

In this chapter, we will only provide details of the following two main results. We design an algorithm for the k-center problem with stochastic pairwise constraints which achieves a 3-approximation on the optimal radius, a 2-approximation on preserving BSP constraints, a 2-approximation in expectation for soft must-link constraints, and exactly satisfies all hard must-link constraints. Moreover, we design an algorithm for the no-k-center problem with stochastic pairwise constraints which has exactly the same guarantees as the one described above, but with the ratio for the radius being 2. These algorithms take the form of a general framework using linear programming and rounding techniques inspired by [6]. A particularly interesting point is that this framework can simultaneously handle all of these types of pairwise constraints without individual care for each specific one.

The no-k-center problem and hardness proof. As stated above, we introduce the no-k-center problem and give an approximation algorithm for it. A natural question is whether this problem is even hard, and we prove that it is in fact NP-hard.

Variation	Approximation ratio (max radius, constraints)
k-center with stochastic pairwise constraints	(3, 2)
No-k-center with stochastic pairwise constraints	(2, 2)
k-center with hard must-link constraints	(2,1)
k-median with hard must-link constraints	(9.83, 1)
k-center with generalized cannot-link constraints	Not approximable
Knapsack-center with stochastic pairwise constraints	(4, 2)
Matroid-center with stochastic pairwise constraints	(4, 2)
k-supplier with stochastic pairwise constraints	(4, 2)
No-k-supplier with stochastic pairwise constraints	(1, 2)
No-k-median with stochastic pairwise constraints	(2, 2)

Table 4.1: Summary of upper bounds.

4.1.4 Summary of Additional Contributions

The following are some additional contributions to these problems, the details of which are omitted from this document.

k-center and k-median with only hard must-link constraints. Must-link and cannot-link constraints have been studied extensively in the semi-supervised learning literature for a variety of objectives [52], but we study them purely from a Combinatorial Optimization perspective. The work of [71] provides a $(1+\epsilon)$ approximation for the must-link case, but only in the restricted $k = 2$ setting. We are able to show that for k-center with only hard must-link constraints, we can achieve a 2-approximation. This is tight due to the known hardness of k-center. We are also the first to show a constant-factor approximation for k-median with only hard must-link constraints.

Hardness of cannot-link constraints. The authors in [72] prove that even

approximating a solution to k-center that satisfies a set of given cannot-link constraints is NP-hard. We significantly generalize this result, by introducing the function f that penalizes the violation of cannot-link constraints and show that k-center with generalized cannot-link constraints cannot be approximated at all, unless $P = NP$.

Extensions of our framework to other variants. There are many natural extensions of our algorithmic results to related problems including k-supplier, no-k-supplier, knapsack-center, and matroid-center. We are also able to adapt the algorithm for the no-k version to minimize the assignment cost as in the k-median problem. These algorithms result from simple modifications to the algorithms of Sections 4.1.7 and 4.1.8.

4.1.5 Further Related Work

Besides the related work mentioned in the previous subsection, there is also a long line of research involving similar problems to what we study here.

For the classical formulation of the k-center problem a 2-approximation has been known for several decades and this is the best possible assuming $P \neq NP$ [73–75]. The k-median problem is also well-studied and understood. The first constant approximation for it was given in [76], and the currently best known is 2.611 by [77].

The knapsack-center and the k-supplier problems can both be approximated within a factor of 3 [73–75]. These results are also the best achievable unless $P \neq NP$ [75, 78]. For the matroid-center problem, there exists a 3-approximation due

to [79].

The work of [80] considers a variety of problems that are similar in flavor to the ones studied here since they involve classical clustering combined with probabilistic constraints. Namely, they study variations of k-center with outliers, but with each point having a a probability $p_u \in [0, 1]$ of being covered in the solution. Then any algorithm should come up with a randomized assignment that respects these probabilities and minimizes the radius.

We also develop connections to the uniform-metric-labeling problem introduced by [6]. In our work, the labels are centers and the cost of assigning a point to a center/label is the distance between them. However, there are a few major differences between the problems: 1) we must choose locations for the centers/labels which determine the assignment cost; 2) our goal is to minimize the maximum assignment cost as opposed to the sum; and 3) we have a bound on the separation probabilities for pairs of points which must be preserved locally for each pair. The authors in [81] discuss the relation between uniform metric labeling and satisfying separation-probabilities constraints. However, having the labels being points—and also having to choose the centers/labels, as in our problem—poses many additional technical difficulties.

4.1.6 A Useful Subroutine

Here, we mention a rounding procedure by [6] that we use in our results. Consider a set of elements V, a set of labels L, a set of tuples $E \subseteq V \times V$, and the

following LP.

$$\sum_{l \in L} x_{l,v} = 1 \qquad \forall v \in V$$

$$z_{e,l} \geq x_{l,v} - x_{l,w} \qquad \forall e = \{v, w\} \in E,\ \forall l \in L$$

$$z_{e,l} \geq x_{l,w} - x_{l,v} \qquad \forall e = \{v, w\} \in E,\ \forall u \in C^{\star}$$

$$z_e = \frac{1}{2} \sum_{l \in L} z_{e,l} \qquad \forall e = \{v, w\} \in E$$

$$0 \leq x_{u,v}, z_e, z_{e,l} \leq 1 \qquad \forall u, v \in V, \forall e \in E, \forall l \in L$$

Suppose now that we have a fractional solution to the above LP. Using only the x values, the authors in [6] provide the following rounding procedure.

Algorithm 4: KT-Round [6]

```
while there exists some v ∈ V that is not integrally assigned to any label l ∈ L do
    Pick a label l ∈ L uniformly at random with probability 1/|L|.
    Pick a value y ∈ [0, 1] uniformly at random.
    For each v ∈ V that has not been assigned yet, assign it to l if y < x_{l,v}.
```

Theorem 12 (From [6])**.** *In polynomial expected time,* ***KT-Round*** *assigns each* $v \in V$ *to a label* $l_v \in L$*, such that* $\forall e = \{u, v\} \in E : Pr[l_u \neq l_v] \leq 2z_e$ *and* $\forall v \in V,\ \forall l \in L : Pr[l_v = l] = x_{l,v}$*.*

4.1.7 k-center with Stochastic Pairwise Constraints

We now present our result regarding k-center with multiple types of stochastic pairwise constraints. Our algorithm works in two stages. In the first, it selects a set of centers using a standard algorithm for k-center. After that, we need to come

up with a randomized assignment that respects all of our constraints. To achieve this, we utilize an assignment LP that closely resembles that of Section 4.1.6. To give some further intuition, the chosen centers of the first phase will serve as labels, and this will guarantee that all assignments made will result in a maximum radius that is within a small constant factor of the optimal. The main technical difficulty lies in proving that the LP we introduce, which uses the approximate centers as labels, yields a non-empty polytope whose fractional solutions can be rounded by the procedure of Section 4.1.6.

To begin with, observe that the optimal radius $R^\star$ must be the distance between two points, and so by performing a binary search on all of the possible $O(n^2)$ values we can always guess it in polynomial time. Therefore, assume we have guessed $R^\star$ correctly. Based on this guess, the first stage of the algorithm uses a standard approach for picking a set of centers C. All points are initially thought of as uncovered, and $C \leftarrow \emptyset$. While there still exists some point $u \in V$ that is uncovered, we choose u as a center and include it in C, i.e. $C \leftarrow C \cup \{u\}$. Also, all points within radius $2R^\star$ around u are now considered covered. Notice that this process opens at most k centers. Because $R^\star$ is the optimal radius, the points that become covered when we choose a center point u include all points that would appear in its cluster in any optimal solution. Therefore, since $|C^\star| \leq k$, k iterations suffice to cover all elements.

Given the set C we write the following LP which will guide us toward an as-

signment function ϕ that would satisfy the pairwise constraints. Let $P' = P \bigcup_{S_i \in S} S_i$.

$$\sum_{u \in C} x_{u,v} = 1 \qquad \forall v \in V \tag{4.1}$$

$$z_{e,u} \geq x_{u,v} - x_{u,w} \quad \forall e = \{v, w\} \in P',\ \forall u \in C \tag{4.2}$$

$$z_{e,u} \geq x_{u,w} - x_{u,v} \quad \forall e = \{v, w\} \in P',\ \forall u \in C \tag{4.3}$$

$$z_e = \frac{1}{2} \sum_{i \in C} z_{e,u} \qquad \forall e = \{v, w\} \in P' \tag{4.4}$$

$$z_e \leq p_e \qquad \forall e = \{v, w\} \in P' \tag{4.5}$$

$$\sum_{e \in S_i} z_e \leq \psi_i |S_i| \qquad \forall S_i \in S \tag{4.6}$$

$$x_{u,v} = 0 \qquad \text{if } d(u, v) > 3R^\star \tag{4.7}$$

$$x_{u,u} = 1 \qquad \forall u \in C \tag{4.8}$$

$$0 \leq x_{u,v} \leq 1 \qquad \forall u, v \in V \tag{4.9}$$

The variable $x_{u,v}$ can be interpreted as the probability of assigning point v to center $u \in C$. To understand the meaning of the z variables, it is easier to think of the integral setting, where $x_{u,v} = 1$ iff v is assigned to u and is 0 otherwise. In this case, $z_{e,u}$ is 1 for $e = \{v, w\}$ iff exactly one of v and w are assigned to u. Thus, z_e is 1 iff v and w are separated. We will eventually show that in the fractional setting z_e is a lower bound on the probability that v and w are separated. Constraint (4.1) simply states that every point must be assigned to a center. Given the previous discussion, constraints (4.5) and (4.6) express the pairwise constraints. Constraint (4.7) states that every point can only be assigned to centers that are within a distance of $3R^\star$

from it, and constraint (4.8) ensures that each center should be assigned to its own cluster. One final detail is that constraints (4.2), (4.3), and (4.4) are defined for *distinct* v, w. Since it is not by any means trivial that the above LP has a feasible solution, we need to prove that the polytope it defines is non-empty.

Lemma 8. *The polytope defined by LP (4.1)-(4.9) is non-empty.*

Proof. For the sake of the analysis, consider the following construction. Let C be the set of centers obtained from the first stage and $C^\star$ be the set of centers in the optimal solution. We define a set C_u for each $u \in C$ as follows. Initially, add to C_u all $u' \in C^\star$ such that $d(u, u') \leq R^\star$. In this phase, a $u' \in C^\star$ cannot be included in more than one C_u. Remember that for any distinct $u, u'' \in C$ we have $d(u, u'') > 2R^\star$. Thus, if we had $d(u, u') \leq R^\star$ and $d(u'', u') \leq R^\star$, the triangle inequality would be violated. After this adding phase there may be some centers of $C^\star$ that are not yet added to any C_u. Then add $u' \in C^\star$ to C_u if u was the first to cover u' during the the first stage of our algorithm, which means $d(u, u') \leq 2R^\star$. Notice that the sets C_u induce a partition of $C^\star$. Also, let $\phi^\star : V \mapsto C^\star$ be the optimal randomized assignment function. Now set:

$$x_{u,v} := \sum_{u' \in C_u} Pr[\phi^\star(v) = u']$$

We will prove that this solution satisfies all the LP constraints.

Constraint (4.1): For every $v \in V$ we have

$$\begin{aligned}\sum_{u \in C} x_{u,v} &= \sum_{u \in C} \sum_{u' \in C_u} Pr[\phi^\star(v) = u'] \\ &= \sum_{u' \in C^\star} Pr[\phi^\star(v) = u'] \\ &= 1\end{aligned}$$

The second line follows from the fact that the C_u sets induce a partition of $C^\star$. The last equality follows from $C^\star$ and $\phi^\star$ being parts of the optimal solution.

Constraint (4.9) holds trivially because of constraint (4.1).

Constraint (4.7): We would like to show that $x_{u,v} = 0$ if $d(u,v) > 3R^\star$. From the optimal solution, we know that $Pr[d(v, \phi^\star(v)) \leq R^\star] = 1$. Now, take a $v \in V$ and $u \in C$ such that $d(u,v) > 3R^\star$. Suppose that $x_{u,v} = \sum_{u' \in C_u} Pr[\phi^\star(v) = u'] > 0$. This implies that there exists a center of $C^\star$, $u' \in C_u$ such that $d(v,u') \leq R^\star$. However, we know from the way we constructed C_u, that $d(u,u') \leq 2R^\star$. This violates the triangle inequality. So $x_{u,v}$ must be 0.

Constraint (4.8): We have that $x_{u,u} = \sum_{u' \in C_u} Pr[\phi^\star(u) = u']$. By the way we constructed C_u, we know that all of the centers $u' \in C^\star$ such that $Pr[\phi^\star(u) = u'] > 0$ are included in C_u. These are exactly the centers for which $d(u,u') \leq R^\star$. Therefore, $x_{u,u} = 1$.

Constraints (4.2)-(4.5): First of all, for every $e = \{v, w\}$ and $u \in C$ set $z_{e,u} := |x_{u,v} - x_{u,w}|$. This immediately satisfies constraints (4.2) and (4.3). For that assignment we have:

$$\begin{aligned} z_{e,u} &= \Big| \sum_{u' \in C_u} Pr[\phi^\star(v) = u'] - \sum_{u' \in C_u} Pr[\phi^\star(w) = u'] \Big| \\ &= \Big| \sum_{u' \in C_u} \big(Pr[\phi^\star(v) = u'] - Pr[\phi^\star(w) = u'] \big) \Big| \\ &\leq \sum_{u' \in C_u} \Big| Pr[\phi^\star(v) = u'] - Pr[\phi^\star(w) = u'] \Big| \end{aligned}$$

Therefore, we can easily upper bound z_e as follows:

$$\begin{aligned} z_e &= \frac{1}{2} \sum_{u \in C} z_{e,u} \\ &\leq \frac{1}{2} \sum_{u \in C} \sum_{u' \in C_u} \Big| Pr[\phi^\star(v) = u'] - Pr[\phi^\star(w) = u'] \Big| \\ &\leq \frac{1}{2} \sum_{u' \in C^\star} \Big| Pr[\phi^\star(v) = u'] - Pr[\phi^\star(w) = u'] \Big| \end{aligned} \tag{4.10}$$

The last line follows from the fact that the C_u sets induce a partition of $C^\star$. Now notice that:

$$\begin{aligned} Pr[\phi^\star(v) = \phi^\star(w)] &= \sum_{u \in C^\star} Pr[\phi^\star(v) = u \ \wedge \ \phi^\star(w) = u] \\ &\leq \sum_{u \in C^\star} \min\{ Pr[\phi^\star(v) = u], \ Pr[\phi^\star(w) = u] \} \end{aligned} \tag{4.11}$$

To relate (4.10) and (4.11) consider the following trick.

$$\sum_{u \in C^\star} \min\{Pr[\phi^\star(v) = u],\ Pr[\phi^\star(w) = u]\} + \frac{1}{2} \sum_{u \in C^\star} |Pr[\phi^\star(v) = u] - Pr[\phi^\star(w) = u]|$$
$$= \sum_{u \in C^\star} \Big(\min\{Pr[\phi^\star(v) = u],\ Pr[\phi^\star(w) = u]\} + \frac{|Pr[\phi^\star(v) = u] - Pr[\phi^\star(w) = u]|}{2} \Big)$$
$$= \sum_{u \in C^\star} \frac{Pr[\phi^\star(v) = u] + Pr[\phi^\star(w) = u]}{2} = 2/2 = 1 \tag{4.12}$$

Finally, combining (4.10), (4.11), and (4.12) we get:

$$z_e \leq 1 - \sum_{u \in C^\star} \Big(\min\{Pr[\phi^\star(v) = u],\ Pr[\phi^\star(w) = u]\} \Big) \leq Pr[\phi^\star(v) \neq \phi^\star(w)] \leq p_e \tag{4.13}$$

The final inequality follows since $\phi^\star$ satisfies the stochastic constraints.

Constraint (4.6): For every $S_i \in S$ we have:

$$\sum_{e \in S_i} z_e \leq \sum_{\{u,v\} \in S_i} Pr[\phi^\star(v) \neq \phi^\star(w)] \leq \psi_i |S_i|$$

The first inequality is due to (4.13), while the second results again from $\phi^\star$ being a feasible assignment. □

Based on Lemma 8 we can find a fractional solution to LP (4.1)-(4.9). Observe that this solution satisfies constraints (4.1)-(4.4), which are necessary for the rounding procedure of Section 4.1.6 to be applied, using V as the set of elements, C as the set of labels, and P' as E. Thus, utilizing this rounding algorithm, we get

an integral assignment, for which the probability that $\{v, w\} \in P'$ are assigned to different centers is at most $2z_e$. It is clear that in our problem, this solution violates the separation probability constraints by a factor of 2, due to (4.5), and the soft must-link constraints again by a factor of 2, this time due to (4.6) and the linearity of expectation. Moreover, (4.7) and Theorem 12 guarantee that no point is assigned to a center more than $3R^\star$ away from it. Also, since $x_{u',u} = 0$ and $x_{u,u} = 1$ for all $u, u' \in C$ with $u \neq u'$, a center can only be assigned to its own cluster, because of line 4 of the rounding process. We conclude with a final remark regarding the feasibility of the returned solution. Observe that any two points $u, u' \in C$ are at distance strictly greater than $2R^\star$. This implies that $Pr[\phi^\star(u) \neq \phi^\star(u')] = 1$, and hence our algorithm clusters the pair $\{u, u'\}$ in the same manner as the optimal solution. In other words, we are not required, under any circumstances, to cluster u and u' together, and hence having them in different clusters in our solution does not affect feasibility. All the above discussion leads to the following.

Theorem 13. *There exists a polynomial time algorithm for the k-center problem with stochastic pairwise constraints which achieves a 3-approximation on the optimal radius, a 2-approximation on preserving BSP constraints, and a 2-approximation in expectation for the soft must-link constraints.*

Observation 14. *Note that hard must-link constraints can be easily incorporated in this framework, since they can be encoded as $p_e = 0$ (i.e., zero probability of separating the points). This implies that the above algorithm can simultaneously combine BSPs, soft must-link, and hard must-link constraints. Moreover, since our*

guarantee on BSPs is $2z_e$, the hard must-link constraints encoded as stated above, will be satisfied exactly.

4.1.8 No-k-center with Stochastic Pairwise Constraints

Here, we address the variant with no constraint on the number of centers, as defined in Section 4.1.1. At first glance, this problem may seem trivial, since in the standard unconstrained setting, having no cardinality requirement allows one to use all points as centers and thus achieve a maximum radius of 0. However, the introduction of stochastic pairwise constraints adds significant complexity to the problem, ruling out such trivial solutions.

No-k-center Hardness

Theorem 15. *No-k-center with stochastic pairwise constraints is NP-hard.*

Proof. We show the hardness of the no-k problem via a reduction from the minimum k-cut problem where we are given a graph $G = (V, E)$ with a subset of k vertices in V that we wish to separate. We seek a cut of size at most γ that separates these k vertices.

We create a set of points P that consists of three disjoint subsets: P_k, P_{not-k}, and $P_{satellites}$. The set P_k contains a point corresponding to each of the k vertices that we wish to separate in the original minimum k-cut graph. The set P_{not-k} contains a point for each of the other vertices from the original problem. The set $P_{satellite}$ is a set of "dummy" points which will contain one point for each point in

P_k and will force the points in P_k to be added to separate clusters.

We now define the metric space. All of the points in P_{not-k} are co-located. The points in P_k are each at distance greater than R from each other and distance exactly R from all of the points in P_{not-k}. Each point in $P_{satellites}$ has a single neighbor point in P_k and is at distance R from its neighbor point, distance greater than $2R$ from every other point in P_k, and distance exactly $2R$ from the points in P_{not-k}. For constraints, we have a hard must-link constraint between each point in $P_{satellites}$ and its neighbor in P_k. For each edge $e \in E$ from the min cut problem, we form a soft must-link constraint between the corresponding points in the union $P_k \cup P_{not-k}$ and add it to the same set S. Finally, the fraction of S we are allowed to violate would be $\gamma/|E|$.

Suppose now that no-k-center with stochastic pairwise constraints can be solved in polynomial time. For the decision version of the problem this implies that we can get a yes or no answer when asking if there is a solution for a given target radius R. The instance we constructed above has a solution of radius R iff the original min-cut problem has a solution of value at most γ. To see this observe that although soft constraints are only preserved in expectation, an algorithm which violates at most $\gamma/|E|$ constraints in expectation implies that there exists a solution which violates at most $\gamma/|E|$ constraints deterministically. $\square$

Approximation Algorithm for No-k Problems

Our algorithm consists of two stages, but differs from that of Section 4.1.7 in that it does not start by finding centers for the unconstrained problem. The first stage closely resembles the second stage of the algorithm of Section 4.1.7, since it uses a similar assignment LP. In fact, a key idea behind our approach is not choosing any centers prior to solving the LP, but instead allowing each point to be assigned to any other nearby point. In other words, the label set L is simply V. After that, we again use the rounding from Section 4.1.6. However, we may have to resolve the issue of points assigned to centers, but also being centers themselves with other points assigned to them. To do that, we perform a reassignment step, that slightly increases the maximum radius of the constructed clustering.

We begin with the assignment LP, assuming once more that we obtained the optimal radius $R^\star$ via binary search. There are three changes to the LP from Section 4.1.7. First, the set of centers is now the entire point set (i.e. $C = V$). Second, constraint (4.8) which was stated as $x_{u,u} = 1 \quad \forall u \in C$ is no longer used because we cannot force every point to be assigned to itself. Instead, we replace it with $x_{u,u} \geq x_{u,v} \quad \forall u \in V, \forall v \in V$. In other words, the fraction of a point u which is assigned to itself must be greater than the fraction of any other point v assigned to u. The third change is that we have $R^\star$ in constraint (4.7).

$$\sum_{u \in V} x_{u,v} = 1 \qquad \forall v \in V \tag{4.14}$$

$$z_{e,u} \geq x_{u,v} - x_{u,w} \quad \forall e = \{v, w\} \in P',\ \forall u \in V \tag{4.15}$$

$$z_{e,u} \geq x_{u,w} - x_{u,v} \quad \forall e = \{v, w\} \in P',\ \forall u \in V \tag{4.16}$$

$$z_e = \frac{1}{2} \sum_{i \in V} z_{e,u} \qquad \forall e = \{v, w\} \in P' \tag{4.17}$$

$$z_e \leq p_e \qquad \forall e = \{v, w\} \in P' \tag{4.18}$$

$$\sum_{e \in S_i} z_e \leq \psi_i |S_i| \qquad \forall S_i \in S \tag{4.19}$$

$$x_{u,v} = 0 \qquad \text{if } d(u, v) > R^\star \tag{4.20}$$

$$x_{u,u} \geq x_{u,v} \qquad \forall u, v \in V \tag{4.21}$$

$$0 \leq x_{u,v} \leq 1 \qquad \forall u, v \in V \tag{4.22}$$

Lemma 9. *LP (4.14)-(4.22) defines a non-empty polytope.*

Proof. The proof is very similar to that of Lemma 8. Suppose that we know the set of centers $C^\star$, used in the optimal solution, corresponding to $R^\star$. Moreover, we know that there exists a randomized assignment $\phi^\star : V \mapsto C^\star$, that satisfies the requirements in the problem's definition. Let $x_{u,v} := Pr[\phi^\star(v) = u]$, where the probabilities are those of the optimal assignment. Since every point must be assigned to a center in $C^\star$ under $\phi^\star$, constraint (4.14) holds. Also, because every point is assigned to a center at a distance at most $R^\star$ away, constraints (4.20) also hold trivially. Now we are going to show that constraints (4.15)-(4.18) hold and

z_e is again a lower bound on the probability that the endpoints of $e = \{v, w\}$ are separated. Set $z_{e,u} = |x_{u,v} - x_{u,w}|$. This immediately satisfies constraints (4.15) and (4.16). Using the same reasoning as in the proof of Theorem 8 we get the two following inequalities:

$$z_e \leq 1 - \sum_{u \in C^\star} \Big(\min\{Pr[\phi^\star(v) = u],\ Pr[\phi^\star(w) = u]\} \Big) \leq Pr[\phi^\star(v) \neq \phi^\star(w)] \leq p_e$$

$$\sum_{e \in S_i} z_e \leq \sum_{\{u,v\} \in S_i} Pr[\phi^\star(v) \neq \phi^\star(w)] \leq \psi_i |S_i|$$

To conclude we need to prove that constraint (4.21) is also satisfied. We proceed with a case analysis. At first suppose $u \in C^\star$. This means that $x_{u,u} = Pr[\phi^\star(u) = u] = 1$, since u is a center and it is always assigned to its own cluster. Therefore, the constraint is obviously satisfied for every $u \in C^\star$ and $v \in V$. Then assume $u \notin C^\star$. This implies that in the optimal solution no point will get assigned to u, since it is not a center. Therefore, for all $v \in V$ we have $Pr[\phi^\star(v) = u] = 0$, and hence $x_{u,v} = 0$. This implies that constraint (4.21) is satisfied in this case as well. □

After solving the LP, we round the LP solution as in Section 4.1.7, with the only difference that this time the label set L is V, enabling us to utilize all points as possible centers. As we have seen before, this produces a random integral solution which violates the pairwise constraints by a factor of 2, but this time the maximum distance between a point and its assigned center is at most $R^\star$ as mandated by the LP. However, due to replacing constraint (4.8), some u may be chosen as a center

with many points assigned to it, while u itself is assigned to some other point u'. It may even be the case that u' gets assigned to u'', and so on in a long chain. This situation violates the requirement that each chosen center should be assigned to its own cluster. To address this, we add a final reassignment step after rounding to guarantee that points are assigned to an open center that is assigned to itself. Throughout this reassignment process, we can't separate any points that have been assigned together because that could increase separation probabilities.

Let $G = (V, E)$ be the directed assignment graph after rounding where an edge $e = (u, v) \in E$ denotes that point v ended up assigned to point u and $u \neq v$. If u is assigned to itself, then there is an out-going edge from u to itself. Before we present the actual reassignment step, we need to prove that G is a forest of directed trees with each vertex having out-degree equal to 1.

Lemma 10. *G is a forest of directed trees with each vertex having out-degree equal to 1.*

Proof. To see that each vertex has out degree equal to 1, note that each point is assigned to exactly one center in the rounding procedure (possibly itself). To see that G contains only trees, suppose for the sake contradiction that G has a directed cycle. Consider the first vertex u in the cycle to be sampled by step 2 of the rounding procedure and have vertices assigned to it in step 4. If u wasn't already assigned to some other center w, then due to constraint (4.21) u is assigned to itself and hence the cycle can never close. On the other hand, if u is already assigned to some w, then w cannot participate in the cycle. This is because u is the first vertex of the

cycle to be chosen as a center. Therefore, since u has at most one out-going edge, the cycle cannot close. In either case we reach a contradiction. □

Now, observe that for each directed tree T of G, we can take a vertex with out-degree 0 as the root (i.e., a center correctly assigned to itself). Let L be the set of leaves, which contains vertices that have in-degree 0. For each vertex v, let $parent(v)$ be the vertex such that there exists an edge from v to $parent(v)$ and let $children(v) = \{w \in V \mid (w, v) \in E\}$. The reassignment step then works for each tree T, bottom up. For every vertex v that is not the root, consider $children(v)$. Choose any $w \in children(v)$ and make that a center. Afterwards, assign every point in $children(v)$ to w. Keep going until you only have the root and its children. At that point you are done, and no further reassignment is needed. The invariant this step should satisfy is that if two points are assigned to the same center by the rounding algorithm, then after the reassignment they should also be assigned to the same center, even if that is different than their previous common center.

Lemma 11. *The reassignment step assigns every point to an open center without separating any points which were clustered together or extending the radius by more than a factor of* 2.

Proof. By definition we always reassign points to a member of their initial cluster, and thus never separate vertices which were assigned to the same center/cluster. In each iteration, we simply move a center to a new point which is a member of the cluster. To see that the radius increases to at most $2R^\star$, note that every edge in G is a distance of at most $R^\star$ in the underlying metric space. Each vertex is assigned

to a center at most $R^\star$ away initially, and then the center is moved an additional distance of $R^\star$. □

All the above discussion leads to the following.

Theorem 16. *There exists a polynomial time algorithm for the no-k-center problem with stochastic pairwise constraints which achieves a 2-approximation on the optimal radius, a 2-approximation on preserving BSP constraints, and a 2-approximation in expectation for the soft must-link constraints.*

4.2 Pairwise Fair and Community-preserving k-Center Clustering

We now turn our attention to a special case of constrained k-center clustering where the additional goal is to satisfy two new definitions of fairness that we introduce. This section is joint work with Darshan Chakrabarti, John P. Dickerson, Samir Khuller, Aravind Srinivasan, and Leonidas Tsepenekas [82].

As discussed in the previous section, clustering is one of the foundational problems in unsupervised learning and operations research. In it, we seek to partition n data points into *clusters* such that points within each cluster are similar according to some distance function. Its numerous applications include document/webpage similarity for search engines [83, 84], targeted advertising including employment opportunities [85], medical imaging [86,87], and various other data mining and machine learning tasks. However, as machine learning has become ubiquitous, concerns have arisen about the "fairness" of many algorithms, especially when the data points represent human beings. In this case, we seek additional guarantees on how people will

be treated beyond the typical goal of pure optimization.

The k-center problem is a fundamental clustering problem. The objective is to select k center points and assign all other points to clusters around them such that the maximum distance from any point to its assigned center is minimized. The problem is NP-hard with the best possible approximation factor being 2 assuming $P \neq NP$ [73, 74]. Fairness for k-center can have many definitions depending on the application. When the points are labeled (e.g., with racial demographics or another protected class), a *group fairness* constraint may require clusters to contain a minimum amount of diversity among labels [88–90]. However, we consider a different kind of fairness which bounds the probability that nearby points (presumably similar or related) are assigned to different clusters. Our approach can also address issues of discrimination against protected classes, albeit in a different way.

We introduce two new notions of fairness to the k-center clustering problem, *pairwise fairness* and *community-preserving fairness*. A k-center algorithm is α-*pairwise fair* if every pair of points has a probability of at most α of being assigned to different centers, where $\alpha(\cdot)$ is an increasing function of the distance between the two points, and $\alpha(0) = 0$. We define a *community* as any subset of points with arbitrary diameter D and a community is *preserved* if its points are assigned to as few different clusters as possible (ideally one cluster). Communities do not need to be known or explicitly identified. An algorithm is β-community preserving if every community has probability at most β of being partitioned into more than t clusters where β is an increasing function of the community diameter D and a decreasing function of the number of clusters t.

The concept of pairwise fairness is relevant in settings where the points represent people and certain clusters may be preferable to others. We may assume the distance between two points represents some similarity between them and by extension, implies they should be treated similarly (assigned to the same cluster) with some related probability. We are thus being "fair" to each point by treating it like its nearby neighbors. The seminal work of [51] also explores this idea of a "*fairness constraint*," that "similar individuals are treated similarly," but applied to classification and differing from our work as discussed in Section 4.2.2.

Community preservation becomes relevant in settings where the data points are people who gain some benefit from sharing a cluster with their near neighbors. For example, consider the drawing of congressional districts and the practice of gerrymandering which has gained enormous attention and study recently. In a single-member district plurality system (e.g., the US House of Representatives), populations are partitioned into clusters called districts which each elect a single candidate based on a plurality vote. In this setting, a person or political party may draw gerrymandered districts in order to divide a community of people with shared needs, thus weakening or eliminating the power of that community to influence elections. Many cities in the United states demonstrate this phenomenon. Notably, the city of Austin, Texas is distributed among five separate congressional districts while its population is small enough to fit comfortably into two. Although it is the 11th largest city in U.S., Austin residents represent a minority in each of those five districts [91].

The US Supreme Court ruled on racial gerrymandering in Thornburg v. Gin-

gles [92], establishing that communities of people belonging to a racial or language group should not be fractured in order to weaken their vote (subject to very specific criteria). However, partisan gerrymandering was recently ruled not justiciable by that court in Rucho v. Common Cause [7], leaving it up to the voters in individual states to advocate for some fairer approach to districting.

To combat gerrymandering, recent research has explored the use of computational approaches to draw or evaluate congressional districts [93–96], including k-clustering approaches [97]. Like many techniques in machine learning, computational redistricting has the familiar promise of being an impartial arbiter in place of biased or adversarial human decisions. While this promise cannot be overstated, we know from the fairness literature that additional fairness constraints are often necessary. An algorithmic redistricting approach may claim to be unbiased because it does not use sensitive features such as party affiliation. However, these sensitive features may be redundantly encoded in other features as in the case of party affiliation correlating with population density in the US. Figure 4.1 gives a simple example of how a community can be deterministically separated by k-clustering using the k-center objective.

This notion of preserving communities can also be extended to problems where people are assigned to a group and benefit from having some neighbors assigned to the same group as in the problem of assigning students to public grade schools. For this problem, Ashlagi and Shi [98] incorporated the concept of *community cohesion*, keeping neighborhoods together. They illustrate their point by quoting Boston mayor Menino [99] saying in a 2012 State of the City address, "Pick any street. A

dozen children probably attend a dozen different schools. Parents might not know each other; children might not play together. They can't carpool, or study for the same tests."

Returning to the issue of protected classes, we observe that the community fragmentation imposed by current implementations of school lotteries disproportionately affects members of protected classes. On the other hand, members of more "privileged" classes are more likely to live in a community where assignment is not determined by lottery.

To further elaborate on the school-choice problem, we note that centers need not correspond to physical locations of schools. Many school districts, such as Boston, do not use a model wherein students are always assigned to their nearest school: e.g., a cluster could be a school bus stop for a set of students who will share a bus which is assigned to some school. We refer to [98] for more details.

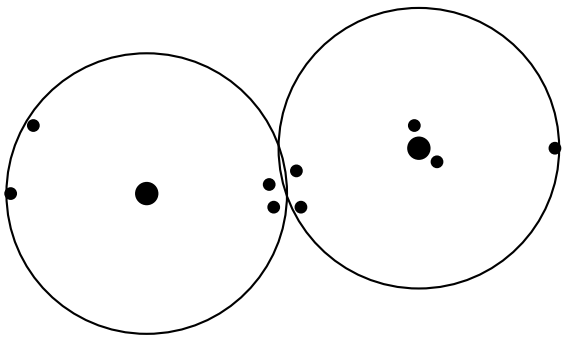


Figure 4.1: An optimal k-center clustering ($k = 2$) with squares denoting the centers. This deterministically separates the community of four nearby points in the middle even though that fractured community has small diameter.

Thus, we see that pairwise fairness and community preservation have broad applications. Even in the apparently benign application of document clustering, we can view a document as its author's voice which could be negatively affected by

an unfair clustering. These fairness constraints can be useful any time we wish to treat nearby points similarly, grant equal access to the strength of a community, or provide protection from efforts to weaken a community.

4.2.1 Definitions and Preliminaries

k-center clustering. In the *classical* (or *unfair*) k-center problem, we are given a set U of n points and a parameter k as input. We assume we can compute some distance function $d(u, v)$ satisfying triangle inequality on any pair of points $u, v \in U$. The objective is to choose k points in U to be *centers* such that we minimize the maximum distance of any point in U to its nearest center. In clustering, each center then defines a cluster. Typically, a point is assigned to its nearest center. However, in fair clustering and other constrained clustering variants, we may assign points to centers other than the nearest one to satisfy other goals.

α-pairwise fairness. We call a k-center algorithm α-pairwise fair if for every pair of points $u, v \in U$, the probability that u and v are assigned to different centers/clusters is at most $\alpha = \alpha(u, v)$ with $\alpha(u, v)$ being an increasing function of $d(u, v)$. In this paper, we give an algorithm for the function $\alpha = d(u, v)/\delta$ where $\delta > 0$ is some distance chosen by the user. As a corollary, we focus on the natural case of $\delta = \psi R$, where R is the optimal radius that can be achieved by an "unfair" algorithm solving the classical k-center problem without fairness constraints and $\psi > 0$ is a user-specified constant. The distance R is used as a natural property of the problem input that can suggest what is "reasonable" to expect. In practice,

δ could be determined by domain knowledge of a specific application. We present an algorithm that achieves $(d(u,v)/\delta)$-pairwise fairness and show that when $\alpha = (d(u,v)/(\psi R))$, the price of fairness is not too bad using both theoretical bounds and experiments.

β-community preserving. We define a *community* as any subset of points with arbitrary diameter D, and a community is *preserved* if its points are assigned to as few different centers/clusters as possible (ideally just one cluster). In our model, communities do not need to be known or explicitly identified as part of the input. An algorithm is β-community preserving if every community has probability at most β of being partitioned into more than t clusters. Here, β is an increasing function of the community diameter D and a decreasing function of t. In our algorithm, every community has probability at most $\beta = (D/\delta)^t$ of being partitioned into more than t clusters, $t \geq 1$, where $\delta > 0$ is some distance chosen by the user (This probability is a decreasing function of t since we may assume $D/\delta < 1$: if $D/\delta \geq 1$, then the probability is trivially at most 1). As with pairwise fairness, we examine the natural choice of $\delta = \psi R$. Here, we show that we can give the guarantee that every community has a probability of at most $\beta = (D/(\psi R))^t$ of being partitioned into more than t clusters. We include t because it captures how fragmented a community becomes more than simply whether or not it has been separated.

Randomization. Both definitions of fairness assume a randomized algorithm and the probabilities discussed are over the randomness in the algorithm. As with some

other fairness problems (e.g., fair allocation of indivisible goods), randomness is essentially required to achieve meaningful gains in fairness. Otherwise, it is easy to construct worst case examples where a fair deterministic algorithm must place all points in one large cluster while a fair randomized algorithm could achieve results close to the unfair optimal. Randomization can even be necessary to meet certain fairness criteria such as the right to a chance to vote in a district with voter distribution similar to a randomly sampled legal district map [100]. We further note that our pairwise fairness definition makes no assumption of independence or correlation between the separation probabilities of different pairs of points. It is an individual guarantee for each pair of points. Consideration of multiple points at once is addressed by the community preservation definition.

Focus on δ as a function of optimal unfair radius R. We consider the special case of δ depending on R in our analysis because R is a reasonable threshold of nearness related to the properties of a given dataset and the k-clustering task at hand. For example, if a community is geographically larger than the optimal unfair clusters themselves, it may be reasonable to partition this community into multiple clusters whereas a small community which can fit easily into a cluster should have some chance of being preserved.

Approximation ratio and price of fairness. The *approximation ratio* of an algorithm for an NP-hard minimization problem like k-center is typically defined as a bound on the ratio of the algorithm's solution to the solution of an optimal

algorithm. The *price of fairness* for a fair variant of a problem is the ratio of the best solution for the fair problem to the best solution for the unfair problem. In our case, the best benchmark we are able to compare our fair algorithm to is the optimal unfair k-center solution. Thus, our approximation ratios simultaneously show a bound on the price of fairness for our proposed fairness definitions. This price of fairness can affect the choice to use a fair algorithm for both practical and legal reasons. From a legal perspective, the disparate impact of an unfair algorithm can be permitted due "business necessity" if the added cost of fairness is too burdensome [101, 102], but a low price of fairness could potentially preclude this defense.

4.2.2 Related Work

There is a long line of work on the classical k-center problem. A 2-approximation is known and is the best possible assuming $P \neq NP$ [73–75]. Followup work has studied many variations of the problem including capacitated [103, 104], connected [105], fault tolerant [104, 106], with outliers [87, 107, 108], and minimum coverage [109]. Other settings include streaming [107, 110, 111], sparse graphs [112], and distributed algorithms for massive data [87]. However, our formulation of pairwise fairness and community preservation, has not been studied.

On the fairness side, our notion of pairwise fairness is partially inspired by [51]. That work focused on binary classification as opposed to clustering and used techniques from differential privacy to achieve fairness guarantees. More specifically, they assume access to a separate similarity metric on the data points and require

similar points to have similar distributions on outcomes. While our model is related, it differs in two crucial ways. First, we do not use (or require) a separate similarity metric. The similarity of two points is defined by the same metric space we are clustering in. Second, we bound the probability that two points are actually *assigned* to the same cluster rather than having similar distributions. This is important for applications in which nearby points derive a benefit from being clustered together or when the meaning of a cluster is not defined prior to the realization of assignments.

For k-center specifically, [88] considered an entirely different "balance" constraint definition of fairness (aka group fairness) wherein each point is given one of two possible labels and each cluster should contain a minimum percent representation of each label. Follow-on work expands their model [113, 114] and addresses concerns in privacy while [115] applied their definition of fairness to spectral clustering. Additional work improved scalability [90] and improved approximation ratios while allowing an unfair solution to be transformed into a fair one [89]. Separately, and motivated by the bias mitigation in data summarization, [116] also looks at a different form of k-center fairness. Zemel et al. [117] address fairness in classification by first transforming the input data into an intermediate representation that balances goodness of representation with removal of certain traits before classification is performed. This first step is a form of clustering with fairness concerns. Finally, there are fair service guarantees for individuals that bound the distance from each point to its nearest center (or facility) [118–120].

Regarding community preservation, [98] observed that assigning students to schools via an independent lottery mechanism fractures communities by sending

neighboring students to different schools. They proposed a correlated lottery algorithm that that maintains the same expected outcomes for individual students while preserving "community cohesion." We note that they define communities by partitioning a city into a grid with each square representing a community, whereas we allow any bounded diameter subset of points to be a community.

Bounding the probability of separating nearby points and similar negative-binomial-type (or discrete exponential) distributions have been used in numerous other settings. Some examples include locality sensitive hashing (LSH) [121–123], randomly shifted grids [124], low diameter graph decompositions [125], and randomized tree embeddings [126,127]. Our work differs from this past work in the modeling of fairness applications and the challenge of balancing fairness with the k-center objective which is not guaranteed in something like LSH. More commonly, an approach like LSH is used to speed up and scale clustering algorithms with approximate near neighbor search or partitioning data for parallel and distributed algorithms.

4.2.3 Our Contributions

In addition to presenting new definitions of fairness in clustering, we show how any algorithm for the k-center problem can be extended to ensure α-pairwise fairness and β-community preservation at the expense of a $\log k$ approximation factor (also price of fairness). We bound our fair algorithm in comparison to the optimal radius achieved in the "unfair" classical k-center problem. There are two reasons for this. One is that the "unfair" optimal serves as the best known lower bound to the fair

optimal. The other is that it captures the *price of fairness*. In other words, it upper bounds the price we must pay in expanding the radius in order to achieve our fairness objectives.

Theorem 17. *There exists an algorithm which finds an $O(\log k)$-approximation to the k-center problem (i.e., the maximum cluster radius is at most $O(R \log k)$) with high probability and such that every pair of points u and v is separated with probability at most $\alpha = d(u,v)/(\psi R)$, where R is the maximum radius obtained by any chosen k-center algorithm and $\psi > 0$ is a user-specified constant.*

The community preserving property in Corollary 2 follows from the pairwise guarantee. A strength of this formulation is that we *do not need to explicitly identify communities* in the data to preserve them with nontrivial probability.

Corollary 2. *There is an efficient $O(\log k)$-approximation algorithm for k-center (i.e., the maximum cluster-radius is at most $O(R \log k)$) with high probability and such that every subset of points with diameter D is partitioned into more than t separate clusters, for any $t \geq 1$, with probability at most $\beta = (D/(\psi R))^t$ where R is the maximum radius obtained by any chosen k-center algorithm. Here, $\psi > 0$ is a user-specified constant.*

For both Theorem 17 and Corollary 2, we note that for some pairs of points (or communities) the value of α (or β) may be greater than 1 and therefore not a valid probability. For these cases, the bound on fairness is trivially true. The constant factors in our big-Oh notation also depend on the constant ψ and our experiments in Section 6.1.10 show that there are not large hidden constants in practice.

Beyond theoretical results, we further explore the algorithm experimentally in Section 6.1.10 on 40 different problem instances of a benchmark dataset to show that it performs as expected or better. On the benchmark problems, we illustrate in Figure 4.2 how tuning a parameter in our algorithm can adjust the trade-off between fairness and minimizing the cluster radius. In Section 4.2.6, we evaluate our algorithm on a real dataset over different target numbers of clusters. The results suggest that our fair approach is not only more fair, but more consistent in its fairness as k varies when compared to a standard "unfair" algorithm. Thus, we can remove the ability of a bad actor to cause unfairness by adjusting the number of clusters k.

While our theoretical and experimental analysis focuses on approximating the radius and fairness, we note that the running time of our proposed algorithm is dependent primarily on the algorithm/heuristic for the initial clustering. Our reassignment algorithm is rather fast with a running time of $O(kn)$. In practice, the running time is dominated by the initial clustering rather than our reassignment algorithm.

4.2.4 The Fair Algorithm

We show how to extend any k-center algorithm to guarantee pairwise fairness at the expense of a larger approximation factor. The idea is to first run an "unfair" k-center algorithm and order the clusters arbitrarily. Then, one-by-one, we expand the radius of each cluster by a value sampled independently from an exponential distribution. Any point which falls within the radii of more than one of these

Algorithm 5: FairAlg

1 **Step 1:** Run any chosen k-center algorithm and order the clusters arbitrarily from 1 to k. Let R be the maximum distance of any point to its center.
2 **Step 2:** Let C_i be a set of points denoting cluster i. Let $c_i \in C_i$ be the center of C_i and R_i be the radius of C_i.
3 **Step 3:** Treat all points including centers as "unclustered" and construct a new set of clusters denoted C'_i.
4 **for** $i = 1$ ***to*** k **do**
5 **4:** Sample an independent random variable x_i from an exponential distribution with parameter $\lambda = 1/(\psi R)$. Let X_i be the realization of that random variable.
6 **5:** Construct cluster C'_i by adding every unclustered point within radius $R_i + X_i$ from original center c_i.
7 **6:** If c_i was unclustered at the start of this iteration designate it as the center c'_i of C'_i. Otherwise, if c_i has been added to a previous cluster C_j, $j < i$, then choose any other previously unclustered point in C'_i to be the center c'_i. If no such point exists, call the cluster empty.

expanded clusters is assigned to the earliest one in the ordering.

We use C_i to refer to the i^{th} cluster found by the initial "unfair" algorithm and c_i to refer to its center. Similarly, we use C'_i to refer to the i^{th} expanded cluster that we will finally output and c'_i to refer to its center. For readability, we also refer to C_i and c_i as original and C'_i and c'_i as final. Let $R_i = \max_{u \in C_i} d(c_i, u)$ be the radius of C_i and $R = \max_i R_i$ be the maximum radius of any cluster found by the original clustering step. Let ψ be any chosen constant greater than 0. The approach is summarized in Algorithm 5.

We note that in the for loop of steps 4 to 6 of Algorithm 5, the centers 1 through k are processed in an arbitrary order. Because of this, our proofs also hold if the center are processed in a random order or some particular order aligned with another side objective.

We first prove that Algorithm 5 achieves α-pairwise fairness for $\alpha = d(u,v)/(\psi R)$. At a high level, the memoryless property of exponentially distributed random variables allows our algorithm to achieve the guarantee in Lemma 12.

Lemma 12. *For any pair of points u and v with distance $d(u,v)$, the probability that Algorithm 5 separates u and v into two separate clusters is at most $d(u,v)/(\psi R)$ where R is the maximum radius obtained by the initial algorithm used in step 1 and $\psi > 0$ is a user-specified constant.*

Proof. For an arbitrary pair of points $u, v \in U$, consider the first iteration i in which at least one of the points is added to a final cluster C_i'. Without loss of generality, let u be the closer point to the original center c_i and note that $d(c_i, v) - d(c_i, u) \leq d(u,v)$ due to triangle inequality. If $d(c_i, v) < R_i$, both points will be added to C_i' regardless of the value of X_i and the probability of separating them is 0. Otherwise, the probability of separating them is the probability that the value $R_i + X_i$ falls between $\max(d(c_i,u), R_i)$ and $d(c_i, v)$ given that $R_i + X_i > d(c_i, u)$.

$$\begin{aligned}
&\Pr[u \text{ and } v \text{ are separated by } C_i' \mid R_i + X_i > d(c_i,u)] \\
&\leq 1 - e^{-\lambda d(u,v)} = 1 - e^{-d(u,v)/\psi R} \\
&\leq \frac{d(u,v)}{\psi R}
\end{aligned}$$

□

We now bound the amount that the radius of any cluster will increase beyond the maximum value R achieved by the original "unfair" algorithm from step 1 of

Algorithm 5.

Lemma 13. *The maximum radius of a cluster found by Algorithm 5 is $O(R \log k)$ with high probability.*

Proof. We start by upper bounding the probability that any cluster C_i' contains a point at distance greater than $O(R \log k)$ from the original center c_i of C_i. This will suffice to prove the lemma for the clusters where $c_i' = c_i$.

$$\begin{aligned}\Pr[\exists X_i > R \log k] &\leq k \Pr[X_i > R \log k] \\ &= k e^{-\lambda R \log k} = k e^{-\log k/\psi} \\ &= k^{1-1/\psi}\end{aligned}$$

Now, suppose c_i was added to some cluster C_j', $j < i$, and could not be chosen as the final center of C_i'. Then the chosen center c_i' of C_i' must be at most $R \log k$ distance from c_i with high probability by the above bound and the fact that X_i and X_j were sampled independently. Thus, by triangle inequality, the radius of such a cluster would be at most $2R \log k = O(R \log k)$ with high probability. □

Lemma 14 extends Lemma 12 to community preservation.

Lemma 14. *For any subset of points S with diameter D, the probability that Algorithm 5 partitions S into more than t separate clusters, $t \geq 1$, is at most $(D/(\psi R))^t$ where R is the maximum radius obtained by the initial algorithm used in step 1 and $\psi > 0$ is a user-specified constant.*

Proof. To bound the probability of the number of final clusters S is partitioned into, let j be the index of the last cluster to recruit a member of S. Let C^S be the set of clusters where some $w \in S$ has $d(c_i, w) \leq R_i + X_i$ and $i \leq j$. In other words, C^S contains the only clusters which could possibly separate S. We observe that the final number of clusters is upper bounded by the number of clusters in C^S whose radii around original center c_i separates S regardless of whether the cluster C_i' was actually able to recruit any unclustered points from S. We note that such a separation can increase the number of partitions by at most one.

By the same arguments as in the proof of Lemma 12, given that at least one point $w \in S$ has $d(c_i, w) \leq R_i + X_i$, the probability that the radius around original center c_i separates S is at most $d/\psi R$. This follows from taking u and v to be the points in S_i which are closest and farthest, respectively, from the center and upper bounding $d(c_i, v) - d(c_i, u) \leq d(u, v) \leq d$. We further note that if any $C_i' \in C^S$ fails to separate S, then any unassigned points in S will be assigned to C_i' and no future clusters will be able to separate S. Thus, for S to be split into more than t clusters, the first t clusters in C^S must each separate S. This occurs independently with probability at most $d/\psi R$ for each cluster after conditioning on the clusters' membership in C^S. □

4.2.5 Benchmark Dataset Experiments

We ran experiments on the well-known p-median dataset from OR-Lib [128] which contains 40 different problem instances. It was originally generated for the

p-median problem [129], but has since been commonly used to evaluate k-center algorithms and heuristics [130, 131]. Another advantage to benchmarking with this data is that the optimal radius is now known for each of the 40 problem instances in the dataset. The specified number of centers, k, varies across the instances with the smallest being $k = 5$ and the largest being $k = 200$. We evaluate our approach on all 40 problem instances.

Experiment Design

We compare three "unfair" algorithms to multiple versions of our fair algorithm using different parameters. In all cases, we use $d(u, v)/R_{Scr}$ as the target separation probability bound where R_{Scr} is the radius found by Scr heuristic defined below. This choice is somewhat arbitrary, but it provides a fixed target to compare the different algorithms and the Scr radius serves as a fairly close approximation to unfair optimal, which we assume is unknown to the algorithms. Thus, if someone were to apply our algorithm in practice, the radius found by Scr (or other chosen heuristic) would be their best guess at the optimal radius. Each of the three deterministic "unfair" algorithms was run once per dataset, while each fair algorithm was run for 10,000 trials in order to evaluate average performance.

The "unfair" algorithms. In order to compare and evaluate our algorithm, we implemented three algorithms for the classical k-center problem: Gonz1, Gonz+, and Scr. The first two are variations of the famous Gonzalez algorithm [74]. While they do not achieve the strongest results on this dataset, they give theoretically op-

timal approximations and are known for their exceptional speed and simplicity. The third algorithm, Scr, achieves nearly optimal results [130] on the dataset. Recent heuristics have yielded marginal improvements over Scr [131], but we choose Scr because it achieves nearly the same results while remaining fairly simple to implement and reproduce.

Fair algorithm implementation. Our implementation of the fair algorithm uses Scr to find the initial set of centers. We choose Scr since it gets the tightest radius to begin with. We parameterize our algorithm with the mean, $1/\lambda$, of the exponential distribution we sample from, where λ is the exponential parameter used in Algorithm 5. For our "Exact" fair algorithm we set $\lambda = 1/R_{Scr}$ which corresponds to a theoretical separation ratio at most $d(u,v)/R_{Scr}$ for each pair of points (u,v). For our "Medium" fair algorithm, we set $\lambda = 4/R_{Scr}$ since $R_{Scr}/4$ is our target community radius described in our comparison criteria below. Finally, for our "Tight" fair algorithm, we simply divide our mean by another factor of 4 to get $\lambda = 16/R_{Scr}$. Using three different parameters gives some indication of the compromise that can be reached between minimizing the radius and optimizing the fairness.

In addition, our implementation makes two natural modifications to Algorithm 5 that do not affect the theoretical bounds. First, the list of centers found in Step 1 is uniformly randomly permuted before growing the clusters. Second, if we have to choose a new center point in Step 6, we choose the point in the cluster which minimizes the radius as opposed to any arbitrary point.

Comparison criteria. We compared the algorithms in terms of three criteria: radius, pairwise fairness, and community preservation. First, we looked at the approximation of the radius with respect to the unfair optimal. This is the ratio of the radius found by each algorithm to the optimal radius (known for this dataset due to [132–134]). For the randomized algorithms, we give the average radius across all trials. More specifically, this is an average taken over the max radius of each trial derived from the cluster with the largest radius in keeping with the k-center objective.

To evaluate the pairwise fairness, we considered only pairs of points with $d(u, v) \leq R_{Scr}$ (i.e. target maximum separation probability at most 1). For each such pair, we compute the ratio of the algorithm's separation probability to the target maximum separation probability. For the deterministic algorithms, the numerator of this ratio is 0 (not separated) or 1 (separated). For the randomized algorithms, the separation probability is given as the number of trials where the points were separated divided by the total number of trials (10,000). Then, for each algorithm, we take the worst separation probability ratio among all pairs of points with distance at most R_{Scr}. For the deterministic algorithms this is determined by the nearest pair of points which is separated.

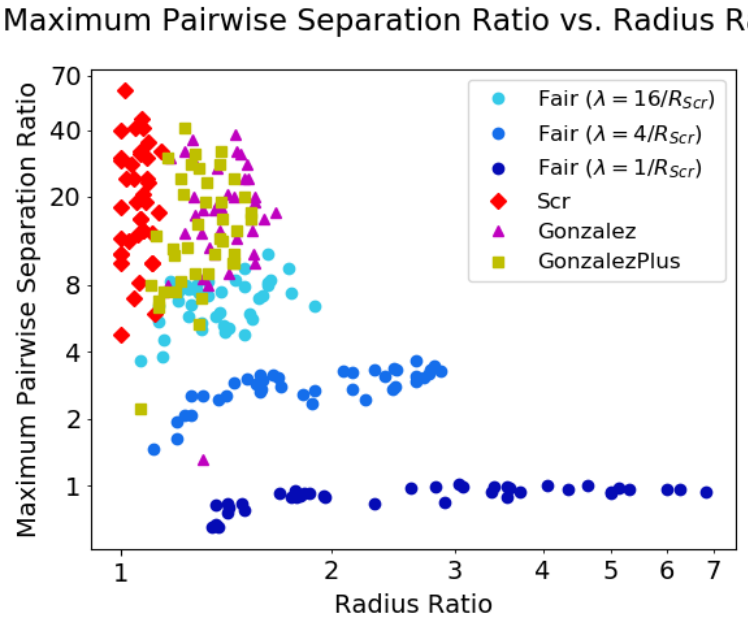
In order to address communities, we needed to define some specific type of community since analyzing every possible subset of points is infeasible. In practical applications there may be some specific target communities based on domain information. However, for this experiment we say that every point defines a community including itself and all other points within a distance of at most $R_{Scr}/4$ from it. In

practical terms, each point could be a person and its community could be that person's neighborhood. We assume the community radius is smaller than the clustering radius as is the case with real world examples such as congressional voting districts. For each point's community, we count the number of different clusters its points have been assigned to. To show the worst case, we highlight the most fractured community, meaning the community split into the most different clusters. For the randomized algorithms, each community gets an average value over all trials and we note the community with the worst average.

Experimental Results

Figure 4.2 summarizes the main results of our k-center benchmark dataset experiments. Overall, we see a clear trade-off between fairness and minimizing the radius with the three different parameters of our fair algorithm.

For the maximum pairwise separation ratio, even our Tight algorithm is more fair than any of the unfair algorithms across almost all instances without paying too much cost in terms of larger cluster radii. This implies that even slight random perturbation of the clusters can dramatically improve fairness with limited impact on the maximum radius of the solution. The pairwise separation ratios for the Exact fair algorithm are roughly 1 or less. Some pairwise separation ratios slightly above 1 are to be expected even for Exact since this is the worst performance of any pair of points in a given problem instance and we are running only 10,000 trials of each randomized algorithm. Likewise, the pairwise separation ratios of the Medium



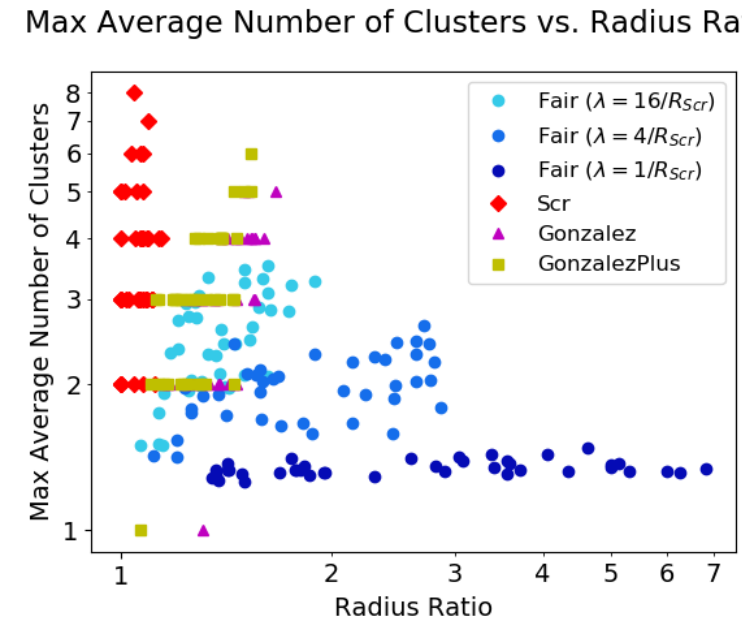


Figure 4.2: Comparison across all 40 instances of the pmed dataset. The three shades of blue circles show our algorithm parameterized by λ of $16/R_{Scr}$, $4/R_{Scr}$, and $1/R_{Scr}$, while other shapes show the unfair algorithms. Points closer to the bottom are more fair while points closer to the left represent solutions with a smaller radius. Our algorithm outperforms the unfair algorithms in both separation ratio (left) and community preservation (right) at the expense of radius as expected. Comparing the three versions of the fair algorithm, we see a clear trade-off between fairness and minimizing the radius.

fair algorithm are roughly upper bounded by 4 as expected. In several cases, the pairwise separation ratio for Exact is actually below 1 meaning that *every* pair of points (u, v) in those instances with $d(u, v) \leq R_{Scr}$ is separated with probability less than $d(u, v)/R_{Scr}$.

With respect to community preservation, we can see that the performance of Tight approaches the two Gonzalez algorithms and is only slightly fairer than the unfair algorithms. However, the maximum average number of different clusters for Exact is always less than two. On some instances, Scr separates some small community of nearby points into 6 or more clusters while Exact gives every community a guarantee that it will be preserved in a single cluster with fairly good probability.

In summary, the fair and unfair algorithms perform as expected yielding a reasonable trade-off between fairness and small radii. The effect of adjusting the λ parameter varies based on the structure of the input. In many cases, using a smaller λ than Exact could be a desirable heuristic if assumptions can be made about the input. Another option, time permitting, is to perform a binary search for the λ which best satisfies a desired balance of fairness and cluster tightness.

4.2.6 Experiments on Real Data

We ran additional experiments on a sample of 1,000 points from the adult dataset [135]. To create the metric space, we normalized the numeric features of age, education-num, and hours-per-week and used them to define points in euclidean space.

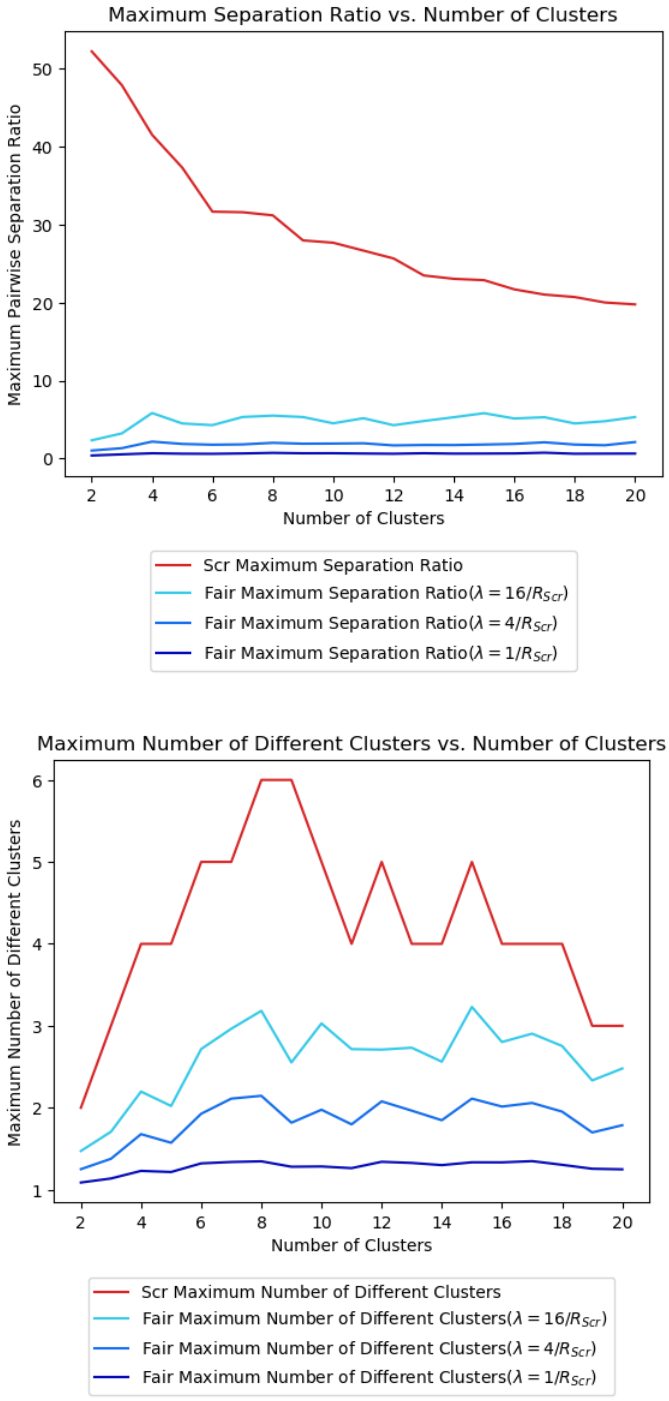


Figure 4.3: Comparison over different numbers of clusters, k, from 2 to 20 on the adult dataset. We measure the maximum pairwise separation ratio (left) and maximum number of different clusters any community is separated into (right). In both cases, lower values on the y-axis are more fair. We compare Scr to three versions of our algorithm parameterized by λ of $16/R_{Scr}$, $4/R_{Scr}$, and $1/R_{Scr}$. We see that the most extreme fair algorithm, $\lambda = 1/R_{Scr}$, is not only the most fair, but most consistent across different values of k.

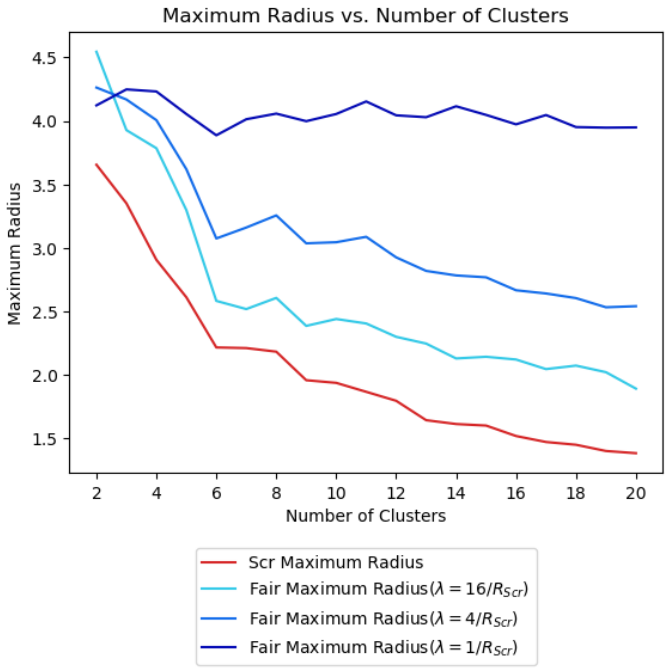


Figure 4.4: Comparison over different numbers of clusters, k, from 2 to 20 on the adult dataset. Here, we measure the maximum radius. In both cases, lower values on the y-axis represent more optimally compact clusters. We compare Scr to three versions of our algorithm parameterized by λ of $16/R_{Scr}$, $4/R_{Scr}$, and $1/R_{Scr}$. We see that the more extreme fair algorithms (smaller λ parameter) suffer a greater price of fairness, but this is constrained within the theoretical bounds shown in Section 4.2.4.

Experimental Design

The design is similar to Section 6.1.10 with the following changes. To evaluate performance while changing the parameter k, we now study a single dataset, but vary the number of clusters, k, from 2 to 20. Given that we do not know the optimal radius for this data under different numbers of clusters, we use the actual radius instead of a ratio in Figure 4.4. In addition, we only consider one "unfair" algorithm, Scr, which gets closest to the optimal radius in practice.

Experimental Results

Figure 4.3 shows that the fairer algorithms are more fair as expected. However, we also see that as we scale the parameter toward greater fairness, the fairness level becomes more consistent and robust to different values of k. Figure 4.4 illustrates the price of fairness we pay in terms of the maximum radius of any cluster. In all plots, we see predictably strange behavior at the extreme low values of k (e.g., when $k = 2$, the maximum number of clusters a community can be fractured into is at most 2).

4.2.7 Future Directions

We introduced and motivated the concepts of pairwise fairness and community preservation to the k-center clustering problem. To explore the practicality of such constraints, we designed a randomized algorithm that can be combined with existing k-center algorithms or heuristics to ensure fairness at the expense of the objective value. We validated our algorithm both theoretically and experimentally.

In terms of future work, there are several open questions around how these new fairness concepts can be combined with other constraints or objectives including other definitions of fairness. For the k-center problem itself, it is unknown whether our bounds on fairness or the objective function can be improved. Further, one could ask if these fairness properties can be extended to variants of k-center such as capacitated k-center which is well-motivated by many real world applications. Other natural constraints to combine with include other notions of fairness or linkage

constraints as seen in semi-supervised learning. We note that pairwise fairness and community preservation can be directly at odds with group fairness (e.g. if points belonging to the same group tend to be close together in the metric space). Finding the trade-off between these fairness concepts is an open problem although it is not clear that many application contexts would require both at the same time. Finally, these definitions could be extended to other common objectives such as k-median and k-means. Our algorithm targets α and β which are functions of the unfair radius R, a natural parameter given the k-center objective. However, for k-median, we may instead use the average distance from points to centers. While it is easy to see how our fairness definitions could apply to other objectives, our algorithm does not extend to these objectives.

Chapter 5: Effects of Gerrymandering Regulation

This chapter represents joint work with Aravind Srinivasan and Shawn Zhao [136].

5.1 Introduction

Computer algorithms and automated systems have become crucial players in the game of drawing and evaluating US electoral districts. Governments have increasingly utilized software to draw districts that influence the outcomes of elections (e.g., to favor a particular political party, incumbent politician, or racial group). Conversely, academics and enthusiastic citizens have proposed algorithms that purport to be fair alternatives [93, 97, 137], a familiar promise of technology that does not always hold true even with the best of intentions. More recently, major US court cases have highlighted computational approaches to evaluate whether a district map is gerrymandered to favor a particular political party [138–141]. These legal challenges include a Pennsylvania Supreme Court case which led to the redrawing of congressional districts in that state [142] and a recent landmark US Supreme court case [7].

At the heart of algorithms claiming to draw fair, unbiased districts or evaluate existing maps, we find a series of metrics and constraints that attempt to define

what makes a district fair, unbiased, or even legal. Our goal in this work is to illustrate unexplored or under-explored aspects of how our choice of metrics and constraints can influence voter behavior and how regulations based on these metrics can be circumvented. In the context of evaluating maps for partisan gerrymandering, we show that using past voting data to evaluate maps can incentivize strategizing among voters even in two-party single-member district plurality systems.

In some sense, strategic voting in an election is not inherently bad. The seminal works of [143, 144] showed that elections with more than two candidates are not strategyproof. We see this play out regularly in US elections when a voter prefers a third party candidate, but chooses to vote for their favorite among the two major-party candidates. However, there are some clear negative effects. First, strategizing reduces the effectiveness of gerrymandering regulations if players can get around them even for single election cycles. Second, the ability to strategize may not be fairly distributed, disenfranchising voters who are less able to strategize. The work of [145] describes this disparate effect in terms of classifiers. In the specific case of redistricting and the strategic act of gerrymandering itself, a political party with a rural voter base can have more power to gerrymander in its favor than an opposing party with an urban base [146].

The ruling in the most recent US Supreme Court case on partisan gerrymandering has essentially left it to states to decide how they will address the issue, if at all [7]. In doing so, state governments will need to consider the downstream effects of how they choose to measure and regulate partisan gerrymandering. To that end, we initiate the study of how measuring gerrymandering using past voter

behavior can incentivize strategic voting to circumvent regulations. We show that careful scrutiny should be given to any measurement which uses past voter behavior to evaluate and affect the choice of district maps.

5.2 Problem Description and Definitions

In this section, we outline many problems and definitions associated with districting and gerrymandering, including proposed methods to measure gerrymandering or draw fairer districts. We begin with some basic definitions, then introduce the growing role of computer science in combating gerrymandering. Finally, we summarize recent attempts to evaluate or draw districts. Throughout the paper, we restrict our discussion to the United States political system, where gerrymandering has become a highly contentious issue, as a guiding example.

5.2.1 Basic Districting and Gerrymandering Definitions

In US politics, many representatives in both federal and state governments are elected via single-member district plurality systems where voters are partitioned into districts which each elect representatives with a plurality vote. A *plurality vote* awards a seat in government to the candidate who has received the most votes even if that candidate has not received a majority of the total votes cast. Election districts may be redrawn every 10 years following a population census. This ensures that districts respond to population shifts over time and remain reasonably balanced in terms of the number of voters per district to avoid vote dilution.

Historically, state governments have been charged with drawing new maps, and the party which holds a majority of seats in a state's government may control the outcome of this process. More recently, some states have transitioned to a system where an "independent" commission draws the maps. Such commissions have been controversial in terms of both how their independence can be guaranteed and whether they violate the US Constitution which assigns this duty to state governments. Nevertheless, the US Supreme Court has upheld the right of voters in a state to give redistricting authority to an independent redistricting commission [147]. The present work mainly explores models where the party controlling the majority of seats also controls the drawing of districts subject to some restrictions. However, our results may be useful in understanding how to regulate independent commissions as well.

The strategy of gerrymandering is to influence the outcome of an election through the process of drawing districts. This can be done by *packing* many voters from one group into a single district where they will cast more votes than needed to win, or *cracking* voters from that group into multiple districts where they will cast votes for losing candidates in each district. In both packing and cracking, the idea is to make the opposing group "waste" as many votes as possible while making one's own group waste fewer votes.

The type of gerrymandering depends on the type of groups being targeted and the intended outcome. *Partisan gerrymandering* attempts to favor one political party over another. Similarly, *racial gerrymandering* attempts to favor one racial group over another. *Incumbent gerrymandering* is slightly different in that it creates

a bias toward re-electing an incumbent candidate. The primary focus of this paper is partisan gerrymandering. However, these different types are often entangled such as when there is a correlation between racial demographics and party membership or when partisan gerrymandering leads to the creation of "safe districts" where incumbents have an advantage. In fact, laws that prevent the cracking of racial groups have been used to justify packing members of a racial group for the apparent purpose of partisan gerrymandering as in Florida's famously snake-like District 5.

The space of legal district maps is restricted by a set of (sometimes competing) constraints and objectives that vary from state-to-state. The most common restrictions are contiguity, community integrity, population balance, hole-freeness, and compactness. *Contiguity* simply means that districts should be contiguous spaces although in the more extreme cases, they may only be connected by narrow paths. *Community integrity* refers to the objective that districts should avoid splitting defined communities (e.g. counties, towns, etc.) if possible. However, communities are routinely split ostensibly to meet other objectives. *Population balance* is the objective that the number of voters in each district should be as balanced as possible in order to give roughly the same weight to each person's vote. The degree to which population balance is violated can depend on other considerations such as community integrity, and congressional district population sizes within a state can vary by as much as 897,080 in Texas District 22 to 713,480 in Texas District 13 [148]. There are also several single-district states where the district size is determined solely by the state population and may be larger than the national average or smaller districts. Montana's at-large district has an estimated popula-

tion of 1,050,493 compared to one of the smallest districts, Rhode Island's District 2 with 520,389 [148]. *Hole-freeness* states that no district should be completely surrounded by one other district. Finally, *compactness* is perhaps the least consistently defined goal with many possible definitions of compactness existing. These include k-median-like objectives minimizing the average distance a voter has to travel to a center point in their district and objectives minimizing ratio of area to perimeter (see Section 5.2.5 for a discussion of algorithms that use these objectives).

5.2.2 Using Computer Science to Combat Partisan Gerrymandering

There are two main directions where computer science has become involved in combating partisan gerrymandering: verifying whether a given map is unfairly gerrymandered and drawing fair maps. Crucial to both directions is the question of how to measure gerrymandering and define fairness in this context. This is clearly true for a verifier, and an algorithm for drawing districts must have some notion of where the line is between fair and unfair in order to avoid crossing it.

Unfortunately, it is not simple to measure whether partisan gerrymandering has occurred nor is it straightforward to say that partisan gerrymandering is "unfair" in the US legal context (e.g. unconstitutional). While gerrymandering may often seem obvious just by looking at a map, this so-called "eyeball test" is not robust [149]. It is possible that a strange looking map is actually subject to the geography of a state with respect to both natural and artificial features that perturb distances between voters such as mountains or highways. On the other hand, it is

also possible for a reasonable looking map to be gerrymandered. Then, supposing we have some test to show that a map is gerrymandered, we must further show that the practice we measure violates the law in some way. In the US legal system, the mere practice of gerrymandering is not illegal even though the concept of candidates choosing their voters instead of the other way around may violate many citizen's sensibilities. For example, the US Supreme Court has ruled that racial gerrymandering is unconstitutional while incumbent gerrymandering is allowed [92, 150].

5.2.3 Approaches to Measurement

Here, we define and discuss some of the most well-known approaches to measure partisan gerrymandering. These include the look test, outlier detection, proportionality, competitiveness, the efficiency gap, and compactness. Our work focuses primarily on outlier detection and proportionality, but is also relevant to any measurement approach using past voting data.

One seemingly obvious standard which may use past voting data is *proportionality*. This is the idea that the proportion of seats assigned to each party should be close to the proportion of votes received by each party. However, there are challenges to this standard as well. Drawing a map which achieves proportional representation is impossible for states like Massachusetts where voters of the two parties are too evenly distributed [151] or single-district states. More importantly in the US, courts have rejected proportionality tests. For the sake of analysis and comparison to other tests, we consider an achievable proportionality concept of maximally proportional

maps. A *maximally proportional map* is one which comes as close as possible to allocating seats proportionally among all *known* maps (given the large search space, we may not know the true most proportional maps).

More central to our work is the recent study of outlier maps. An *outlier map* is one which is abnormal by some measure of representation. In this paper, we use the natural measure of seat count awarded to each party for a map based on past voting data. Thus, a map may be an outlier if it awards 3 out of 11 total seats to a given party while almost every other legal map awards 4 or 5 seats to that party.

A key challenge in detecting an outlier map is that it is difficult to determine what an average map is. The space of all possible maps is too large to check each one. This has led to approaches that aim to approximately randomly sample from the set of all possible legal maps using Markov Chain Monte Carlo techniques [138–140]. One can generate thousands of random maps and use past voting data to determine how many seats each party would win on each map assuming voters were to vote the same way they have in the past. We can compare a given map to this random sample to determine if it is an outlier. The seat count measure of representation we consider is used in [139]. Other works use more fine-grained measures that capture smaller changes in the distribution of voters to districts [138].

Additional measures which use past voter data include competitiveness and the efficiency gap. The *competitiveness* of a district attempts to capture the extent to which it might be won by either of two parties. The *efficiency gap* measures the difference in the number of wasted votes between two parties [152]. Generally speaking, any vote cast for a losing candidate is wasted, and for a winning candidate,

a wasted vote is any vote beyond 50% (or alternatively the minimum needed to win in some variants). This method was used unsuccessfully in a US Supreme Court case challenging gerrymandering in the state of Wisconsin [153] and critiques of it can be found in [154, 155]. Nevertheless, we conjecture that an efficiency gap standard might incentivize participation and truthful voting in contrast to the outlier detection method discussed here.

Finally, we address measures that explicitly do not take past voting behaviour into account. These typically involve some mathematical definition of what a good cluster should look like. For example, a ratio of perimeter to area, average distance from all voters to single meeting point, or even the average number of neighboring districts. They often arise in the discussion of algorithms for drawing districts, and we save a closer examination for Section 5.2.5.

5.2.4 Redistricting Subject to Gerrymandering Regulations

In this paper, we consider models where humans control the redistricting process (possibly using any algorithms they please), but they may be restricted by gerrymandering metrics. In particular, we focus on a model we call the *majority party draw model* where the political party currently in power in a state controls the redistricting process. Thus, we assume a biased, partisan agent is drawing the maps to favor one party over the other. Districts are drawn by the party holding the majority of seats in order to maximize that party's utility subject to any constraints. We constrain the drawing party with the typical restrictions that districts

must be contiguous and balanced in terms of population. In addition, we include a regulation proposed to reign in partisan gerrymandering, *banning outlier maps*. This regulation prohibits the drawing of outlier maps with respect to seat counts awarded to each party based on the votes cast in the most recent previous election.

5.2.5 Other Approaches to Drawing Districts

While this work primarily studies approaches to measuring gerrymandering and their potential effects, we briefly discuss the topic of drawing districts as one might argue, "Why not just use one of the existing tools for drawing fair districts?" We describe two broad lines of work devoted to drawing districts without partisan gerrymandering and sketch why these existing techniques, by themselves, may not be sufficient or desirable. We group the techniques by whether or not they use past voting data.

The first general class of algorithms embraces the natural idea of drawing districts without considering voting preferences at all. We describe some of these works and then argue why it is still important to test them for partisan bias.

Many of these proposed tools attempt to optimize some measure of compactness. A k-median-based algorithm is featured in visualizations of different districting schemes on the website FiveThirtyEight [156]. The similar objective of balanced centroidal power diagrams is used in [97]. There are also methods that focus on simple, achievable objectives like the shortest splitline algorithm which recursively finds the shortest line dividing the population in half until the desired number of equally sized

districts is found [137].

Although tools in this category are often self-described as unbiased, testing their output for gerrymandering via unintentional bias (e.g., using measures from Section 5.2.3) is still needed. In the classic fairness example of classifiers, we know that sensitive features such as race can be redundantly encoded in other features even if race itself is omitted from the feature set. Similarly, party membership (or race) may be encoded in other features of population data that are used by these algorithms.

In the US, features such as party membership and race are correlated with urban versus rural residence. Thus, any clustering algorithm which is sensitive to the density of points being clustered could potentially be biased. For example, a bias based on population density could be present in algorithms which minimize average distance from voters to a central meeting point such as those based on the k-median objective or balanced centroidal power diagrams. This is not to diminish the value of these algorithms or claim that they are biased, but rather to show that they should be tested for biases. This evaluation requires a measure like those described in Section 5.2.3 and studied in our present work.

We note that this concern about hidden bias is not new. Other works have explored the effects of an urban/rural party split [146,157] and the following statement expressed by Justice Scalia in [158] was discussed in [141].

> Consider, for example, a legislature that draws district lines with no objectives in mind except compactness and respect for the lines of political

subdivisions. Under that system, political groups that tend to cluster (as is the case with Democratic voters in cities) would be systematically affected by what might be called a "natural" packing effect.

In the second category, are approaches that explicitly use voting data. Some of these use measures from Section 5.2.3 to guide the algorithm. An evolutionary algorithm called PEAR uses a combination of objectives including competitiveness based on past voting data and compactness measured as 4π times the area of a district divided by its perimeter squared [93]. There are also cake cutting approaches which allow two parties to divide up a state [159]. This is shown to meet a definition of fairness to the two major parties, but may not be fair to other groups such as third parties, geographic regions with shared interests (e.g., farming communities), or racial groups. Allowing the two major parties to collaborate on drawing a map may also lead to undesirable incumbent gerrymandering as in California in the past [160]. While our work does not directly address these tools, it implies that they could be susceptible to strategic voting.

5.2.6 Other Related Work

Here, we summarize some additional related work on social choice theory and fairness.

The classic Gibbard-Satterthwaite Theorem [143,144] established that at least one of the following must hold in ordinal voting systems electing a single candidate.

1. The system is a dictatorship; one voter chooses a winner.

2. The are only two candidates.

3. The system is not strategyproof and inspires tactical voting.

While our result shows strategizing in a two party election with single candidates elected per district, we note that in our model, a single voter's strategy can affect the outcomes of multiple elections. Even though each district elects a single candidate, we show how a vote in one district can affect all districts in future elections. Thus, our work highlights a situation where the Gibbard-Satterthwaite Theorem may appear to apply, but surprisingly does not.

In the context of fairness, [145] considered the disparity that results from different groups having different abilities to strategize. Our work shows how certain regulations can create additional opportunities for strategic manipulation in voting systems and opens the question of whether additional disparity results from those regulations. This is relevant in elections where political parties are scrutinized to determine if they are leveraging such disparities to disenfranchise groups of voters (e.g., using voter id laws to target communities living in urban areas where drivers' licenses are not ubiquitous or restrictions on early voting to target people with less flexible working hours).

5.3 Our Contributions

We illustrate how using past voting data to regulate the drawing of district maps can incentivize voters to strategize and vote untruthfully. In particular, we show that a single-member district plurality system under the policy of banning

outlier maps with only two parties is not strategyproof.

Under the policy of banning outliers, we show examples of how a party holding the majority of seats can vote and draw districts strategically to increase the number of seats they win. For banning outliers, we provide a heuristic for a party controlling the districting process to identify pure strategies that lead to winning more seats and test this heuristic empirically. Finally, we use grid graph models to explore questions from the US Supreme Court case Rucho v. Common Cause [7] relating to outlier maps.

While not the primary pursuit of this work, our observations may also be relevant to the areas of election security and machine learning. In the case of election security, voter fraud, and tampering with election results, we essentially reveal a scenario wherein a political party could gain in the long run by generating votes for the opposing party. Such a scenario may be difficult to detect if one is assuming that a cheating party would only assign votes to itself. In relation to machine learning, we can view recent computational efforts to detect gerrymandered maps as classifiers. Thus, our work explores how these classifiers interact with a broader system.

5.4 Modeling the Game of Voting and Redistricting

We introduce a simple game to model the cycle of drawing districts, voting, redrawing districts, and voting again. The drawing of districts is constrained by a regulation which uses past voting data to decide if a district map is legal or not. The goal of the model is to clearly illustrate how a regulation can incentivize voters

to vote strategically rather than truthfully and better understand how regulations might be circumvented.

In this game, there are two political parties, *red* and *blue*, which we denote R and B, respectively. Voters are vertices in a graph $G = (V, E)$ with the number of vertices $|V| = n$ and the edge set E signifying neighboring voters. We define a map m as a partition of G into k districts. Each district must be a connected component in G with size equal to n/k (we assume for simplicity that k is odd and n/k is an odd integer). In this way, we enforce the rules that districts must be contiguous and perfectly balanced in terms of population. In Sections 5.5 and 5.7, we further restrict G to be a grid graph. Thus, in those sections, each district will be an equal-sized, non-overlapping polyomino as seen in the maps of 3×3 grids in Figure 5.1.

Each voter v has a true preference for one of the two parties. The set of true preferences for all voters $P \in \{R, B\}^n$ is known to both parties at the start of the game and does not change over the course of the game. Having true preferences known to the parties captures the fact that parties may know more about their voters (including how they have influenced them) in comparison to the regulation which only "sees" how people have voted. Without loss of generality, we assume the red party holds the majority of seats at the start of the game. We also use $q_v \in \{R, B\}$ to denote the most recent vote by each voter v and let $Q \in \{R, B\}^n$ be the known set of votes from the most recent election. We reiterate that set Q of votes from the previous election is known to both parties and the regulation at all times.

Finally, we have a regulation $\psi\colon m \mapsto \{legal, banned\}$ which determines whether a given map m is legal to use or banned and cannot be used. For the remainder of this work, we focus on the regulation of *banning outliers*. In this case, ψ also takes as input the previous votes Q, set of all possible maps M (or a set of maps sampled from M in Section 5.8), and a threshold $\tau \in (0, 1]$. If the fraction of maps awarding the same number of seats to the red party as m is less than τ, then m is banned. Otherwise, it is legal. In other words, m is a banned outlier if the number of maps awarding the same number of seats to red as m is less than $\tau|M|$.

At the start of the game, we let $Q = P$ assuming that in a prior election voters cast votes according to their true preferences. The reason for this choice is two-fold. First, it is the simplest and easiest to analyze case which still addresses our major question of whether strategic voting can be incentivized by gerrymandering regulation. Second, this models the adoption of a new regulation. We can assume that voters have been voting their true preferences in the past and the game starts at the moment when the regulation is imposed.

The game proceeds in four rounds so that we can observe the effect of a round of voting on which maps can legally be chosen. The objective of each party in this zero sum game is to win as many total seats as possible.

Round 1: The majority party (red) draws a map m subject to a gerrymandering regulation ψ. The voters' true preferences are used as the past voting data for the purpose of regulating this first round (i.e., $Q = P$).

Round 2: All voters vote simultaneously, but voters of the same party may collude. The votes are tallied, and each district's seat is awarded to whichever party won the majority of votes in that district.

Round 3: The party which won the majority of seats in Round 2 draws a new map m subject to ψ. However, Q is now the set of votes from Round 2 and this may affect ψ.

Round 4: Again, all voters vote simultaneously, but voters of the same party may collude. The votes are tallied, and each district's seat is awarded to whichever party won the majority of votes in that district.

While this game only captures two election cycles, we will show that it is able to reveal an incentive to vote strategically. Natural extensions to more cycles or more rounds of voting before redistricting will be addressed briefly in Section 5.10. We further note that in this short game, we can assume voters will vote their true preferences in Round 4 since these final votes are only used to determine the outcome of a single election.

5.4.1 Simplifying Assumptions

Here, we outline and discuss a number of simplifying assumptions in our model and analysis. Further discussion of these assumptions and related future directions

appears in Section 5.10.

To simplify the analysis of collusion, we assume each party controls all of its voters and chooses how they will vote. Furthermore, we require each voter to vote for one of the two parties. They do not have the option to abstain or vote for some third party. Therefore, in our model, a candidate winning a plurality of votes also wins a majority. To further guarantee clear majorities without ties, we use an odd number of districts with an odd number of voters in each district.

We consider the utility of a party to be a linear function of seat count. This reduces the space of strategies to explore since a party cannot gain utility by sacrificing a seat in one round in order to gain a seat in another round. However, one could also envision a more complex model with a nonlinear function that captures real world effects. For example, in many cases in the US system, their is a large added benefit to holding a 2/3rd majority. In a nonlinear model, sacrificing a seat in one round to win a seat in another could be beneficial.

We note that in the US system, there are typically multiple elections between the rounds of redistricting that can occur following each decennial population census. Thus, our abstraction replaces a series of elections with a single voting round.

5.5 A Simple Example Game on a 3×3 Grid

To illustrate our game from Section 5.4 and the effects of regulation, we start by looking at the regulation of banning outliers applied to a specific set of voter preferences on a 3×3 grid with 3 districts of size 3. In this setting, we can clearly

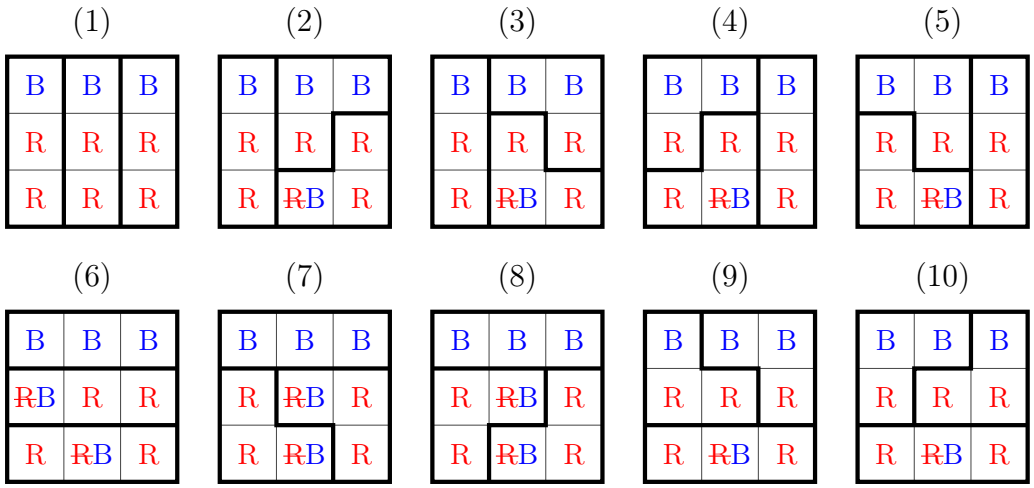


Figure 5.1: All 10 possible maps of a 3×3 grid into 3 contiguous districts of equal size. Blue B's indicate voters who prefer the blue party and red R's indicate voters who prefer the red party. Squares with a crossed-out ~~R~~ followed by a B represent red party voters who can vote for the blue party in Round 2 in order to make map 1 appear to be a non-outlier map for the next round of drawing districts. Map 1 is the only map which awards 3 seats to the red party under true preferences, making it the favorite map for red, but also an outlier.

visualize an exhaustive set of maps.

A 3×3 grid admits 10 maps of 3 districts when contiguity and population balance are the only restrictions. Figure 5.1 shows all 10 maps along with a set of true voter preferences and strategies. The top row of voters prefer the blue party, while the bottom two rows prefer the red party. We can see plainly in Figure 5.1 that the red party would prefer the first map which partitions the voters into three columns. This map cracks the blue party so that red wins all 3 seats and it is the only map in which red wins 3 seats as opposed to 2.

For this simple example game, we consider the regulation of banning outliers with a threshold of τ that is strictly greater than 0.1, essentially the smallest meaningful threshold for this graph (we consider smaller, more realistic values of τ in later sections). Thus, the first map in Figure 5.1 will be banned with respect to the

voters' true preferences P and therefore banned in Round 1. Because this is the preferred map for the red party, but it cannot be chosen given the regulation ψ and voting history $Q = P$, we call map 1 the target map for red. In other words, this is the map that red would like to draw in Round 3 in order to win an extra seat in the game.

Now, suppose one or more of the red voters were to vote for blue in Round 2 without giving up a seat in that round. This could make it appear that the first map will award 1 seat to blue in future elections when in fact the red voters could then vote truthfully to award all seats to red. We may then ask if blue can respond with its own strategic voting, but in this case red has a pure strategy for any choice of starting map that blue cannot respond to as stated in Observation 18.

Observation 18. *For any choice of starting map the red party has a pure strategy (illustrated in Figure 5.1) which leads to winning 5 seats total over the course of the two voting rounds in the game defined in Section 5.4.*

Note that in several maps (2, 3, 4, 5, 9, and 10) in Figure 5.1, red can flip a single voter to blue in order to make it appear that the middle column district of map 1 will be awarded to blue. In addition, this will not cause red to lose a seat in Round 2 voting compared to voting true preferences. To potentially counter this effect, blue could flip its voter in the middle column to red. However, this would cause blue to lose that voter's district in Round 2, immediately giving 3 seats to red. Conversely, in other maps (6, 7, and 8) blue can afford to flip one voter to red without losing a district and this forces red to flip two voters from two separate

districts to guarantee that map 1 will not be an outlier when drawing a map in Round 3. Thus, given any legal starting map (all maps besides map 1), red can vote strategically to win 5 seats overall in the game, whereas truthful voting would only yield 4 seats overall. In Section 5.6, we explore this phenomenon more by establishing a set of conditions that permit a majority party to strategize in more general settings.

5.6 Conditions for Strategizing Against Banning Outliers

In the previous section, we saw the existence of a set of preferences which incentivized strategic voting to gain seats when outlier maps are banned. Now, we define a set of conditions under which a pure strategy exists for the majority party to gain at least one seat by carefully drawing maps and influencing its supporters to vote strategically. Taken together these conditions are sufficient to incentivize strategic voting, but may not be necessary, especially when mixed strategies are considered.

In order to strategize, the majority party must be able to find two maps, a *starting map* which it can legally choose in Round 1 and a *target map* which awards the party more seats than the starting map, but is an outlier under true preferences. Additionally, given a starting map and target map, we identify a set of majority party voters called the *shills* who will vote for the opposing party in the first vote of Round 2, but then revert to supporting the majority party in the second vote of Round 4. We also require a set of minority party voters called the *accomplices*

whose truthfull votes in Round 2, combined with the shills' fake votes, can be used to construct a district which appears to favor the minority party while actually favoring the majority party under true preferences. We note that the accomplices from the minority party are, of course, unwilling accomplices whose votes will be used against their own party no matter how they vote. The majority party will put them in the following position. If the accomplices vote truthfully in Round 2, their party will ultimately lose a seat in Round 4, while if they vote untruthfully in Round 2, their party will immediately lose a seat in Round 2. Given these definitions, the conditions below are sufficient for the majority party to strategize.

1. There exists a starting map which is not an outlier with respect to true preferences.

2. There exists a target map which awards more seats to the majority party than the starting map, but is an outlier with respect to true preferences.

3. There does not exist a non-outlier map which, if chosen in all rounds, awards as many or more seats than the combination of choosing the starting map followed by the target map under true preferences.

4. In the starting map, the shills are in a different district from the accomplices.

5. In the starting map, the majority party does not lose a seat if the shills all vote for the minority party.

6. In the starting map, the minority party will lose a seat if any one of the accomplices votes for the majority party.

7. In the target map, the shills are in the same district as the accomplices.

8. In the target map, the district containing the shills and accomplices will go to the majority party under true preferences, but appears to go to the minority party if the shills and accomplices vote for the minority party in the first round.

9. The target map is not an outlier in Round 3 if the accomplices and all majority party voters besides the shills vote truthfully in Round 2.

Using these conditions, we can identify opportunities to strategize as described in the next section.

5.7 Heuristic for Strategizing Against Banning Outliers and Experiments

Based on the conditions from Section 5.6, we devise a heuristic for finding pure strategies for the majority party subject to the regulation of banning outliers. To test this heuristic, we use a grid graph model with a 5×5 grid and 5 districts of size 5.

5.7.1 The Heuristic

We implemented a simple heuristic which takes a set of preferences as input and searches for starting maps, target maps, accomplices, and shills satisfying the conditions from Section 5.6. Considering all potential (starting map, target map)

pairs takes at most $O(\tau|M|^2)$ time since there are $O(|M|)$ potential starting maps and only $O(\tau|M|)$ possible target maps given that target maps are outliers by definition. Searching a pair of maps to find shills and accomplices takes $O(n)$ time with the appropriate preprocessing of each map. Thus, the entire process requires $O(n\tau|M|^2)$ time per preference set. However, the practical running time is much faster since we can stop as soon as we find a satisfying pair of maps.

5.7.2 Experiment Design

We test our heuristic in the scenario of a 60/40 split in the true preferences of voters. Our "state" is a 5×5 grid with 5 districts of size 5. For this grid, we consider all possible sets of true preferences where the majority party has 15 of 25 total voters ($\sim$3.2 million preference sets). A benefit of using this simple model is that we can consider all 4,006 contiguous and balanced district maps in order to identify outliers for a given set of preferences. This divorces the analysis of outlier regulation policies from the questions of how to sample maps and detect outliers that are the focus of [138–140].

Outlier threshold We choose a threshold τ of 2% for banning outliers. In other words, given a true preference set P or voting history Q, any particular seat count awarded to the majority in less than 2% of the 4,006 possible maps is considered an outlier and any maps awarding that many seats would be banned. Aside from being a "natural looking" threshold, this choice is well-suited to our scenario. For a large number of true preference sets, the 2% threshold bans maps that award all 5 seats

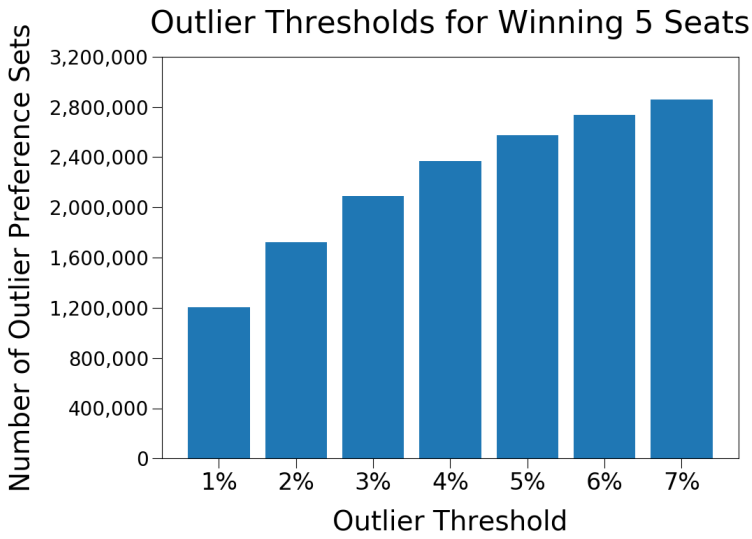


Figure 5.2: The number of preference sets out of 3,268,760 total where a map awarding 5 seats to the majority party is an outlier according to percentage thresholds (1% - 7%).

to the majority party (Figure 5.2). On the other hand, it allows maps that award 4 seats to the majority party for nearly all true preference sets (Figure 5.3). Awarding 4 seats to the majority may be seen as a reasonable deviation from proportionality due to the geography of the voter population. Allowing this amount of deviation is one way for regulators to be clear that they are not enforcing proportionality.

During the two district drawing rounds (1 and 3), the majority party is allowed to choose any of the non-outlier maps. During the voting rounds (2 and 4), both parties' voters may vote strategically, but our heuristic draws maps based on pure strategies where either only the majority party has an incentive to vote strategically or no party votes strategically. For each set of true preferences, we test whether our heuristic can find a strategy to gain more total seats than could be gained through voting truthfully.

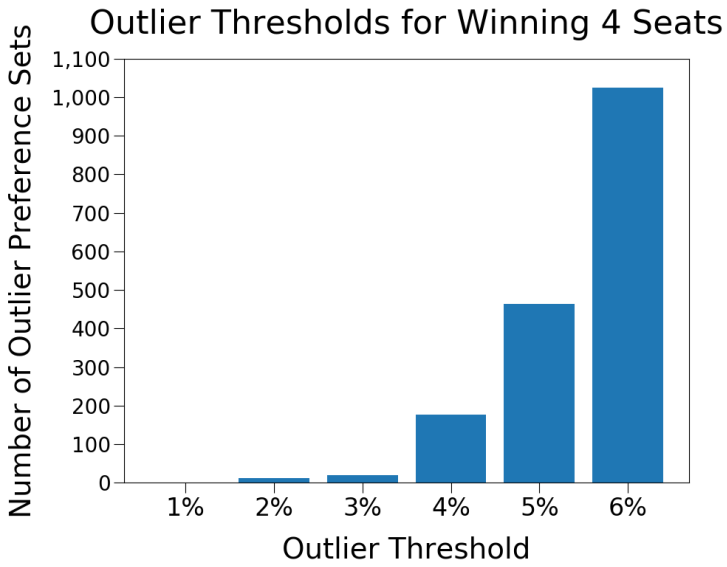


Figure 5.3: The number of preference sets out of 3,268,760 total where a map awarding 4 seats to the majority party is an outlier according to percentage thresholds (1% - 6%).

5.7.3 Experimental Results

Figure 5.4 shows the results of our experiments. We see that for roughly half of the 3,268,760 preference sets, maps awarding all 5 seats to the majority party are not outliers. In those cases, no strategizing is needed. The majority party can simply pick a map which awards it 5 seats and use that map for the entire game. However, on most of the remaining preference sets, the majority party is limited to choosing maps in Round 1 that award fewer than 5 seats under true preferences, but our heuristic is able to find a pure strategy which leads to winning an additional seat in Round 4. For fewer then 200,000 preference sets, the best non-outlier map awarded 4 seats and we were not able to find a strategy. We do not know if any pure or mixed strategies exist for these preference sets.

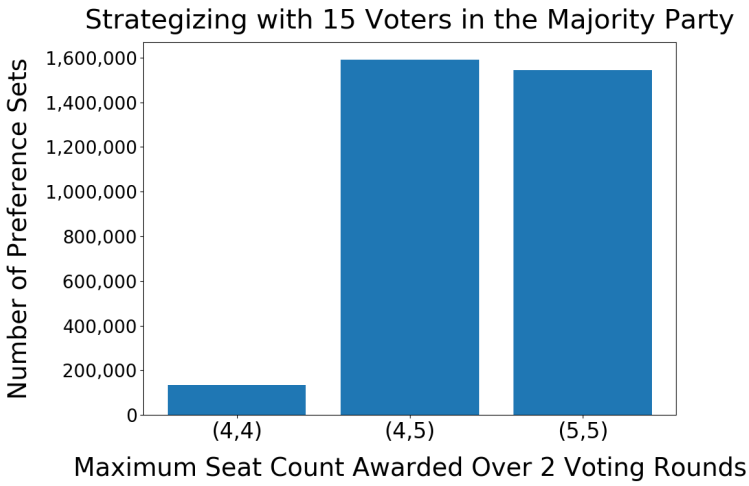


Figure 5.4: Illustrating the ability of our heuristic to find pure strategies on all 3,268,760 preference sets with 15 majority party voters. (4,4) indicates preference sets where non-outlier maps award 4 seats to the majority party under true preferences and our heuristic is unable to strategize for more seats in the second vote. (4,5) indicates preference sets where non-outlier maps award 4 seats to the majority party under true preferences and our heuristic finds a pure strategy to win 5 seats in the second vote. (5,5) indicates preference sets where non-outlier maps award 5 seats to the majority party under true preferences and thus, there is no benefit to strategizing. This figure omits 12 preference sets where the best non-outlier map awards only 3 seats under true preferences and we can strategize to win 4 seats in the second voting round.

5.8 Application of our Heuristic and Model to Real Voting Data

5.8.1 Background

North Carolina is often highlighted as one of the most gerrymandered states in the nation. In the 2012 North Carolina congressional election, over half of the total votes went to Democratic candidates, yet only four of the thirteen congressional representatives were Democrats [139]. In fact, court cases have struck down North Carolina's 2012 and 2016 congressional maps for partisan gerrymandering and a case involving North Carolina's map recently went all the way to the Supreme Court in [7]. In one effort to address this issue, the "Beyond Gerrymandering" project sponsored by the Duke Center for Political Leadership, Innovation, and Service brought together an independent commission of 10 bipartisan retired judges to redraw North Carolina's congressional map without the use of past political data or election results with the intention to generate a more fair district map. The hypothetical map that was produced as a result of this summit will be referred to as the *judges' map*.

5.8.2 Our Results in Brief

In order to lend credence to our model and heuristic, we apply simplified versions to real North Carolina voter data. In the more restricted model, we find one possible strategy for the Republican party to circumvent an outlier ban by using the judges' map as a starting map and the 2016 North Carolina congressional map as

a target map. The changes outlined in the next section make the problem of finding a strategy much more tractable for the larger, messier real world problem. However, we also describe how this more restricted model can still inform our understanding of real scenarios.

5.8.3 Modified Model

We retain concepts of our original game such as banning outliers, a two-party system, and four-round drawing-voting-drawing-voting cycle, as outlined in Section 5.4. We treat each of the 2,692 voter tabulation districts (VTDs) in North Carolina [139] as points on a planar graph and districts as a partition of this graph into connect subgraphs. Each VTD is encoded with Democratic and Republican votes. Crucially, we modify our original game such that the strategizing is one-sided. While the majority party can strategize, the minority party must vote truthfully. Thus, this model prevents the minority party from counter-strategizing. In our previous model, each basic unit was an equivalent voter. In this case, each basic unit (a VTD) does not carry the same weight and the number of potential reactionary strategies from the minority party increases dramatically.

While this restricted game captures fewer real scenarios, we highlight two interesting observations. First, our model is motivated by the idea of a "surprise attack" in which one party decides to manipulate votes to gain an advantage and the other party either does not expect the attack or knows, but cannot mobilize a counter strategy. In practice, this surprise attack could even be a secret attack via

hacking electronic voting systems. Second, although the majority party strategy we find may not be an optimal pure strategy in the original game, it can be used to illustrate the existence of some mixed strategy in the original game. Thus, it still shows an incentive for strategic voting on real data in a less restricted game.

5.8.4 Conditions for Strategizing on the Modified Model

We modify our conditions from Section 5.6 to account for the inability of the minority party to react. The *shills* and *accomplices* are now groups of voters within VTDs. However, they still perform the same role as in our original model. Taken together, these conditions are sufficient, but may not be necessary, for a party to strategize in the modified model of this section. Note that we have maintained the numbering from Section 5.6 in the list below to facilitate easier comparison between the two lists.

(1) There exists a starting map which is not an outlier with respect to true preferences.

(2) There exists a target map which awards more seats to the majority party than the starting map, but is an outlier with respect to true preferences.

(3) There does not exist a non-outlier map which, if chosen in all rounds, awards as many or more seats than the combination of choosing the starting map followed by the target map under true preferences.

(5) In the starting map, the majority party does not lose a seat if the shills all

vote for the minority party.

(7) In the target map, the shills are in the same district as the accomplices.

(8) In the target map, the district containing the shills and accomplices will go to the majority party under true preferences, but appears to go to the minority party if the shills and accomplices vote for the minority party in the first round.

(9) The target map is not an outlier in Round 3 if all voters besides the shills vote truthfully in Round 2.

In comparison to the conditions from Section 5.6, conditions (4) and (6) are no longer needed and (9) is slightly relaxed. This is because we are assuming in the modified model that the minority party accomplices vote truthfully. Thus, we do not need to meet conditions that discourage the accomplices from strategic voting or account for that possibility.

5.8.5 Simplified Heuristic

Because there is now an enormous number of possible maps and a large set of voters, we cannot search exhaustively like our procedures from our grid model. In addition to drastically narrowing our search space to specific starting (judges' map) and target (2016 map) maps, we implement a more targeted heuristic for identifying shills. For the most competitive districts (least margin between voters of the two parties) in our target map not already won by the majority party, we obtain the

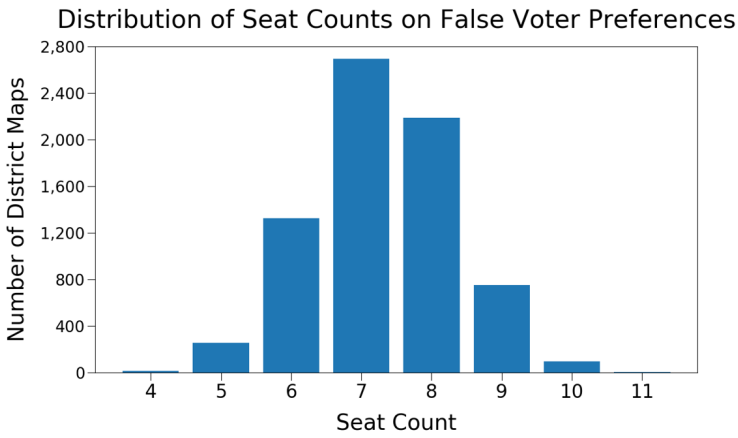


Figure 5.5: Plot of Republican seat distributions of the false voter preferences after all of the Republican votes in the judges' map district 9 VTDs have been flipped to Democrat votes. We determined the seat counts of the false voter preferences on 7319 total maps generated by Monte Carlo Markov Chain that we retrieved from [139]. We note that under our false voter preferences, the 2016 map awards 9 Republican seats which is not an outlier for the false preferences. However, the true voter preferences award 10 seats on the 2016 map, which is an outlier.

list of VTDs in those districts. We then flip as many majority voters from those VTDs as possible without losing any districts on the starting map, such that those VTDs entirely consist of minority votes in Round 2. We then compare the new set of voter preferences on the starting map and target map within the sampling distribution of maps to make sure neither are outliers. We perform this procedure on each district in our target map from most competitive to least competitive until we find a successful strategy or have exhausted the set of losing districts.

5.8.6 Experiment Design

We note that the judge's map awards 9 out of 13 seats to the Republican party and is therefore not an outlier, while the 2016 map awards 10 out of 13 seats to the Republican party and is an outlier [139]. We test our modified model over North

Carolina 2016 House voting data. We set the judges' map as our starting map and the 2016 map as our target map. We set the Republican Party as the majority party and the Democratic Party as the minority party. Voter data, the judges' map, the 2016 map, and a sampling distribution of maps were retrieved from the Github provided in [139].

5.8.7 Results

We find that flipping all of the Republican votes to Democrat votes in VTDs found in district 9 of the judges' map causes district 2 of the 2016 map to seem to be a Democratic district without the Republicans losing any districts in the judges' map. However, according to the real voting history, district 2 of the 2016 map is a Republican district. Thus, under the false preference sets, the 2016 map seems to award only 9 seats to the Republicans. From Figure 5.5, we note that 9 seats is not an outlier under the flipped voter preferences. This enables the Republican party to draw the 2016 map in Round 3 and secure 10 seats in Round 4. Thus, we have shown the existence of a successful strategy in our modified model to go from the bipartisan judges' map to the gerrymandered 2016 map which was shown to be an outlier [139].

5.9 Questions Raised in Rucho v. Common Cause

In this section, we address several questions raised during the recent US Supreme Court case Rucho v. Common Cause [7]. All of these issues were dis-

cussed in the opening oral arguments of that case. We consider them as they relate to the regulations and models explored in this paper.

5.9.1 Banning Outliers versus the Proportional Allocation Objective

Skeptics of the outliers metric and banning outliers regulation have argued that it is similar to or a proxy for a proportionality rule. Using basic grid models models, we illustrate where these two rules diverge in valuing and restricting maps.

First, we note that Justices Alito, Gorsuch, Kavanaugh, and Roberts have asked in some way whether a rule which includes outlier detection (among other tests) amounts to a proportionality rule in the oral arguments of [7]. This is not an unreasonable concern. The concept of banning outliers contains important features which are open to manipulation such as

1. Which metric do you use to compare maps (e.g., seat count [139] or variance in the proportions of one party's voters among the districts [138])?

2. How do you set the threshold for what kind of map is an outlier?

3. How do you choose among multiple non-outlier maps which may nevertheless assign more seats to one party or another?

4. How do you define the space of legal maps that you are sampling from? In other words, how closely can your mathematical constraints on what constitutes a legal map approximate the legal definitions in state constitutions?

We show here that despite these concerns, the concept of banning outliers

differs from enforcing or favoring proportionality in important ways. On our 5×5 grid model, we observe what percentage of maps award proportional representation for given sets of preferences. Figure 5.6 captures all possible preference sets on a 5×5 grid with 20 majority voters and 5 minority voters. Figure 5.7 captures all possible preference sets on a 5×5 grid with 15 majority voters and 10 minority voters. It is clear that the seats awarded by the most proportional map can differ greatly from the most likely randomly chosen maps that respect the voter geography. In the case of 20 majority voters, we found that a proportional map was the most likely random map for only 6.9% of the preference sets and for the case 15 majority voters a proportional map was most likely in only 41.5% of preference sets. Comparing Figures 5.6 and 5.7 also suggests that proportional maps are generally more rare when the minority party represents a smaller proportion of the population, while proportional maps are more common for preference sets with more even splits between the parties.

Perhaps the most convincing argument for the difference between outlier detection and proportionality comes in Figure 5.6. We can see that for many preference sets, proportional maps exist, but are rare and could be explicitly forbidden under a regulation banning outliers. In other words, for certain arrangements of voters, proportional maps could actually be outlawed by banning outliers.

5.9.2 Individual Harm

Central to the argument that partisan gerrymandering is unconstitutional is the notion of individual harm. We may ask if an individual's vote was diluted by

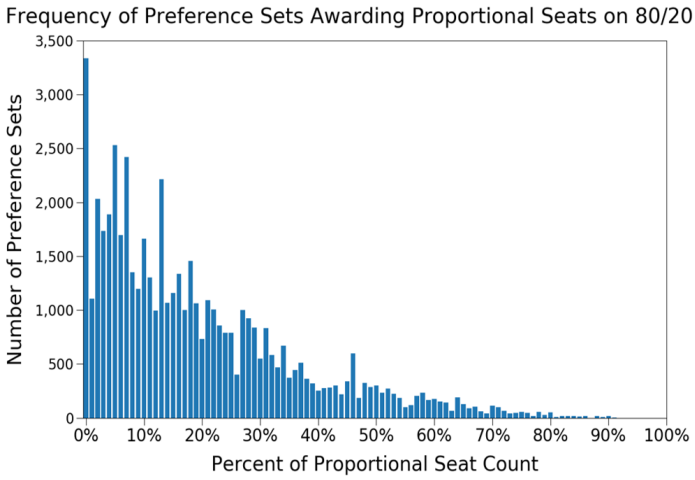


Figure 5.6: Charting all possible preference sets on a 5×5 grid with 20 majority voters and 5 minority voters. We show the numbers of preference sets in which a given percentage of maps achieves proportional representation. The large bar on the left at 0% indicates preference sets where no map awarded proportional representation. The second bar from the left indicates the number of preference sets in which only 1% of maps awarded proportional representation (i.e. preference sets in which proportion maps exist, but are very rare).

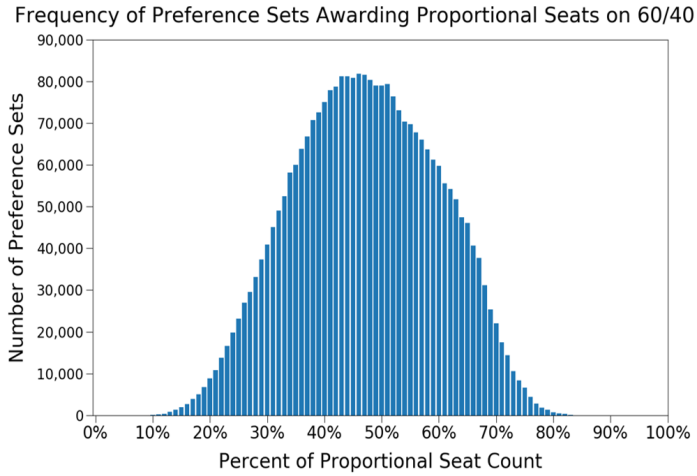


Figure 5.7: Charting all possible preference sets on a 5×5 grid with 15 majority voters and 10 minority voters. We show the numbers of preference sets in which a given percentage of maps achieves proportional representation.

a map based on their political preference. A line of inquiry in [7] focused on the following situation. Suppose we can show that for a given individual, most maps place them in a district where their chosen party wins, but the proposed map is one of a few maps which does not. Can this test be used to show individual harm?

Here, we give evidence of a problem with this test. Suppose there are many such individuals and no single map is fair to all of them. In this case, no deterministically drawn map could be considered fair. However, it could still be argued that a randomly drawn map is fair by this standard (this could trivially be achieved by picking a map uniformly at random from the set of sampled maps).

As evidence of this issue, we consider a 5×5 grid where the majority party has 13 voters in a checkerboard pattern. In this simple example, each voter can be placed in a district where their party wins in over 64% of maps. However, there is no single map which provides this opportunity to all 13 of those voters. Under a philosophy that places value on the most likely maps for a given geography, this raises the question of whether a deterministically drawn district can truly be called fair.

5.10 Conclusion, Recommendations, and Future Directions

We have shown first and foremost that careful scrutiny should be given to any measurement which uses past voter behavior to evaluate and affect the choice of electoral district maps. The model we presented primarily serves to illustrate the basic phenomenon of strategic voting in the presence of gerrymandering regulation.

To better understand this issue, more complex models should be considered.

One of the most unrealistic assumptions in this work is that a political party is able to totally control all of its voters. An obvious concern with the current model is whether it is feasible to organize a large enough group of voters from one party to cast votes for the opposing party in some, but not all elections on a single ballot. Perhaps the only real analog close to this would be the hacking scenario in which people's votes are changed illegally and without their knowledge. Thus, if we wish to consider the possibility of actual strategic voting in practice, we must modify the model in some way. A natural extension would be to consider the realistic influence that political parties do have over a voter's decision between voting for their preferred party or abstaining. It would be interesting to model and explore whether parties can distort gerrymandering measurement by choosing where to spend their limited budgets on "get out the vote" efforts using the power of modern tools such as targeted advertising.

Other useful directions would be to add noise to the model or change the number rounds. The total number of rounds as well as the number of voting rounds between redistricting could be arbitrary. Voter preferences might have some probability to change between rounds. While our four-round model is sufficient to show a basic incentive to vote strategically, richer behavior might evolve in a longer process. For example, majority party voters in our model vote truthfully in Round 4 to capitalize on their choice of a favorable map in Round 3. However, minority party voters might then wish to voter strategically if Round 4 were not the final round.

Regarding the problem of finding strategies to circumvent regulation, the

heuristics presented here were fairly simple and lightweight. It is likely that more sophisticated algorithms and more computing power could be employed to greater effect especially on real data. As with any problem in algorithmic game theory there is the two-fold challenge of figuring out what the optimization problem is as well as how to solve it. We essentially identified one type of strategy and how to execute it. However, there are likely other approaches, especially when considering different models.

Finally, it is worth considering how gerrymandering metrics that use past voting data can be less susceptible to strategizing. We note that a metric which uses voting data from multiple elections would likely be harder to trick. In the US system in particular, data from senate and gubernatorial races could be especially useful since the voters within a state are not partitioned into districts for these elections and that could have a confounding effect on incentives.

Chapter 6: String Algorithms and Bioinformatics

In this chapter, we present our work on string algorithms with applications in bioinformatics and genomics. Section 6.1 presents a tool and data structure, we developed for fast and precise clustering of 16S rRNA gene sequences inspired by the Method of the Four Russians (originally published in [161]). Section 6.2 details a deeper theoretical investigation into how to make the Method of the Four Russians more space-efficient (originally published in [162]). Finally, Section 6.3 describes new algorithms and techniques for the maximum duo-preservation string mapping problem (originally published in [8]).

6.1 16S rRNA Gene Clustering

16S rRNA amplification and sequencing plays a major role in the study of microbiota. This gene codes for the small ribosomal subunit and is highly conserved in bacteria, which makes it an ideal marker gene for taxonomic analysis. However, even a small dataset of 16S rRNA sequencing reads may contain millions of distinct sequences due to both biological processes and errors in sequencing. Thus, it is helpful to cluster this data around a much smaller number of representative sequences for further analysis.

Originally, the pipeline for clustering 16S rRNA gene [163] sequences involved building a multiple sequence alignment of all sequences, computing a pairwise distance matrix of sequences based on the multiple sequence alignment, and clustering this matrix [164]. However, finding the best multiple sequence alignment is computationally intractable and belongs to the class of NP-hard problems [165]. Another natural, but inefficient way of clustering sequences is to perform every possible pairwise comparison to compute a similarity metric such as edit distance and perform hierarchical clustering to merge closely related sequences together. Again, this is inefficient since the resulting running time is guaranteed to be at least quadratic in the total number of sequences.

The need for computational efficiency in many genomic applications moved to the forefront with the development of faster and cheaper DNA sequencing technologies. Currently, metagenomic sequencing datasets can contain over 1 billion short reads [1]. At this scale, the more naive strategies described above can prove to be very expensive and take months to generate clusters. In response, heuristic-based methods like greedy clustering have become commonplace. While some of these methods still have worst case quadratic running time, they can run faster in practice [166–168].

Greedy Clustering. The greedy clustering approach (similar to CD-HIT [166], UCLUST [167], and DNACLUST [168]) can be described at a high level as follows. Let the multiset $\mathcal{S}$ be the set of n sequences to be clustered. Let m be the maximum length of any sequence in $\mathcal{S}$. For simplicity of exposition and analysis, we will assume throughout this section of the proposal that all sequences have length

exactly m. We also assume m is much smaller than n.

First, de-replicate the multiset $\mathcal{S}$ to get the set $\mathcal{U}$ of distinct sequences. Optionally, impose some ordering on $\mathcal{U}$. Then, iteratively remove the top sequence from $\mathcal{U}$ to form a new cluster center s_c. Recruit all sequences $s \in \mathcal{U}$ that are within d distance from s_c. When we recruit a sequence s, we remove it from $\mathcal{U}$ and add it to the cluster centered at s_c. If s_c does not recruit any sequences, we call it a singleton and add it to a list of singletons, rather than clusters. We continue this process until $\mathcal{U}$ is empty.

We order the sequences of $\mathcal{U}$ in decreasing order of their abundance/multiplicity in $\mathcal{S}$. This is also the default ordering used by UCLUST. Alternatively, DNACLUST uses decreasing order of sequence length. The reason for ordering by abundance is that assuming a random error model, the abundant sequences should be more likely to be "true" centers of a cluster. The reason for DNACLUST ordering by length is to preserve triangle inequality among sequences in a cluster when performing semi-global alignment allowing gaps at the end with no penalty. Semi-global alignment is necessary for comparing reads generated by specific sequencing technologies such as 454 (no longer being used). However, since we perform global alignment, triangle inequality is guaranteed regardless of the ordering and thus, ordering by abundance is preferred.

We show that some of these heuristics still struggle under certain conditions by scaling inefficiently and/or producing an inexact greedy clustering. To address this, we developed and implemented a new method for reducing that worst case quadratic running time of exact greedy clustering in practice when the distance metric is the

Levenshtein distance [169] and similarity is determined by a maximum distance of d. Our algorithm improves the speed of the recruitment step wherein we seek all strings within d distance of a chosen center. In addition to promising experimental results, we give slightly weaker, but provable, guarantees for our techniques while many existing methods do not. We also analyze the quality of the clusters output by our method in comparison to the popular greedy clustering tool UCLUST. We show that the clusters we generate can be both tighter and larger due to our method being exact. In other words, our clusters are both lower diameter in terms of the pairwise similarity of all points they contain and at the same time these tighter clusters contain more points. In particular, we observe that our tool outperforms UCLUST when searching for clusters of low diameter which were shown to be ideal in some cases [170].

6.1.1 Related Work

The problem of comparing a query string against a large string database has been widely studied for at least the past twenty years. For similarity metrics like the edit distance, a dynamic programming algorithm [171] can be used to compare two sequences in $O(m^2)$ time, where m is the length of the sequences. When we only wish to identify strings which are at most edit distance d apart, the running time for each comparison can be reduced to $O(md)$ [172] using a modified version of the standard dynamic programming algorithm. This type of sequence alignment is referred to as *banded alignment* in the literature since we only need to consider a diagonal

"band" through the dynamic programming table. The simple dynamic programming approach can also be sped up by using the Four Russians method [173, 174], which divides the alignment matrix into small square blocks and uses a lookup table to perform the alignment quickly within each block. This brings the running time down to $O(m^2 \log(\log(m))/\log(m))$ and $O(m^2/\log m)$ for arbitrary and finite alphabets, respectively. Myers [175] considered the similar problem of finding all locations at which a query string of length m matches a substring of a text of length M with at most d differences. They used the bit vector parallelism in hardware to achieve a running time of $O(mM/w)$ where w is the machine word size. However, when used for clustering sequences, these methods need to perform pairwise comparison of all sequences, thereby incurring the high computational cost of $O(n^2)$ comparisons where n is the total number of sequences.

Sequence search against a database is a crucial subroutine in sequence clustering in general and greedy clustering in particular. However, an interesting property of 16S rRNA gene data is that many of the sequences generated by experiments are highly similar to each other. To exploit sequence similarity and reduce the computation performed in dynamic programming, the DNACLUST [168] algorithm lexicographically sorts the sequences and compares sequences against the center sequence in sorted order. Since the adjacent sequences in sorted order share a long prefix, the part of the dynamic programming table corresponding to their longest common prefix remains unchanged, allowing the "free" reuse of that part of the table for further alignments. This method fails when two sequences differ at the start but are otherwise similar. In this case, the entire dynamic programming table needs to

be recomputed. The UCLUST [167] algorithm uses the USEARCH [167] algorithm to compare a query sequence against a database of cluster centers. However, the algorithm used by UCLUST makes several heuristic choices in order to speed up the calculation of clusters and thus, the resulting clusters are not guaranteed to satisfy any specific requirement. For example, the distances between sequences assigned to the same cluster should ideally satisfy triangle inequality (ensuring that the cluster diameter is at most twice the radius) and the cluster diameters should be both fairly uniform and within the bounds specified by the user.

6.1.2 Distance Metric

We use the same edit distance-based similarity metric as DNACLUST [168], namely

$$\text{similarity} = 1 - \frac{\text{edit distance}}{\text{length of the shorter sequence}}$$

Here, we define edit distance to be Levenshtein distance with uniform penalties for insertions, deletions, and substitutions. The "length of the shorter sequence" refers to the original sequences' lengths without considering gaps inserted by the alignment. We say that two sequences are *similar* if their alignment meets or exceeds a given similarity threshold. Let d be the maximum edit distance between two sequences aligned to the same cluster. This distance is usually computed from a similarity threshold provided by the user, e.g., 97% [61]. Our algorithm performs *banded alignment* with d as the maximum allowable distance. If we determine that two sequences have distance greater than d, we need not report their actual distance.

In the past, a common threshold used was 97% and several works still advocate for this [60]. However, there is evidence recently that higher thresholds at or above 99% are a better choice [170]. Our approach takes the threshold as an input parameter, but is optimized to run at 99%, the default, or higher. This is partially due to the philosophy that it is better to cluster conservatively upfront and merge clusters downstream if desired.

6.1.3 Intervals

Our algorithm involves dividing each sequence into overlapping substrings of length k at regular *intervals*. We formalize the definition of an interval as follows. Given a *period length* p such that $k = p + d + 1$, we divide each sequence into $\lfloor m/p \rfloor$ intervals of length k. For $i \in \{0, 1, \ldots, \lfloor m/p \rfloor - 1\}$, the i^{th} interval starts at index ip inclusive and extends to index $ip + k$ exclusive. We will see in Section 6.1.7 that we must choose p to be at least d. However, choosing a larger p may give a better speedup when dealing with highly similar sequences. Further, for an interval i, we define b_i to be the number of *distinct* substrings for interval i over all sequences in $\mathcal{S}$ and we define $b = \max_i b_i$. We will show in Section 6.1.8 that when b is much smaller than n we get some theoretical improvement on the running time. Figure 6.1 shows an example of how a sequence is partitioned into a set of overlapping substrings. We store these intervals in a data structure we call an *Edit Distance Interval Trie (EDIT)* which is described in detail in Section 6.1.9.

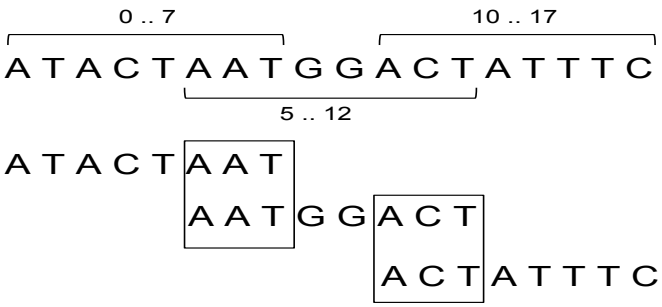


Figure 6.1: An example of how a string is divided in overlapping substrings called intervals. In this case, the length of each substring (k) is 8. Since the substrings must overlap by $d + 1$ characters, which in this case is 3, the period length (p) is 5.

6.1.4 Our Contributions

We developed a method for recruiting in exact greedy clustering inspired by the classical Four Russians speedup. In Section 6.1.5, we describe our algorithm and prove that the worst case theoretical running time is better than naive all-versus-all banded alignment under realistic assumptions on the sequencing data used for clustering. In section 6.1.10, we present experimental results from using our method to cluster a real 16S rRNA gene dataset containing about 2 million distinct sequences. We show that on real data the asymptotic running time of the algorithm grows linearly with the size of the input data. We also evaluated the quality of the clusters generated by our method and compared it with UCLUST, which is one of the widely used methods. We show that our method generates tighter and larger clusters at 99% similarity both when considering edit distance and evolutionary distance. At 97% similarity, we show that the our method produces clusters with a much tighter edit distance diameter compared to UCLUST. While UCLUST runs faster at similarities 97% and less, our approach is faster at higher similarities. In

particular, we highlight that UCLUST does not scale linearly at the 99% similarity threshold while our approach does.

6.1.5 Outline of Techniques and Results

We show two ways in which the classical Four Russians speedup can be adapted to banded alignment and give a theoretical bound for our approach. Then, we describe a trie-like data structure for storing and querying sequences. Finally, we present empirical results in comparison to UCLUST.

6.1.6 Classic Four Russians Speedup

In the classical Four Russians speedup of edit distance computation due to [173, 174], the dynamic programming table is broken up into square *blocks* of size k-by-k as shown in the center of Fig. 6.2. These blocks are tiled such that they overlap by one column/row on each side (for a thorough description of this technique see [176]). When computing banded alignment, we only need to tile the area within the band as in the righthand of Fig. 6.2. Let the maximum edit distance be d and the string lengths be m. Then our block size k can be as small as $d+1$ and we require roughly $2m/k$ blocks in total.

The high level idea of the Four Russians speedup is to precompute all possible solutions to a *block function* and store them in a lookup table (In our implementation we use lazy computation and store the lookups in a hash table instead of precomputing for all inputs). The block function takes as input the two substrings

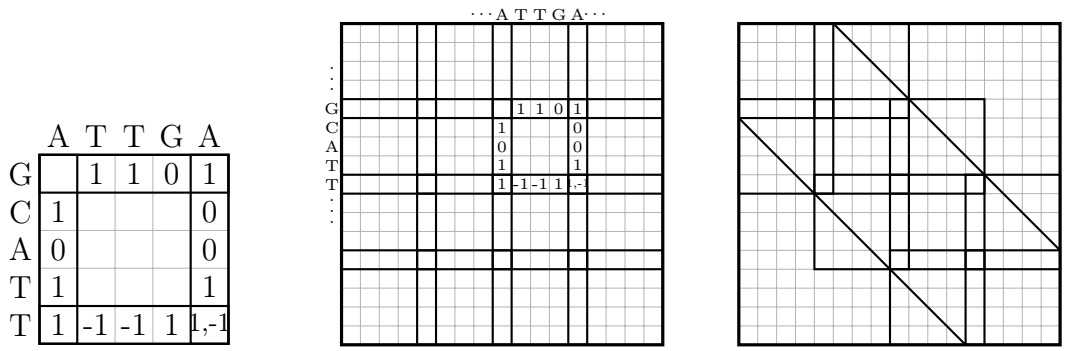


Figure 6.2: Example of classic Four Russians. **Left:** a single block. For any input in the upper left corner, we can sum that value with one path along the edges of the block to recover the value in the lower right corner. The offset value in the lower right corner may be different for the row and column vectors overlapping at that cell. In this example, the lower right cell is one more than its left neighbor and one less than its above neighbor. **Center:** the full dynamic programming table divided into nine 5×5 blocks. The offset values in the example block may not correspond to the optimal alignment of the two substrings shown since they depend on the global alignment between the two full length strings. **Right:** blocks covering only the diagonal band in the context of banded alignment.

to be compared in that block and the first row and column of the block itself in the dynamic programming table. It outputs the last row and column of the block. We can see in the Fig. 6.2 that given the two strings and the first row and column of the table, such a function could be applied repeatedly to compute the lower right cell of the table and therefore, the edit distance. In our application, cells outside the band shown on the right in Fig. 6.2 will not be used since any alignment visiting those cells must have distance larger than d.

There are several tricks that reduce the number of inputs to the block function to bound the time and space requirements of the lookup table. For example, the input row and column for each block can be reduced to vectors in $\{-1, 0, 1\}^d$ (or $\{-1, 0, 1\}^k$ for the general problem). These *offset vectors* encode only the difference between one cell and the next (see Fig. 6.2) which is known to be at most 1 in the

edit distance table. It has also been shown that the upper left corner does not need to be included in the offset vectors. This bounds the number of possible row and column inputs at 3^d (or 3^k in general) each [173]. More generally, when edit costs are derived from a penalty matrix, the number of row/column inputs is bounded by ψ^k where ψ is the number of possible offset values and depends on the penalty matrix. However, we only consider unit costs in this section.

Notice that for the banded alignment problem, this may not provide any speedup for comparing just two strings of length m. Indeed, building and querying the lookup table may take more time than simply running the classical dynamic programming algorithm restricted to the band of width $2d + 1$. However, our final algorithm will do many such comparisons between different pairs of strings using the same lookup table. In practice, we also populate the lookup table as needed rather than pre-computing it. This technique, known as *lazy computation*, allows us to avoid adding unnecessary entries for comparisons that don't appear in our dataset. Additionally, decomposing sequences into blocks will be a crucial step in building the data structure in Section 6.1.9.

6.1.7 Our Approach to the Four Russians Speedup

Notice that the previous approach will not offer much benefit in practice when d is small (e.g. $d = 2$). The overhead of looking up block functions and stitching them together may even be slower than simply running dynamic programming on a block. Further, our dataset may not require us to build a lookup table comparing

all possible strings of length k.

Here, we consider a different block function. This function is designed for situations in which we wish to use a block size k that can be larger than $d+1$. The blocks now overlap on a square of size $d+1$ at the upper left and lower right corners. We will call these overlapping regions *overlap squares*. Our block function now takes as input the two substrings to be compared and the first row and column of the the upper left overlap square. It outputs the first row and column of the lower right overlap square as well as the difference between the upper left corners of the two overlap squares.

Thus, we can move directly from one block to the next, storing a sum of the differences between the upper left corners. In this case, reaching the final lower right cell of the table requires an additional $O(d^2)$ operation to fill in the last overlap square, but this adds only a negligible factor to the running time.

This approach succeeds when the number of possible substring inputs to the block function is limited by some properties of the dataset as opposed to an absolute theoretical upper bound such as $O(|\sigma|^k)$ based on the number of possible strings of length k for an alphabet σ. Rather than computing and storing all possible inputs, we simply store the inputs encountered by our algorithm. The advantage is that a larger block size reduces the number of lookups needed to compare two strings which is $m/(k-d-1)$ for this approach. Naturally, the same tricks such as offset encoding of the input rows and columns as some vector in $\{-1,0,1\}^d$ can be applied.

Another benefit of this approach is that it is more straightforward to implement in practice. Each block depends on the full output of one previous block. In contrast,

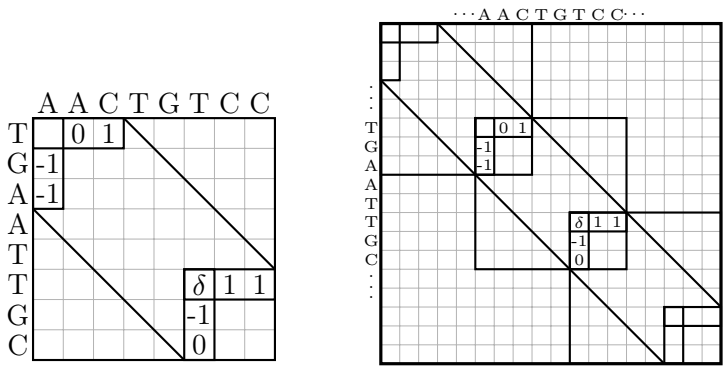


Figure 6.3: Example of our approach to the Four Russians speedup. **Left:** a block for maximum edit distance $d = 2$. The output δ represents the offset from the upper left corner of the current block to the upper left corner of the next block. We only need to consider a diagonal band of the block itself. **Right:** using these blocks to cover the diagonal band of the dynamic programming table for banded alignment.

the classical approach requires combining partial input from two previous blocks and also sending output to two separate blocks.

6.1.8 Theoretical Bound on the Running Time of Our Approach

To give some intuition, we prove a theoretical bound on the running time under the assumption of at most b distinct substrings per interval in the dataset. This is a reasonable assumption for certain application in computational biology. For example, the 16s rRNA gene is highly conserved and thus b is much smaller than n for such datasets. While standard banded alignment takes $O(n^2md)$, we show that for small enough b this can be reduced to $O(n^2m)$. We prove this bound for our approach to using the Four Russians speedup for banded alignment, but it extends to the classical approach as well.

Theorem 19. *If $b \leq \frac{n}{3^d\sqrt{d}}$, we can find all pairs of distance at most d in $O(n^2m)$ time.*

Proof. To simplify, we will assume the lookup table is pre-computed. Then, we must show that if $b \leq \frac{n}{3^d\sqrt{d}}$, then building the lookup table and doing the actual string comparisons can each be done in $O(n^2m)$ time. We further assume $k \approx 2d$ (in practice we choose a larger k).

First, we show that there are at most $\frac{m}{k-d}b^2 3^{2d}$ entries in the lookup table. There are at most $\frac{m}{k-d}$ intervals and since each interval has at most b distinct strings, there are at most b^2 relevant string comparisons. Each distinct string comparison must be computed for all 3^{2d} offset vector inputs. The cost of generating each lookup entry is simply the cost of computing banded alignment on a block, kd. Thus, the lookup table can be built in time $\frac{m}{k-d}b^2 3^{2d}kd$. Keeping our goal in mind we see that

$$\frac{m}{k-d}b^2 3^{2d}kd \leq n^2 m \qquad \text{is true when} \qquad b \leq \frac{n}{3^d\sqrt{d}} \qquad \text{since } k \approx 2d$$

To bound the running time of the string comparisons, notice that comparing two strings requires computing $\frac{m}{k-d}$ block functions. The time spent at each block will be $O(k+d)$ to look up the output of the block function and update our sum for the next corner. Thus, building the lookup table and computing the edit distance between all pairs using the lookup table each take $O(n^2m)$ time. □

6.1.9 The Edit Distance Interval Trie (EDIT)

To facilitate the crucial step of identifying all strings within edit distance d of a chosen cluster center, we construct a trie-like data structure on the intervals. This structure will be built during a pre-processing stage. Then, during recruitment, any

recruited sequences will be deleted from the it. The procedure for building this structure is summarized in Algorithm 6 and illustrated in Figure 6.4. The main benefit, like any trie, is that it exploits prefix similarity to avoid duplicating work.

The mapping in step 2 of Algorithm 6 is a one-to-one mapping to integers from one to the number of distinct substrings. It reduces the size of the data structure since the number of distinct substrings will typically be much less than all possible length k strings on the given alphabet. This mapping also speeds up calls to the lookup table during the recruitment subroutine summarized in the next section.

Algorithm 6: BUILD-EDIT

1. Partition each sequence into overlapping intervals of length k, such that each interval overlaps on exactly $d+1$ characters.
2. Map each distinct substring of length k appearing in our list of interval strings to an integer.
3. Assign an integer vector *signature* to each sequence by replacing each block with its corresponding integer value.
4. Insert these signatures into a trie with the leaves being pointers to the original sequences.

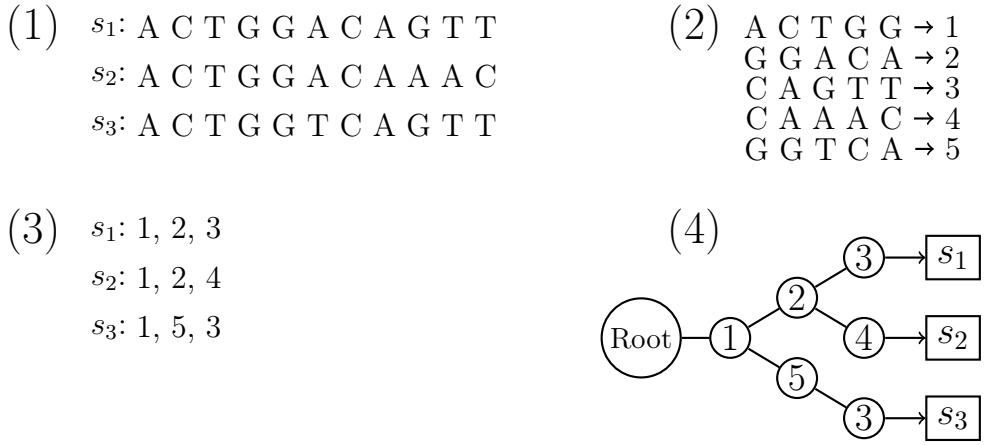


Figure 6.4: Example illustrating the steps of Algorithm 6 with $d = 1$ and $k = 5$.

6.1.10 Experimental Results

Properties of our recruitment algorithm and data structure

In this section, we highlight some key features of our recruitment algorithm and the EDIT data structure. To evaluate our method, we used a dataset consisting of about 57 million 16S rRNA amplicon sequencing reads with 2.7 million distinct sequences. To understand the impact of the number of input sequences to cluster on the average number of comparisons in each recruitment step, we ran our algorithm on different input sizes at different similarity thresholds. We counted the average number of tree nodes explored while recruiting a particular center sequence and used it as a quantitative representation of the amount of comparisons made since all nodes represent a substring of fixed length k. Figure 6.5 shows the plots for the average number of tree nodes explored for different similarity thresholds. For the 95% and 97% similarity thresholds, the average number nodes explored decreases as more sequences are clustered. This happens because of the fact that although more sequences are clustered, due to the lower similarity threshold a large number of sequences get clustered in each traversal of the tree. For 99% similarity threshold, the average number of nodes explored increases initially with the number of sequences, but becomes uniform after about $100,000$ sequences. The strict increase in the number of nodes can be explained by the high similarity threshold. However, in all cases, the number of nodes explored by each center does not increase linearly with the number of input sequences. Thus the total number of comparisons made for given dataset is observed to be increasing as function of n rather than the worst

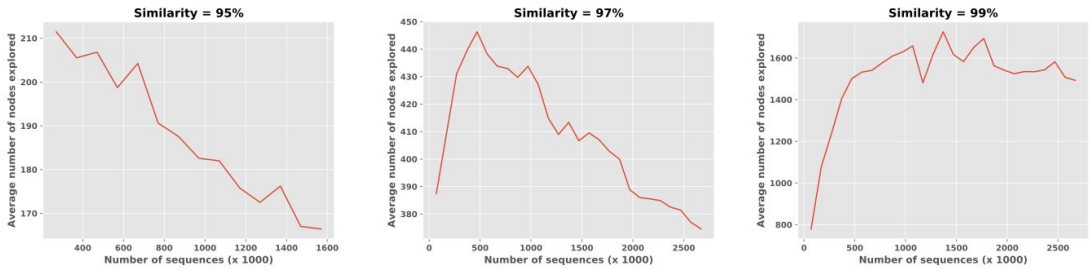


Figure 6.5: Plots for the average number of nodes explored in the tree while recruiting sequences to a cluster center.

case n^2.

Running time analysis

We compared the running time of EDIT and UCLUST on a subsample 1.07 million distinct sequences at different similarity thresholds. Figure 6.6 shows the plot for running time at different similarity thresholds. We observed that the running time of EDIT stays fairly constant at different similarity thresholds whereas the running time of UCLUST was very low for lower similarity thresholds, but increased non-linearly at higher similarity thresholds. Especially, between 98.5% to 99%, the running time of UCLUST grows 5 folds. We did further analysis of running time at 99% similarity threshold using different sample sizes as input. Figure 6.6 shows the running time comparison of UCLUST and EDIT. It can be observed that, the running time of UCLUST on large sample sizes ($>$ 1 million) grows much faster that the running time of EDIT, which scales almost linearly. For the largest sample of 2.7 million sequences, UCLUST running time was ten times greater than EDIT running time. This evaluation implies that higher similarity thresholds ($>$ 98%), EDIT was faster compared to UCLUST. Also, the running time of EDIT showed

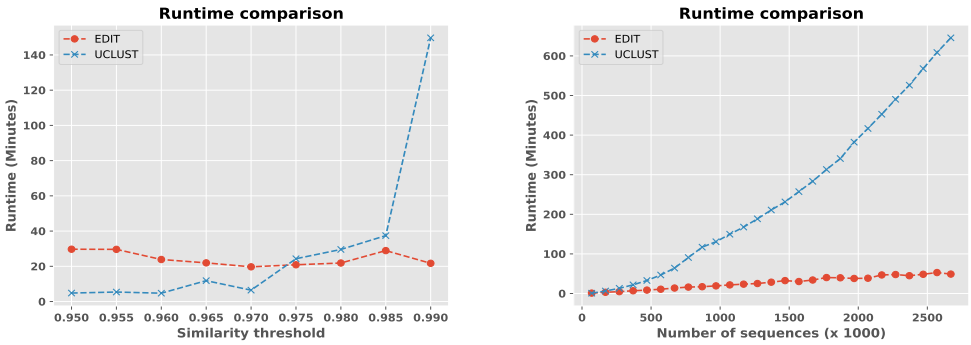


Figure 6.6: Running time comparison of EDIT and UCLUST as a function of similarity threshold and number of sequences.

low variance compared to UCLUST for different similarity thresholds.

Evaluation of clusters

We subsampled 135,880 distinct sequences from the entire dataset and ran both methods at the 97% and 99% similarity thresholds. We then compared the outputs of both methods using three metrics: the cluster size, the cluster diameter based on the sequence similarity, and the cluster diameter based on the evolutionary distance. To compute the cluster diameter based on sequence similarity, we computed the maximum edit distance between any two sequences in each cluster. To compute the cluster diameter based on evolutionary distance, we first performed a multiple sequence alignment of the sequences in each cluster using clustalW [177]. Once the multiple sequence alignment was computed, we used the DNADIST program from the phylip [178] package to compute a pairwise evolutionary distance matrix. The maximum distance between any pair of sequences is defined as the DNADIST diameter. Using two orthogonal notions of cluster diameter helps to define the "tightness" of clusters. Figures 6.7 and 6.8 show violin plots for different comparison

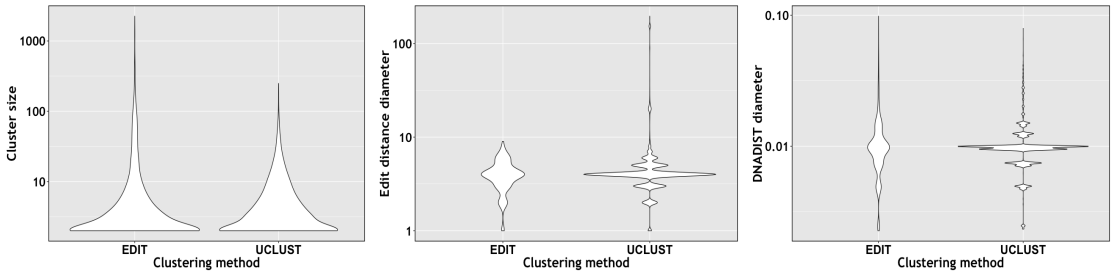


Figure 6.7: Evaluation at similarity threshold of 99%. All of the plots are log scaled.

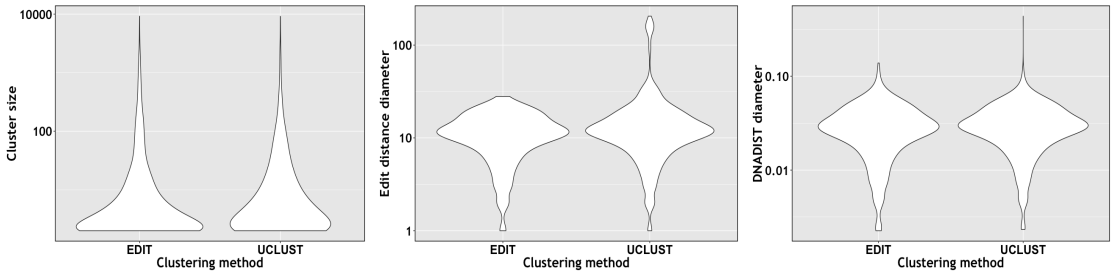


Figure 6.8: Evaluation at similarity threshold of 97%. All of the plots are log scaled.

metrics at the 99% and 97% similarity thresholds, respectively. At the 99% similarity threshold, EDIT is able to produce larger clusters compared to UCLUST. The edit distance diameters for the clusters generated by EDIT is fairly well constrained. However, the edit distance diameters for the clusters generated by UCLUST had a large variance, implying that several dissimilar sequences may be getting clustered together. The DNADIST diameter for both methods was comparable. At the 97% similarity threshold, both EDIT and UCLUST generated similar sized clusters. Even in this case, the edit distance diameter for UCLUST clusters showed a larger variance compared to the edit distance diameter for EDIT clusters. The DNADIST diameter for UCLUST has slightly more variance compared to that of EDIT clusters, implying some of the clusters generated by UCLUST had sequences with a large evolutionary distance between them. This validation confirms that the sequences in the clusters produced by EDIT at different similarity thresholds are highly similar to each other.

At the 99% similarity threshold, we observed a stark difference between the cluster sizes of EDIT and UCLUST. For example, the two largest clusters produced by EDIT had sizes 7,978 and 3,383 respectively whereas the two largest clusters produced by UCLUST were of sizes 249 and 233, which is almost 30 times smaller than the largest EDIT cluster. To investigate this further, we used BLAST [179] to align all clusters of UCLUST against the top two largest clusters of EDIT. We only considered the alignments with 100% alignment identity and alignment coverage. We observed that 765 distinct UCLUST clusters had all of their sequences aligned to the largest EDIT cluster and 837 distinct clusters had at least 80% of their sequences aligned to the largest EDIT cluster. Only 82 UCLUST clusters out of 16,968 total (not including singletons) had less than 80% of their sequences mapped to the largest EDIT cluster. Those 82 clusters accounted for only 255 sequences, roughly 30 times fewer than the number of the sequences in the largest EDIT cluster alone. As far as singletons (the clusters with only one sequence) are concerned, EDIT generated 22,318 singleton clusters whereas UCLUST generated 33,519 singleton clusters. For the size of the sample considered in this analysis, this difference is very significant. This evaluation implies that at a high similarity threshold, heuristic based methods like UCLUST tend to produce fragmented clusters whereas EDIT was able to capture a higher number of similar sequences in a single cluster.

6.2 Succinct Four Russians Speedup for Edit Distance

Edit distance (a.k.a. Levenshtein distance) is one of the most natural and ubiquitous measures of similarity between two strings. In the most common variant, *unit cost*, it counts the minimum number of edits needed to transform one string into another. Here, we use the Levenshtein definition of *edits* which include insertions, deletions, or substitutions of a single character. However, in some cases edit operations may be assigned differring costs from a penalty matrix and additional operations (e.g. inversions or transpositions) may be considered. Computing this distance is a fundamental problem with applications in many areas such as computation biology, natural language processing, and information theory.

The most well known algorithms use dynamic programming to solve the problem in $O(m^2)$ time where m is the length of the strings. The only improvement to this has been the Four Russians algorithm [173], running in $O(m^2/\log m)$ time. While the conditional hardness results, such as [180], suggest this is unlikely to be improved further for arbitrary strings even on small alphabets [181].

The problem of comparing a string against a large set of sequences is of central importance in domains such as computational biology, information retrieval, and databases. The banded alignment variant (a.k.a. the d differences approximate string matching problem), in which we only report the distance when it is at most some parameter d is also highly relevant. It is useful in numerous settings wherein we only care about finding small distances or the maximum distance between any two strings in known to be small. As stated in the previous section, solving this

problem is a key subroutine in many *greedy* clustering heuristics for gene clustering.

Another area of research surrounding the Four Russians speedup is how to apply it in practice. While the theoretical result uses a block size of $\log m$, such a large block size is impractical due the size of the lookup table exceeding hardware constraints. For the unit cost version, [182] showed how to drastically reduce the required space, especially for large alphabets, by avoiding redundant string comparisons. We show that our approach can be combined with theirs to reduce the space (and preprocessing time) requirement even further.

6.2.1 Related Work

The edit distance problem is extremely well-studied and the following related work is by no means exhaustive. We focus on the aspects most related to our problem: pairwise comparison, the Four Russians speedup, and one-against-many comparison. For simplicity, we describe all results in the context wherein all strings have length exactly equal to m.

The most well-known approach for computing the edit distance between a pair of strings of length m uses dynamic programming and requires $O(m^2)$. This was later improved to $O(m^2/\log m)$ in 1980 using the Four Russians speedup [173, 174] and [183] achieved $O(m^2/\log m)$ for unrestricted scoring matrices. The Four Russians speedup, originally proposed for matrix multiplication, has been adapted to many problems besides edit distance including: RNA folding [184], transitive closure of graphs [185], and matrix inversion [186]. On the negative side, [180]

recently showed that no algorithm for edit distance can do better than $m^{2-\epsilon}$ time unless the Strong Exponential Time Hypothesis (SETH) is false and [181] extended this to include strings on a binary alphabet. They accomplished this by reducing a satisfiability problem to edit distance and showing that a subquadratic algorithm for edit distance implies a subexponential algorithm for satisfiability. However, if we fix a maximum distance d and only care about reporting the exact distance when it's less than d, we call this the *banded alignment* problem. This problem has seen improvements to $O(md)$ time [187] and the current best algorithm takes only $O(m + d^2)$ time [188, 189].

One-against-many edit distance comparison involves comparing a single string to a set of n other strings. Here, we consider only the banded alignment version of the problem wherein we seek to find the distance to all strings within maximum distance d. This problem can be solved in $O(nm + nd^2)$ or $O(nmd)$ time by iteratively applying the pairwise banded alignment algorithms discussed above. Heuristic approaches may run much faster in practice by exploiting properties of the input strings such as prefix similarity and storing the set of strings in a clever data structure such as a trie or BK-tree [187]. However, little theoretical progress has been made. A popular approach to this problem in the context of spell checkers employs Levenshtein automata and/or transducers [190–192]. Assuming d is a fixed constant, these algorithms run in $O(nm)$ time. However, in practice they consider extremely small values of d (at most 3 or 4) and their runtime appears to grow exponentially in d. In the context of gene clustering in computational biology, [161] show that all pairs banded alignment can be performed in $O(n^2m)$ time under the

assumption that all strings are extremely similar. They also use an extension of the Four Russians speedup to one-against-many banded alignment, but our approach to this problem requires no assumptions on the input strings.

The Four Russian speedup is well-studied in context of the regular expression membership problem where the goal is to determine if a particular string matches a given regular expression. Myers [193] showed that for a regular expression of length P and a string of length m, the exact regular expression membership problem (no mismatches are allowed) can be solved in $O(mP/\log m)$ time using the Four Russian speedup compared to the naive $O(mP)$ runtime. Wu, Manber, and Myers [194] extended this result for approximate regular expression membership problem where the goal is to check if a string is within an edit distance d from the given regular expression. They showed that approximate regular expression matching problem can be solved in $O(mP/\log_{d+2} m)$ time.

Space efficiency is also a major concern in practical applications of the Four Russians speedup since the entire lookup table must be stored in main memory. Thus, block sizes as small as $k = 4$ or 5 may be used. The classical approach for the unit cost variant uses $O(3^{2k}k|\Sigma|^{2k})$ space. Kim, Na, Park, and Sim [182] showed how to remove the dependence on the alphabet size, generating a lookup table in $O(3^{2k}(2k)!k^2)$ time and $O(3^{2k}(2k)!k)$ space. This offers a significant improvement, for example, when $|\Sigma| = 20$ for protein sequences or $|\Sigma| = 26$ for the English language.

6.2.2 Preliminaries

For simplicity of presentation, we assume all strings have equal length m. However, the results extend easily to the case where strings have different lengths. We assume the lookup table is any data structure that can perform lookups and insertions in $O(k)$ time for blocks which are identified by distinct keys of length $O(k)$.

See Section 6.1.6 for a summary of the classical four Russians approach.

6.2.3 Our Contributions

We show a new way to store and query block functions. For a given pair of strings corresponding to a k-by-k block in the dynamic programming table, we store an entry in the lookup table using only $O(k^2 \lg k)$ time and $O(k^2)$ space. We show how to query this entry in $O(k)$ time. By contrast, the classical approach requires $O(\psi^{2k}k^2)$ time and $O(\psi^{2k}k)$ space, where ψ is the number of possible offset values and depends on the costs of edits, to store a lookup entry for a pair of strings since it computes the function for all possible row/column offset vectors and $O(k)$ time per query. Thus, we improve the time and space complexity of that aspect by a factor of at least ψ^{2k}/k and remove the dependence on ψ. This result is stated in Theorem 20.

Theorem 20. *Given two strings corresponding to a k-by-k block, we can store a lookup entry using $O(k^2 \lg k)$ time and $O(k^2)$ space such that given any values for the first row and column of the block, we can compute the last row and column of*

the block in $O(k)$ time.

We demonstrate the power of our technique for block functions by designing an algorithm for the fundamental problem of one-against-many banded alignment. In particular, comparing one string of length m to n other strings of length m where we only need to report distances within a maximum distance threshold d can be performed in $O(nm + md^2 \lg d + nd^3)$ time. When d is reasonably small, this improves on the common, naïve approach which requires $O(nmd)$ time to iteratively run an $O(md)$ time pairwise banded alignment algorithm. It also approaches the best theoretic result of $O(nm+nd^2)$ achieved by using the best known pairwise algorithm running in $O(m+d^2)$ time [188,189]. We note that the author of [188], describes the $O(m + d^2)$ time algorithm as "impractical" and "primarily of theoretical interest". We are somewhat more optimistic, observing that our algorithm blends neatly with approaches such as those described in the previous section on gene clustering and as discussed in Section 6.2.6, can be implemented in a way that exploits the prefix similarity occurring in practice.

Theorem 21. *Performing banded alignment with maximum distance d between a string of length m and n other strings also of length m can be done in $O(nm + md^2 \lg d + nd^3)$ time.*

We extend the classic result of [173] which computes the edit distance between two strings in $O(m^2/\log m)$ time to remove the dependence on ψ even when edits have costs derived from a penalty matrix. Here, the number of entries in the lookup table does not depend on the penalty matrix. We acknowledge that [183]

also achieves the same $O(m^2/\log m)$ running time on unrestricted scoring matrices. However, there are some differences between our approach and theirs which may make one or the other more advantageous in different settings. Most notably our approach adheres more closely to the classic Four Russians speedup and uses a uniform block size which is necessary for our one-against-many algorithm. Uniform block sizes also allow our technique to be combined easily with the space-efficient approach in [182] and the gene clustering technique in [161] since both rely on splitting the dynamic programming table into uniform size blocks. For [161], this is crucial to exploiting the prefix similarity among highly conserved genomic sequences. On the other hand, the blocks in [183] vary in size in a clever way to take advantage of the compressibility of the strings being compared. This yields a faster running time for pairwise comparison of strings with small entropy, $O(hn^2/\log n)$, where $h \leq 1$ is the entropy of the text.

Theorem 22. *Given a penalty matrix for edit operations, the edit distance between two strings can be computed in $O(m^2/\log m)$ time.*

In practical applications wherein space efficiency is important and smaller block sizes k are used (notably $k < |\Sigma|$), [182] showed how to remove the dependence on the alphabet size for the unit cost version, generating a lookup table in $O(3^{2k}(2k)!k^2)$ time and $O(3^{2k}(2k)!k)$ space. Combining their work with ours yields an improvement to $O((2k)!k^2 \lg k)$ time and $O((2k)!k^2)$ space. Figure 6.9 illustrates the differences in space efficiency achieved by each approach for small block sizes k.

Theorem 23. *For a block size k, a lookup table can be generated in $O((2k)!k^2 \lg k)$*

time and $O((2k)!k^2)$ *space such that we can find the unit cost edit distance between two strings of length* m *in* $O(m^2/k)$ *time.*

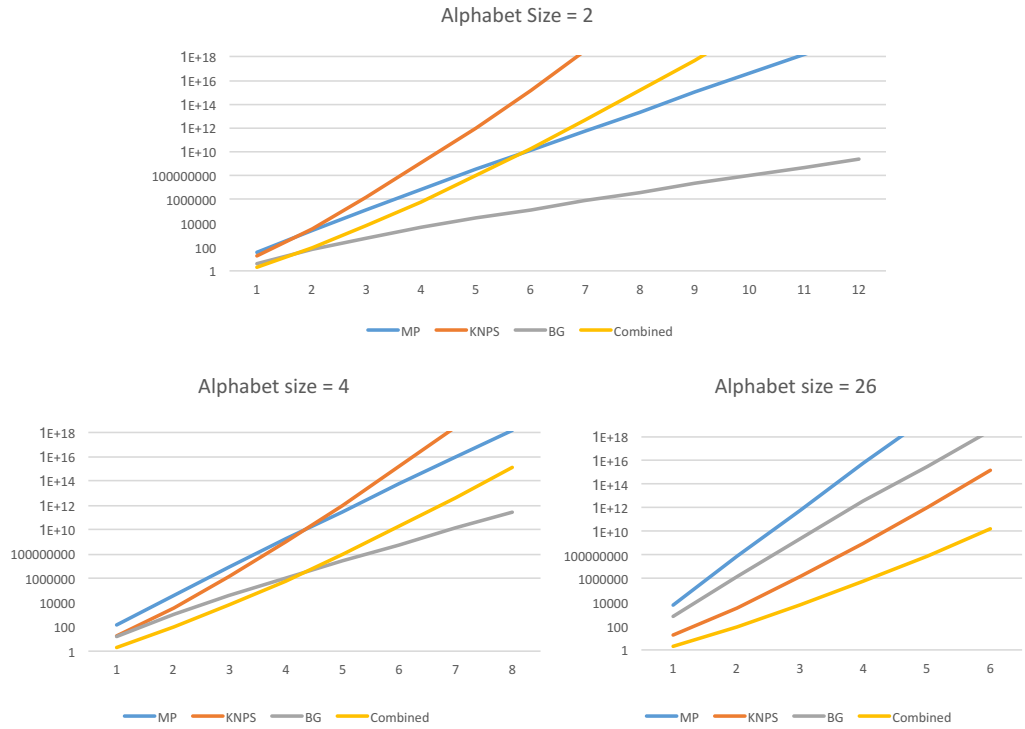


Figure 6.9: Plots showing the theoretical space efficiency for different block sizes k on alphabet sizes 2, 4, and 26. MP is the classic approach [173], KNPS is the space efficient version from [182], BG is our approach, and Combined is the combination of our method and [182].

6.2.4 Storing and Querying the Block Function

Here, we consider the crucial subroutine in our algorithms and prove Theorem 20. For a block size k, we first show how to store a lookup entry for any two strings of length k in $O(k^2 \lg k)$ time and $O(k^2)$ space. Then, we show how, given two strings of length k and the first row and column of the block, we can compute

the last row and column in $O(k)$ time by querying the corresponding lookup entry. Notice that in contrast to the classical Four Russians speedup, the information we precompute and store for a block function is based only on the two strings being compared. Thus, we avoid having to store an entry for each of the 3^{2k} possible input vectors considered in [173] (For unit costs, they encode rows/columns as offset vectors in {-1, 0, 1} since the values in adjacent cells differ by at most 1, yielding 3^k possible inputs each for the row and column vectors).

Notation

We start by defining some notation, illustrated in Figure 6.10. Let $U = \{u_1, u_2, \ldots, u_{2k-1}\}$ be an ordered set of the cells in the first row and column of the block and let $V = \{v_1, v_2, \ldots, v_{2k-1}\}$ be an ordered set of the cells in the last row and column of the block. For both sets, the ordering starts with the upper right corner and ends in the lower left corner. Thus, both u_1 and v_1 correspond to the upper right corner, u_k corresponds to the upper left corner, v_k corresponds to the lower right corner, and both u_{2k-1} and v_{2k-1} correspond to the lower left corner.

For each pair of cells (u, v), we store the least cost $c_{u,v}$ of any path through the block from u to v. If no such path exists, we set $c_{u,v} = \infty$ and if u and v correspond to the same cell, we set $c_{u,v} = 0$. Note that $c_{u,v}$ is not necessarily based on the optimal alignment within the entire block. It corresponds to an alignment of the subset of the block with u as the upper left corner and v as the lower right. Also, recall that this block will be part of a larger dynamic programming table and the

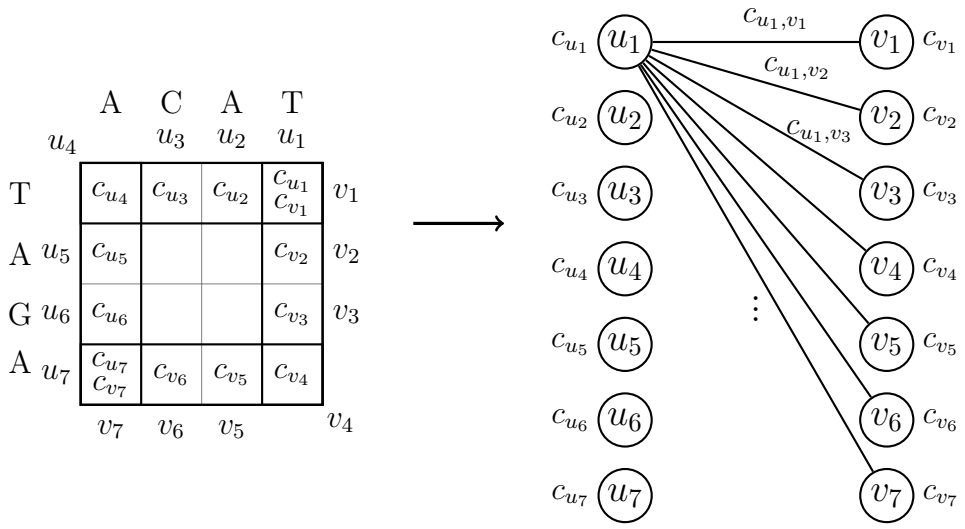


Figure 6.10: Illustration of how the dynamic programming table is represented as a bipartite graph of least cost paths. **Left:** The dynamic programming table for a block comparing the strings "ACAT" and "TAGA" with all u, v, c_u, and c_v labeled. **Right:** The bipartite graph representation. Note that this will be a complete, weighted bipartite graph with costs $c_{u,v}$ for all pairs in $U \times V$.

path through the block corresponding to the best global alignment may not be the same as the path corresponding to the best local alignment within the block.

We can think of this set of costs as a complete, weighted bipartite graph $G = \{U, V, U \times V\}$ with weights $c_{u,v}$ on the edges. We also use c_u and c_v to denote the values stored in the corresponding cells of the block within the dynamic programming table. When we query a block function for two strings, the c_u values (input row and column) will be given as input and our goal will be to compute the c_v values (output row and column). Thus, if we consider the values stored in the cells after the full dynamic programing table has been computed, we have that $c_v = \min_{u \in U}(c_u + c_{u,v})$.

Storing Lookup Entries

For every pair of substrings we wish to query eventually, our lookup table will simply store the cost c_{uv} for every edge in the graph G defined by comparing those substrings. These cost values will be stored in a $|V| \times |U|$ matrix M with a row for each $v \in V$ and a column for each $u \in U$. Cell M_{ji} will contain $c_{u_i v_j}$. We now show that computing G and storing M for any pair of substrings of length k can be done in $O(k^2 \lg k)$ time.

Lemma 15. *Given a pair of strings of length k, we can compute all $c_{u,v}$ in $O(k^2 \lg k)$ time.*

Proof. Note that each $c_{u,v}$ can be seen as the weight of the shortest path in a grid graph of dimension $k \times k$. Thus the algorithm of [195] can be applied. That algorithm requires $O(k^2 \lg k)$ preprocessing time and can then compute each of the $O(k^2)$ $c_{u,v}$ entries in $O(\lg k)$ time. This leads to an overall running time of $O(k^2 \lg k)$.

□

For completeness, we also state the simple fact that the space requirement for an entry is $O(k^2)$.

Lemma 16. *Given a pair of strings of length k, storing the entry requires $O(k^2)$ space.*

Proof. The proof follows directly from the fact that we are simply storing the edges of a complete, weighted bipartite graph $G = \{U, V, U \times V\}$ with $|U| = |V| = 2k - 1$. □

Querying a Block Function

Given the two substrings and the input row and column vectors, we now show how to use our lookup entry matrix M to compute the output row and column (a.k.a all c_v for $v \in V$) in $O(k)$ time.

Lemma 17. *Given the input row and column vectors and the $O(k) \times O(k)$ lookup entry matrix M, we can compute the output row and column in $O(k)$ time using the SMAWK algorithm [196].*

Proof. Let $\vec{w}$ be the vector of all c_u values generated from the input row and column vectors. Scaling each column of M by the corresponding cell in $\vec{w}$ gives us a new matrix M' wherein the minimum value in each row j is our desired output value $c_{v_j} = \min_{u \in U}(c_u + c_{u,v_j})$. It is known that M' is totally monotone [195, 197] and thus we can find row minima in $O(|U|) = O(k)$ time using the classic SMAWK algorithm [196]. Note that we need not explicitly generate M' since the value of any cell we wish to query can be computed from M and $\vec{w}$ as $M'_{ji} = M_{ji} + \vec{w}_i$. □

The proof of Theorem 20 follows from Lemmas 15, 16, and 17.

Alternatives to Query a Block Function without SMAWK

While our algorithm for banded alignment in Section 6.2.5 uses larger block sizes than the typical pairwise Four Russians approach, in many cases, the blocks will be small enough for SMAWK to be inefficient in practice. As such, we introduce a simpler query algorithm here and briefly discuss the potential for future work to

speed up the query function in practice.

This simpler query algorithm achieves a slightly worse asymptotic running time of $O(k \lg k)$ and can be described as follows. Recall that our goal is to find the minimum value of each row in the totally monotone matrix M' with $|U|$ columns and $|V|$ rows. We first find the minimum value in row $|V|/2$ and let *mincol* be the column containing that cell. We then perform the same operation recursively on two submatrices of M'. The first submatrix includes all rows up to $|V|/2$ and all columns up to (and including) *mincol*. The second includes the rows after $|V|/2$ and columns from *mincol* to $|U|$. We do not claim this simpler algorithm is a novel approach to finding row minima and include it merely to illustrate possible alternatives to SMAWK.

Lemma 18. *The algorithm described here runs in $O(k \lg k)$ time and outputs the correct result.*

Proof. For the running time, note that each recursive call nearly partitions the columns of M' (pairs of submatrices overlap at single columns), resulting in $O(|U|) = O(k)$ time spent at each level of recursion. Since we split the rows in half at each level, there will be $O(\lg |V|) = O(\lg k)$ levels total, giving a final running time of $O(k \lg k)$.

The correctness follows directly from the properties of totally monotone matrices also utilized in the analysis of SMAWK. □

Looking to the future, we note that neither SMAWK nor the algorithm in this section leverage all of the specific properties of the matrix M'. For example, M' is

not an arbitrary totally monotone matrix. It comes from M, a matrix which we can afford to spend k^2 time preprocessing, scaled by $\vec{w}$, a vector with the property that adjacent cells differ by at most 1 in the unit cost setting.

6.2.5 One-against-many Comparison

Extending the Four Russians Approach to Banded Alignment

For our algorithm for one against many banded alignment, we use the extension to banded alignment from [161] which simplifies both the analysis and practical implementation. The extension uses a slightly different block function and way of tiling blocks to cover the relevant diagonal "band" of the dynamic programming table. The blocks now overlap on a square of size $d + 1$ at the upper left and lower right corners. We will call these overlapping regions *overlap squares*. The block function still takes as input the two substrings to be compared. The set U contains only the first row and column of the the upper left overlap square and V contains only the first row and column of the lower right overlap square as well as the difference between the upper left corners of the two overlap squares.

Thus, we can move directly from one block to the next, storing a sum of the differences between the upper left corners. In this case, reaching the final lower right cell of the table requires an additional $O(d^2)$ operation to fill in the last overlap square, but this adds only a negligible factor to the running time.

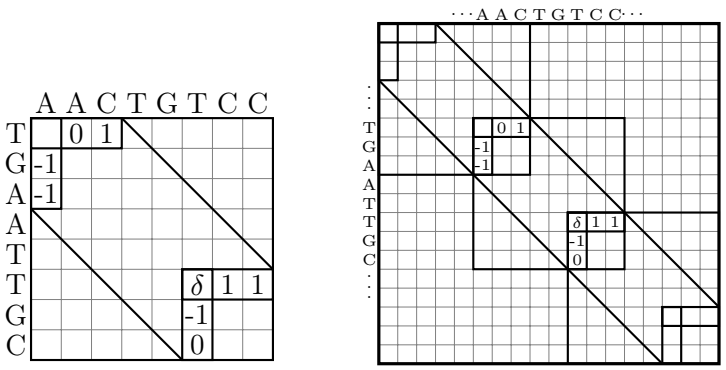


Figure 6.11: Example of our approach to the Four Russians speedup. **Left:** a block for maximum edit distance $d = 2$. The output δ represents the offset from the upper left corner of one block to the upper left corner of the next block. Note that we only need to consider a diagonal band of the block itself. **Right:** using these blocks to cover the diagonal band of the dynamic programming table in the context of banded alignment.

Our Algorithm

We start with some notation and definitions. For a string s, let $s_{i,i+k}$ be a length k substring starting at index i of s. We define two types of block comparisons, *identities* and *differences*, based on the strings being compared. An identity comparison is between the substring $s_{i,i+k}$ and another substring that is identical to one of the substrings $s_{j,j+k}$ for $j \in \{i-d, i-d+1, \ldots, i, \ldots, i+d\}$. All other comparisons are difference comparisons. In other words, identity comparisons between two strings will come from long common subsequences between the two strings. Difference comparisons will come from the locations where an edit occurs. Note that we can stop comparing two strings once we've encountered more than d differences among their prefixes. Let S be a set of strings and let p be the single string we wish to compare to all strings in S.

The algorithm can be summarized as follows. We first compute and store

lookup entries for all possible identity comparisons for each block in p. We then perform pairwise comparisons between p and each string in S. A pairwise comparison is computed as follows. For each block, we first query the lookup table using the corresponding substrings. If we find an entry (similarity comparison), we query it as described in Section 6.2.4. Otherwise (difference comparison), we perform standard banded alignment on the two strings with the first row and column of the table initialized to the values of the input row and column of the block. If at any time during a pairwise comparison the distance accumulated exceeds d, then we immediately halt and move on to the next pair.

We divide the analysis into three parts: the time to compute and store the lookup table, the time to query the lookup table during pairwise comparison, and the time to compute the block function for difference comparisons.

Lemma 19. *The time to compute and store the lookup table for all block identity comparisons in a single string p of length m and max distance d is $O(md^2 \lg d)$.*

Proof. Let the block size $k = 2d$. Then p will be divided into $m/d-1$ blocks. For any given block, let $p_{i,i+k}$ be the substring of p corresponding to that block. Then, for every $j \in \{i-d, i-d+1, \ldots, i, \ldots, i+d\}$, we need to store the comparison between $p_{i,i+k}$ and $p_{j,j+k}$. We need not compare $p_{i,i+k}$ to any substrings outside this range since that would imply an alignment of distance greater than d. Thus, for each block we need to store lookups for at most $2d+1 = O(d)$ different identity comparisons. Computing the lookup entry for each comparison takes $O(k^2 \lg k) = O(d^2 \lg d)$ time by Theorem 20. Putting it all together, we have $O(m/d \cdot d \cdot d^2 \lg d) = O(md^2 \lg d)$. □

Lemma 20. *Excluding the time to compute block functions for difference comparisons, the time to compare a string p of length m to n other strings using the precomputed lookup table is $O(nm)$.*

Proof. Each pairwise comparison involves computing $m/d - 1$ block functions. If a block corresponds to an identity comparison querying the block function takes $O(k) = O(d)$ time by Theorem 20. Otherwise, if it's a difference comparison block, the only time will come from checking the lookup table which we've assumed takes $O(d)$ time. It follows that the running time for each pairwise comparison is $O(m)$ and comparing p to all n strings requires $O(nm)$ time. □

Lemma 21. *The time needed to compute block functions for difference comparisons between p and all n other strings is $O(nd^3)$.*

Proof. Notice that each edit is present in at most two overlapping blocks. It follows that for a given pair of strings, the number of block queries corresponding to differences can be at most $2(d + 1) = O(d)$ since we will halt a comparison if the distance ever reaches $d + 1$ or more. Thus, the running time to compute the full dynamic programming for difference blocks for all n pairwise comparisons is $O(n \cdot d \cdot d^2) = O(nd^3)$. □

The proof of Theorem 21 follows from combining Lemmas 19, 20, and 21.

6.2.6 Extensions and Applications

In this section, we briefly show how the results of Section 6.2.4 can be applied to other settings in which the Four Russians speedup is used for computing string

edit distance.

Comparing Two Arbitrary Strings with a Penalty Matrix

As with the classical Four Russians, when the alphabet size is constant, we can choose the block length k to be an appropriate logarithmic function of the string length m such that the lookup table can be computed efficiently. For an alphabet Σ, there are $|\Sigma|^{2k}$ pairs of string of length k. By Theorem 20, each pair requires $O(k^2 \lg k)$ time to compute the lookup entry regardless of the costs of the edits. Thus, the preprocessing for $k = (\log_{|\Sigma|} m)/2$ takes $O(m \log^2 m \log\log m)$ time. Since the total number of blocks in the dynamic programming table is $O(m^2/k^2)$ and computing each block function from the lookup table takes $O(k)$ time by Theorem 20, the running time to compute the distance using the lookup table is $O(m^2/\log m)$. This completes the proof of Theorem 22.

Improved Space-efficiency

The approach in Section 6.2.4 can be combined with the work of [182] to achieve the improved time and space bound in Theorem 23 for computing the lookup table. Notice that Theorem 20 gives a time and space bound for each pair of substrings for which we need to compute a block function. Specifically, each pair of strings contributes $O(k^2 \lg k)$ time and $O(k^2)$ space. As a complement, [182] showed how to encode strings in such a way that we reduce the number of redundant string comparisons. There, the number of strings compared is reduced to $O((2k)!)$.

Theorem 23 follows from these simple observations.

Exploiting Prefix Similarity in One-against-many Comparison

Since the one-against-many banded alignment algorithm in Section 6.2.5 uses the same extension to banded alignment as [161], it can be combined with other techniques from that paper. In particular, they divide all of the strings in the database S into blocks and store the blocks in a trie-like data structure. This allows them to exploit prefix similarity of the strings of S and further improve the running time in practice. Additionally, that uses *lazy computation*, the technique of computing and storing the lookup table on-the-fly rather than precomputing it to heuristically avoid comparing substrings which don't actually appear in the dataset. In the context of Theorem 21, that could potentially reduce the md^3 factor.

6.2.7 Conclusion and Future Directions

In this section, we provided an approach to storing and querying block functions in the Four Russians speedup for edit distance computation using less time and space than the original method. We demonstrated how this approach can lead to an algorithm for the one-against-many banded alignment problem. Finally, we showed how our approach can easily be combined with prior work to gain additional improvements such as space-efficiency.

The problems of comparing two similar strings and one-against-many comparison of highly similar strings have applications in variety of domains. For example,

searching a query sequence against the database of multiple sequence within a certain similarity threshold is one of the basic tasks in designing database management systems. In the case of document plagiarism detection, the task is to compare two documents which are assumed to be highly similar to each other. In the case of computational biology, sequence similarity detection is a ubiquitous task in most analysis. Although there have been efficient algorithms proposed in literature, they are not very easy or practical to implement on a routine basis. Our algorithm may bridge this gap and be easier to implement while yielding similar theoretical bounds.

There are many questions and potential future directions following this work. One natural question is whether the techniques in this paper can be applied to other problems yielding a Four Russians speedup. In many cases, such as boolean matrix multiplication, the answer is no. However, problems more closely related to edit distance may yield some improvement. Regarding the specific problems in this paper, the $O(nd^3)$ term in the one-against-many result can likely be improved to $O(nd^2)$ to match [188] and doing so using practical techniques would be a nice addition to this work. Similarly, improving the constant factors in the query by using a more specialized algorithm than SMAWK (even an asymptotically worse algorithm) could enhance the practical applications of our approach. On the hardness side, which of these results are tight?

6.3 Maximum Duo-preservation String Mapping

String comparison is a fundamental problem in many fields such as bioinformatics and data compression. The difference between two strings is often measured by some notion of edit distance, the number of edit operations required to transform one string into another. The classic Levenshtein distance definition includes insertion, deletion, and/or substitution operations on single characters. However, the more general edit distance with moves problem studied in [198] allows an additional operation wherein an entire block of text is shifted within a string.

Variations of these shift operations, also known as rearrangements, are commonly studied in genomics [199, 200] with several biologically motivated twists on the above definition. String comparison of DNA or protein sequences can provide an estimate of how closely related different species are. In data compression, we may want to store many similar strings as a single string along with the edits required to recover all strings. These two applications even overlap naturally in the field of bioinformatics where extremely large datasets of biological sequences are common. For example, the challenge of pan-genome storage is to store many highly similar sequences from the same clade such as a bacterial species.

One way to capture just the "moves" operation on two strings which are permutations of each other is the Minimum Common String Partition problem (MCSP). In that problem, we cut (partition) each string into a multi-set of substrings such that the two multi-sets are identical and the number of cuts is minimized. This paper studies the complementary problem to MCSP, the Maximum Duo-Preservation

String Mapping Problem (MPSM) and its weighted variant (MWPSM). Our goal is to find a one-to-one mapping from the letters of one string to the other. The objective is to maximize the pairs of consecutive letters (duos) which map to pairs of consecutive letters in the other string (i.e. pairs that are not cut in MCSP).

While MCSP has been well-studied for some time, a recent flurry of work on MPSM has given us a deeper understanding of that problem. Mehrabi [201] introduced the Maximum Weight Duo-Preservation String Mapping Problem (MWPSM) to better capture applications in comparitive genomics. Beyond identifying the number of block moves, the weighted variant allows us to address questions like, "How far did these blocks move?" This better captures the concept of "synteny" in genetics [202, 203]. Also addressing practical considerations, Dudek, et al [204] included a quadratic time version of their approximation algorithm whereas much of the prior work has focused on improving the approximation in polynomial time.

6.3.1 Problem Description

The Maximum Duo-Preservation String Mapping Problem (MPSM) is defined as follows. We are given two strings $A = a_1 a_2 \dots a_n$ and $B = b_1 b_2 \dots b_n$ of length n such that B is a permutation of A. Let a_i and b_j be the i^{th} and j^{th} characters of their respective strings. A *proper* mapping π from A to B is a one-to-one mapping with $a_i = b_{\pi(i)}$ for all $i = 1, \dots, n$. A *duo* is simply two consecutive characters from the same string. We say that a duo (a_i, a_{i+1}) is *preserved* if a_i is mapped some b_j and a_{i+1} is mapped to b_{j+1}. The objective is to return a proper mapping from the

letters of A to the letters of B which preserves the maximum number of duos. Note that the number of duos preserved in each string is identical and by convention we count the number of duos preserved in a single string rather than the sum over both strings. Let OPT_{MPSM} denote the number of duos preserved from a single string in an optimal solution to the MPSM problem. Figure 6.12 shows an example of an optimal mapping which preserves the maximum possible number of duos.

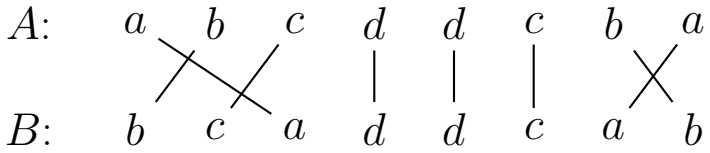


Figure 6.12: Illustration of a mapping π from A to B that preserves 3 duos: bc, dd, and dc. A solution to the complementary MCSP problem on the same strings would be partitions $P_A = a, bc, ddc, b, a$ and $P_B = bc, a, ddc, a, b$ with $|P_A| = |P_B| = 5$.

The complementary Minimum Common String Partition problem (MCSP) seeks to find partitions of the strings A and B where a partition P_A of A is defined as a set of substrings whose concatenation is A. The objective is to find minimum cardinality partitions P_A of A and P_B of B such that P_B is a permutation of P_A. Let OPT_{MCSP} denote the cardinality of a partition in an optimal solution. We can see that $OPT_{MCSP} = |P_A| = |P_B| = n - OPT_{MPSM}$. The variants, k-MPSM and k-MCSP, add the restriction that each letter occurs at most k times in each string. For a given algorithm, let ALG_{MPSM} be number of duos preserved by the algorithm. The approximation ratio for that algorithm is defined as OPT_{MPSM}/ALG_{MPSM}.

In MWPSM, a weight is assigned to every pair of preservable duos and we seek to maximize the weight of the solution. While [201], discusses using weights to capture the positions of preserved duos within their respective strings, the weights

in MWPSM can be arbitrary and are not required to be a function of position.

6.3.2 Related Work

The Maximum Duo-Preservation String Mapping Problem (MPSM) was introduced in [205] along with the related Constrained Maximum Induced Subgraph (CMIS) and Constrained Minimum Induced Subgraph (CNIS) problems. They used a linear programming and randomized rounding approach to approximate the k-CMIS problem which they show is a generalization of k-MPSM. This led to a k^2-approximation for $k \geq 3$ and a 2-approximation for $k = 2$. This was improved by [206] to a 4-approximation independent of k and running in $O(n^{3/2})$ time as well as approximation ratios of 3 for $k = 3$ and 8/5 for $k = 2$. [206] also showed that k-MPSM is APX-hard even for $k = 2$, meaning no polynomial-time approximation scheme (PTAS) exists assuming $P \neq NP$. The approximation was subsequently improved to 3.5 using local search [207], 3.25 using a combinatorial triplet matching approach [9], and finally $2 + \epsilon$ for any $\epsilon > 0$ in $n^{O(1/\epsilon)}$ time, again using local search [204]. The work of [204] also presented a 2.67-approximation running in $O(n^2)$ time.

The recent interest in MPSM led to the study of several variants including Maximum Weight Duo-preservation String Mapping (MWPSM), k-MPSM, and fixed-parameter tractability (FPT). The weighted variant of MPSM was introduced in [201] along with an algorithm achieving a 6-approximation. That work was the first to apply the local ratio technique developed by Bar-Yehuda and Even [208]

to an MPSM problem. Recent work on k-MPSM led to a $(1.4+\epsilon)$-approximation for 2-MPSM [209]. On the FPT side, [210] showed that MPSM is fixed-parameter tractable when parameterized by the number of preserved duos and [211] achieved a faster running time with a randomized algorithm.

The Minimum Common String Partition problem (MCSP) has been extensively studied from many angles including polynomial-time approximation [198, 205, 212–215], fixed-parameter tractability [216–219], and heuristics [220–222]. FPT algorithms have been parameterized by maximum number of times any character occurs, minimum block size, and the size of the optimal minimum partition. Heuristic approaches range from an ant colony optimization algorithm [220] to integer linear programming (ILP) based strategies [221, 222] which in some cases solve the problem optimally for strings up to $2,000$ characters in length.

The problem was shown to be NP-hard (thus implying MPSM is also NP-hard) and APX-hard even for 2-MCSP [213]. The current best approximations are an $O(\log n \log^* n)$-approximation due to [198] for general MCSP bases on the related edit distance with moves problem and an $O(k)$-approximation for k-MCSP due to [214]. Applications to evolutionary distance and genome rearrangement can be found in [199, 200].

Unclaimed results in prior work: An analysis of prior work shows that 4-approximations to both problems studied here can be achieved using slight modifications to existing work. For MWPSM, the algorithm in [206] can be extended by choosing a maximum weight matching and partition rather than maximum cardinality. For the unweighted problem, Goldstein and Lewenstein [223] showed

an $O(n)$ time greedy algorithm for MCSP. Although not discussed in their paper which pre-dated MPSM, we note that the greedy algorithm for MCSP achieves a 4-approximation for MPSM by a fairly straightforward charging argument. Formal proofs of these claims are outside the scope of this work and we leave them to the interested reader. Additionally, we will not refer to these approximations when comparing our work to previous best known results. We simply mention them here for completeness and to give a nod to two nice papers in the area.

6.3.3 Our Contributions

We show a transformation of the Maximum Duo-Preservation String Mapping (MPSM) problem into a related tractable problem. This transformation leads to new algorithms for both weighted and unweighted MPSM. For the weighted case, we present an 8/3-approximation running in $O(n^3)$ time. This improves upon the previous best 6-approximation in polynomial time [201] (a tighter bound on the running time is not given in the paper). It also matches the best quadratic time approximation for the unweighted problem of 2.67 and approaches the best unweighted approximation of $2 + \epsilon$ for any $\epsilon > 0$ in $n^{O(1/\epsilon)}$ time, both due to [204]. We further show in Corollary 3 that we can improve the running time at the cost of a weaker approximation. For the unweighted case, we present the first linear time algorithm with an 8/3-approximation again matching the previous best quadratic time algorithm and coming fairly close to the best known $(2 + \epsilon)$-approximation achieved by a significantly larger running time. In particular, the move from quadratic to linear

time in length of the strings is significant for practical settings wherein the string length may be long enough that quadratic time is prohibitive. Finally, we introduce the first streaming algorithm for MPSM in the streaming model where each string is read one character at a time. We show that a single pass suffices to find a 4-approximation on the size of an optimal solution using only $O(\alpha^2 \lg n)$ space.

In addition, the techniques here are novel to this problem and may inspire future improvements. While [9] also used a form of triplet matching, the structure of the triplet matching is different as is the approach to achieving a feasible solution to MPSM. Our main results are summarized in the theorems below.

Theorem 24. *There exists an algorithm which finds an* $8/3$*-approximation to MWPSM on strings of length* n *in* $O(n^3)$ *time.*

Corollary 3. *Using the approximate weighted matching algorithm of [224], we can find an* $8/(3(1-\epsilon))$*-approximation to MWPSM on strings of length* n *for any* $\epsilon > 0$ *in* $O(n^2\epsilon^{-1} \lg \epsilon^{-1})$ *time.*

Theorem 25. *There exists an algorithm which finds an* $8/3$*-approximation to MPSM on strings of length* n *over alphabets of size* α *in* $O(n + \alpha^7)$ *time.*

Corollary 4. *There exists an algorithm which finds an* $8/3$*-approximation to MPSM on strings of length* n *over constant-sized alphabets in* $O(n)$ *time.*

Theorem 26. *There exists a single-pass streaming algorithm which finds a* 4*-approximation to the size of an MPSM on strings of length* n *over alphabets of size* α *using only* $O(\alpha^2 \lg n)$ *space.*

6.3.4 Preliminaries

Let $A = a_1a_2 \dots a_n$ and $B = b_1b_2 \dots b_n$ be two strings of length n with a_i and b_j being the i^{th} and j^{th} characters of their respective strings. A *duo* $D_i^A = (a_i, a_{i+1})$ contains a pair of consecutive characters a_i and a_{i+1}. We use $D^A = (D_1^A, \dots, D_{n-1}^A)$ and $D^B = (D_1^B, \dots, D_{n-1}^B)$ to denote the sets of *duos* for A and B, respectively. We similarly define a *triplet* $T_i^A = (a_i, a_{i+1}, a_{i+2})$ as a set of three consecutive characters a_i, a_{i+1}, and a_{i+2} in the string and sets of *triplets* $T^A = (T_1^A, \dots, T_{n-2}^A)$ and $T^B = (T_1^B, \dots, T_{n-2}^B)$ for strings A and B, respectively. Observe that the duos D_i^A and D_{i+1}^A correspond to the first two and last two characters, respectively, of the triplet T_i^A. We refer to duos D_i^A and D_{i+1}^A as *subsets* of the triplet T_i^A.

A proper mapping π from A to B is a one-to-one mapping from the letters of A to the letters of B with $a_i = b_{\pi(i)}$ for all $i = 1, \dots, n$. Recall that a duo (a_i, a_{i+1}) is preserved if and only if a_i is mapped to some b_j and a_{i+1} is mapped to b_{j+1}. We call a pair of duos (D_i^A, D_j^B) *preservable* if and only if $a_i = b_j$ and $a_{i+1} = b_{j+1}$. For MWPSM, let $w(D_i^A, D_j^B)$ be the weight gained by mapping D_i^A to D_j^B.

For consistency, we define the concept of conflicting pairs of duos using the terminology of [206]. Two preservable pairs of duos (D_i^A, D_j^B) and (D_h^A, D_ℓ^B) are said to be *conflicting* if no proper mapping can preserve both of them. These conflicts can be of two types type 1 and type 2. In *type 1 conflicts*, either $i = h \wedge j \neq \ell$ or $i \neq h \wedge j = \ell$. In *type 2 conflicts*, either $i = h+1 \wedge j \neq \ell+1$ or $i \neq h+1 \wedge j = \ell+1$.

The algorithms here only show how to map the characters of the preserved duos. In all cases, note that any unmapped characters can be mapped arbitrarily

to identical characters in the other string in linear time.

6.3.5 Main Techniques and Algorithm for MWPSM

For both algorithms, we first solve a weighted bipartite matching problem we call Alternating Triplet Matching (ATM). In this section, we define ATM, show that a solution to ATM has weight at least 3/4 of an optimal solution to MWPSM, and finally show that we can convert a solution to ATM to a feasible duo mapping while preserving 1/2 of its weight. Combining these facts leads to an 8/3-approximation to MWPSM.

The Alternating Triplet Matching (ATM) Problem

Here, we define this problem in terms of MWPSM. Modifications for the unweighted variant (to admit a faster solution) will be defined in Section 6.3.6. Let $T^{A'} = \{T_i^A \mid i \text{ is odd}\}$, $T^{B'} = \{T_i^B \mid i \text{ is odd}\}$ and $T^{B''} = \{T_i^B \mid i \text{ is even}\}$. Throughout, we refer to triplets starting at odd indices in their respective strings as *odd triplets* and similarly use the term *even triplets*. Note, we do not use the even triplets from A.

Using these subsets, we formulate bipartite matching problems on two separate graphs $G' = \{T^{A'}, T^{B'}, E'\}$ and $G'' = \{T^{A'}, T^{B''}, E''\}$. The edges of G' depend on the letters in the triplets. Consider triplets $T_i^{A'} = (D_i^A, D_{i+1}^A)$ and $T_j^{B'} = (D_j^B, D_{j+1}^B)$. For each pair of duos D_h^A and D_ℓ^B with $h \in \{i, i+1\}$, $\ell \in \{j, j+1\}$, and $D_h^A = D_\ell^B$, we add an edge $e = (T_i^{A'}, T_j^{B'})$ with weight $w(e) = w(D_h^A, D_\ell^B)$. Additionally, if

$T_i^{A'} = T_j^{B'}$, we add an edge $e = (T_i^{A'}, T_j^{B'})$ between them with weight $w(e) = w(D_i^A, D_j^B) + w(D_{i+1}^A, D_{j+1}^B)$. In other words, the edge gets the combined weight of the duo pairs preserved by mapping the substring $T_i^{A'}$ to the substring $T_j^{B'}$. The graph G'' is defined similarly. There could be up to five edges total if the triplets contain one letter repeated (e.g. "AAA"). In the case of multiple edges between a pair of triplets, we only need to consider the heaviest edge among them since each triplet can be matched at most once. However, we keep all edges for the sake of simplifying some of the proofs. Figure 6.13 illustrates the procedure of generating an ATM instance.

MWPSM Algorithm and Analysis

Let $OPT_{G'}$ and $OPT_{G''}$ be the weights of maximum weight matchings in G' and G'', respectively. Note that we can find these matchings in the time it takes to compute maximum weight bipartite matching. Since our graphs have $O(n)$ vertices and could have $O(n^2)$ edges, this takes $O(n^2 \lg n + n \cdot n^2) = O(n^3)$ time [225]. Lemma 22 states that either $OPT_{G'}$ or $OPT_{G''}$ will be a (3/4)-approximation to the weight of an optimal solution to MWPSM, OPT_{MWPSM}. Let $OPT_{ATM} = \max(OPT_{G'}, OPT_{G''})$.

Lemma 22. $OPT_{ATM} \geq (3/4)OPT_{MWPSM}$.

Proof. We divide the edges of OPT_{MWPSM} into two partitions. The first partition, P^{same}, includes mappings, in which both letters occur at odd indices or both letters occur at even indices. The second partition, P^{diff}, includes the remaining mappings wherein one letter is at an odd index and the other is at an even index (this could

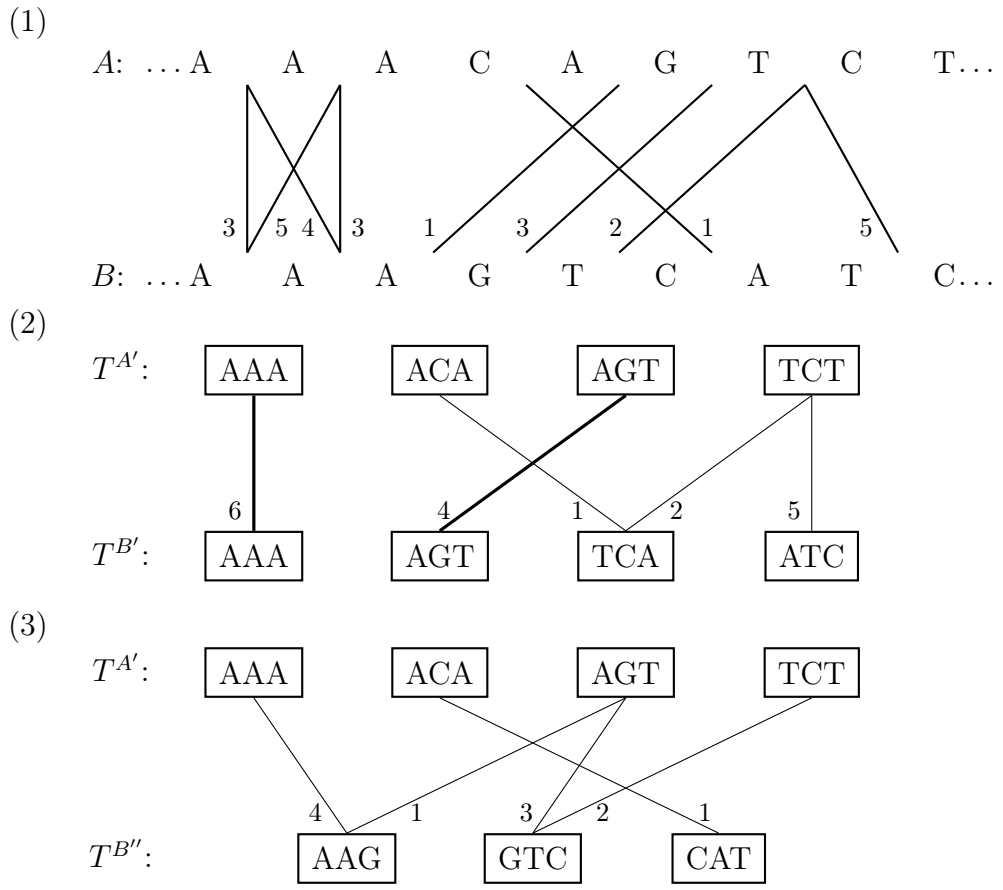


Figure 6.13: Illustration of how to generate an ATM instance from an MWPSM instance. (1) Substrings of the original two strings, A and B, starting at some odd index and featuring weighted edges representing the weight of preserving a pair of duos. (2) The graph G' with thicker edges representing an exact match between two triplets. In the case of multiple edges between a pair of triplets (e.g. the five edges between the "AAA" triplets), we only show the heaviest weight edge. (3) The graph G''. Note that that the weight of a mapping which maps the two "AGTC" strings to each other is 6, which can be achieved by a matching in G', but not in G''.

be odd from A, even from B or even from A, odd from B).

Note that the mapping of each preserved pair of duos (D_i^A, D_j^B) will be contained in one of these two partitions. Without loss of generality, let the weight of P^{same} be at least the weight of P^{diff}. We show how to transform OPT_{MWPSM} into a feasible bipartite matching in G' while retaining the full weight of P^{same} and at least half of the weight of P^{diff}. Thus, we retain at least 3/4 of the weight of

OPT_{MWPSM}.

For each triplet in the vertex set of G' that contains one or two preserved duos from P^{same}, we can add an edge to our matching with weight equal to the weight of the preserved duos. This works because consecutive pairs of preserved duos (D_i^A, D_j^B) and (D_{i+1}^A, D_{j+1}^B) with i and j both being odd will correspond to a "double" edge in the ATM instance with weight equal to $w(D_i^A, D_j^B) + w(D_{i+1}^A, D_{j+1}^B)$. On the other hand, if i and j are both even, then the duos of (D_i^A, D_j^B) and (D_{i+1}^A, D_{j+1}^B) are contained in four different triplets and will be added separately. Thus, we can maintain all of the weight of P^{same} in a matching in G'.

A slightly trickier case arises with P^{diff}. Any consecutive pairs of preserved duos (D_i^A, D_j^B) and (D_{i+1}^A, D_{j+1}^B) in P^{diff} will have i and j of different parity. This results in the duos being contained in three triplets, two from one partition and one from the other. That means the edges in the ATM instance capturing the weights of the two pairs will be conflicting. Thus we can only preserve the weight of one of the two pairs in our ATM solution. To guarantee that we add at least half of the weight of P^{diff} to our solution, we further partition it into pairs (D_i^A, D_j^B) with i being odd and those with i being even. Then we simply choose the heavier of those two partitions to add to our ATM solution.

For the case where P^{diff} is heavier than P^{same}, we can do a similar construction for G''. Thus, our ATM solution in either G' or G'' could have at least 3/4 the weight of an optimal solution to MWPSM. □

We can now show how to transform an optimal solution to ATM (the heavier

of the two matchings) into a feasible string mapping which preserves at least half of the weight of the ATM solution. Let $G = (D^A, D^B, E)$ be a bipartite graph on the duos of A and B with edge weights equal to the weight of preserving each pair of duos. We first show how to convert an ATM solution into a matching M in G. Then, we show how to resolve conflicts of type 2 (conflicts of type 1 will not arise since M is a matching).

The transformation is simply a reversal of how we constructed the ATM graphs. For each edge between triplets in our ATM solution (the heavier of the two matchings in G' and G''), we add an edge or edges to M corresponding to the duos that "created" that triplet edge.

To resolve conflicts, we consider the conflict graph C wherein we have a node for each edge in M and an arc between nodes if their corresponding edges are in conflict. We can prove that C has maximum degree 2, meaning it will be a collection of paths and cycles. Further, we note that each cycle will have even length due to Lemma 23 and the fact that the underlying graph is bipartite. Thus, for each path or cycle, we choose the heavier of the two maximal independent sets in that path or cycle to add to our final MPSM solution. Lemma 23 establishes that C has maximum degree 2.

Lemma 23. *Each edge in M conflicts with at most one other edge at each endpoint.*

Proof. First, we note that each duo is contained in at most one triplet edge from the ATM solution and therefore can only be matched once in M. In other words, M is a classical matching in the bipartite graph of duos. This follows from the fact

that consecutive triplets in a string starting at only odd (or only even) indices will overlap at exactly one letter.

This ensures that no conflicts of type 1 can arise since that would require a duo to be matched twice. We can also show that at most one conflict of type 2 arises at each endpoint. Without loss of generality, consider the endpoint D_i^A. Consider the duos D_{i-1}^A and D_{i+1}^A where such a conflict might arise. Notice that one of these duos must have come from the same triplet as D_i^A, while the other comes from a different triplet. The duo from the same triplet will either be unmatched or matched as a non-conflicting parallel edge. Thus no conflict arises from that duo. The duo from a different triplet could contribute at most one conflicting edge by the above claim that each duo is matched at most once. Applying this argument to both endpoints of a given edge completes the proof. □

Lemma 24. *M can be converted into M', a feasible solution to MWPSM, such that the weight of M' is at least $(1/2)OPT_{ATM} \geq (3/8)OPT_{MWPSM}$.*

Proof. The conflict graph on the edges of M must be a collection of paths and even length cycles since it has maximum degree 2 and G is bipartite. We can simply decompose each path or cycle into two independent sets and choose the heavier of the two. This operation discards at most half of the weight of M while removing all conflicts and leaving us with a feasible solution to MWPSM. □

The proofs of Theorem 24 and Corollary 3 follow from the preceding lemmas.

6.3.6 Linear Time Algorithm for Unweighted MPSM

The basic approach follows roughly the same steps as the weighted algorithm from Section 6.3.5: construct an ATM instance, solve the matching problem, transform the solution into a duo matching on the strings, and resolve conflicts. We show that with a small modification, each step can be done in linear time for the unweighted problem. The key insight that allows for this speedup is that identical triplets can be collapsed into single vertices and we can solve a b-matching problem we call b-ATM. In the *b-matching* variant of classical matching, each vertex in the graph has a capacity and can be matched that many times. We will abuse notation a bit and refer to each vertex as having capacity b, although we actually allow the capacity of each node to be different. The following subsections illustrate how to perform the aforementioned steps and bound the running time of each step.

Constructing the b-ATM Instance in $O(n + \alpha^4)$ Time

We construct a triplet matching problem as in Section 6.3.5 with one crucial adjustment: identical triplets are collapsed into single vertices with capacity equal to the number of occurrences of that triplet in its given set ($T^{A'}$, $T^{B'}$, or $T^{B''}$). The number of times each vertex is allowed to be matched is equal to its capacity. Similarly, each edge can be matched multiple times up to the smaller capacity among its two endpoints. Algorithm 7 shows how to construct a b-ATM instance from the two input strings in linear time.

As in Section 6.3.5, let $OPT_{G'}$ and $OPT_{G''}$ be the weights of maximum weight

Algorithm 7: CONSTRUCT B-ATM

1 Traverse each string to build a set of triplets with counts for A', B', and B''.

2 For G' and G'', create a vertex for each triplet with capacity equal to its count. Add edges between the triplets as in Section 6.3.5 with the following modification. If two triplets match exactly, give the edge weight 2 and if they only share a duo in common, give the edge weight 1.

b-matchings in G' and G'', respectively. Lemma 25 states that either $OPT_{G'}$ or $OPT_{G''}$ will be a (3/4)-approximation to the size of an optimal solution to MPSM, OPT_{MPSM}. Let $OPT_{b\text{-}ATM} = \max(OPT_{G'}, OPT_{G''})$ as constructed by Algorithm 7.

Lemma 25. $OPT_{b\text{-}ATM} \geq (3/4)OPT_{MPSM}$.

Proof. This proof follows from Lemma 22. Suppose we constructed an ATM instance as in Section 6.3.5, but for the unweighted problem. By Lemma 22, we would have $OPT_{ATM} \geq (3/4)OPT_{MPSM}$. Now note that we can collapse all identical triplet vertices in each partition of OPT_{ATM} to get a feasible solution to the b-ATM problem without reducing the weight. □

Lemma 26. *Algorithm 7 constructs a graph with $O(\alpha^3)$ vertices and $O(\alpha^4)$ edges in $O(n + \alpha^4)$ time.*

Proof. Step 1 of the algorithm clearly runs in less than $O(n + \alpha^4)$ time. It simply traverses each string once, storing the triplets in some appropriate data structure with constant insert and query time.

To bound the running time of step 2, we first bound the number of edges created. Note that the bipartite graph of b-ATM has $O(\alpha^3)$ vertices in each partition since that is the maximum number of 3-mers in an alphabet of size α. To bound

the edge set, notice that for any 3-mer, there exist at most 4α other 3-mers with a substring of length 2 in common. Thus, the max degree of each node is $O(\alpha)$ and the size of the edge set E is at most $O(\alpha^4)$. When adding edges, we can check for the existence of each edge in constant time, again assuming the triplet are stored in some appropriate data structure. □

Solving b-ATM Quickly

Algorithm 8 shows how to solve b-ATM within our time constraints. Lemma 27 proves the correctness of this algorithm while Lemma 28 bounds its running time.

Algorithm 8: SOLVE B-ATM

1 Add each edge with weight 2, corresponding to two identical triplets, to the matching.
2 Find a maximum b-matching in the remaining "unweighted" graph using maximum flow techniques.

Lemma 27. *Algorithm 8 finds a maximum weight b-matching in the b-ATM instance.*

Proof. Here, we need to justify Step 1 of Algorithm 8 by showing that there always exists some maximum b-matching which contains all of the edges corresponding to identical pairs of triplets. First note that it is feasible to include all such edges since they can never conflict with each other. For each triplet in one partition, there is at most one identical triplet in the other partition.

We apply the following claim iteratively to complete the proof. Given a maximum weight b-matching M which does not include all identical pair edges, we can

always add one such edge without decreasing the weight of the solution. Consider an arbitrary identical pair edge e that is not in M. To add e to M we need to remove at most two edges from M, one for each endpoint of e. Since e has a weight of 2 while the removed edges have weights of 1 each, swapping those edges for e will not reduce the weight of the solution. □

Lemma 28. *Algorithm 8 runs in $O(n)$ time plus the time to compute an unweighted maximum b-matching on a graph with $O(\alpha^3)$ vertices and $O(\alpha^4)$ edges and total capacity $O(n)$. Using current maximum flow algorithms, Algorithm 8 can run in $O(n + \alpha^7)$ time.*

Proof. If the graph were unweighted, we could find a maximum b-matching in $O(|V||E|) = O(\alpha^7)$ time using the maximum flow approach in [226]. Fortunately, by Lemma 27, we can first add all edges with weight 2 to our solution. Thus, we are left with an "unweighted" residual problem that can be solved using a maximum flow algorithm. □

Transforming b-ATM to a Duo Matching and Resolving Conflicts

Now that we have solved our b-ATM problem we need transform it back to a duo matching. The obvious challenge here is that each b-ATM vertex represents roughly b copies of a given 3-mer that must each be assigned to a triplet in the original string in linear time while preserving the weight of the b-ATM solution. There are $b!$ such assignments and b could be on the order of n. However, the important observation here is that we can do this *arbitrarily* and still preserve the

size of the b-ATM solution.

Algorithm 9: TRANSFORM B-ATM TO MPSM

1 Assign each copy of a 3-mer and its edge from the b-ATM solution to a triplet from the original strings to get an ATM solution.
2 Transform the ATM solution into a duo matching as detailed in Section 6.3.5
3 Resolve conflicts by traversing the paths/cycles of the conflict graph and discarding every other edge.

Lemma 29. *Algorithm 9 constructs a feasible solution to MPSM with size equal to half the weight of* $OPT_{b\text{-}ATM}$.

Proof. The proof follows from Lemma 24. Notice that we assign exactly one copy of a 3-mer to each triplet and the result is a feasible solution to the ATM problem. □

Lemma 30. *Algorithm 9 runs in* $O(n)$ *time.*

Proof. Assigning each copy of a 3-mer and its edge to a triplet can be done in constant time if we maintain lists of the indices at which each 3-mer occurs in each string, resulting in $O(n)$ time overall. Similarly, generating the duo-matching can easily be done in $O(n)$ time. Resolving conflicts in the unweighted problem involves traversing $O(n)$ edges and removing every other one which can be done in $O(n)$ time as well. □

The proofs of Theorem 25 and Corollary 4 follow from the preceding lemmas.

6.3.7 A Streaming Algorithm for MPSM

We observe that the algorithm of [206] can be adapted into a single-pass streaming algorithm in the streaming model where each string is read one char-

acter at a time. We present an algorithm using $O(\alpha^2 \lg n)$ space and giving a 4-approximation of the size of an MPSM solution without providing an explicit mapping. In [206], they upper bound MPSM by a maximum matching in the duo graph. Then they show that a feasible MPSM solution can be found while preserving at least 1/4 of the edges in the matching.

The algorithm is simple. Maintain a counter for each 2-mer in the alphabet and a counter for the size of the matching. While processing the first string, count the number of occurrences of each 2-mer. For the second string, each time you encounter a duo with a nonzero count, decrease its count by 1 and increase the size of the matching by 1. At the end, divide the size of the matching by 4 to get a 4-approximation to the size of the optimal MPSM. The following Lemmas establish the space-efficiency and correctness of the the algorithm.

Lemma 31. *The streaming algorithm uses only $O(\alpha^2 \lg n)$ space where α is the alphabet size and n is the length of the strings.*

Proof. The number of 2-mers from an alphabet of size α is α^2. We require only $O(\lg n)$ bits of space for each 2-mer counter since no 2-mer could appear more than $O(n)$ times where n is the length of the strings. Similarly, we keep just one counter for the size of the matching which requires only $O(\lg n)$ bits of space since the size of the matching is at most n. In addition to the counters, we must store the previously seen letter since our streaming model involves reading one character at a time, but we are counting duos. However, this only requires $O(\lg \alpha)$ space. □

Lemma 32. *The streaming algorithm achieves a 4-approximation to MPSM.*

Proof. We first show that the size of a maximum matching in a bipartite duo graph G as defined in [206] is equal to the sum of the minimum number of occurrences of each duo among the two strings. Notice that G can be decomposed into a set of connected components for each 2-mer since each vertex only has edges to other vertices corresponding to the same 2-mer. Further, each of these connected components is a complete bipartite graph with maximum matching size equal to the minimum size of the two partitions.

Thus, computing the above sum gives us the size of the maximum matching. We note that the number of times the matching size counter increase due to vertices of a given 2-mer is exactly equal to the minimum number of times that 2-mer appears in either of the two strings.

Finally, as shown in [206], a maximum matching in the duo graph is an upper bound on the optimal solution to MPSM and can always be converted into a feasible MPSM solution while preserving at least 1/4 of its size. □

The proof of Theorem 26 follows from Lemmas 31 and 32.

6.3.8 Conclusion and Future Directions

We showed a transformation of the Maximum Duo-Preservation String Mapping (MPSM) problem into a related tractable problem. This led to new algorithms for both MWPSM and MPSM. For the weighted case, we presented a tighter approximation closing in on the best unweighted result using a reasonably fast algorithm. We also showed that the running time could be improved at the expense of a slightly

weaker approximation. For the unweighted case, we presented the first linear time algorithm with an approximation matching the previous best quadratic time algorithm and fairly close to the best known approximation achieved by a significantly larger running time. Finally, we presented the first streaming algorithm for MPSM showing that a constant approximation is achievable in the single-pass streaming model.

We believe the most pressing future direction is to explore the applications and utility of this problem further. The complementary relationship with Minimum Common String Partition (MCSP) has driven much of the current interest in MPSM. However, given their relationship, new approximations for MPSM do not directly lead to any improvements for MSCP. It is reasonable to ask if the study of MPSM can teach us anything about MCSP or at least inspire new heuristics. We note that some current linear-time algorithms for MCSP are greedy algorithms [223] with a proven lower bound of $\Omega(n^{0.46})$ [227] (Although this bound arises from carefully constructed strings over a $(\log n)$-sized alphabet). This is in contrast to the best known approximation for MCSP, $O(\log n \log^* n)$ [198]. Perhaps the linear time MPSM algorithm presented here could be combined with greedy approaches leading to better, more robust heuristics. Further, since MPSM currently appears to be "easier" than MCSP, it would be fruitful to explore more applications for MPSM itself in bioinformatics, data compression, and beyond.

On the theoretical side, the biggest questions revolve around the factor of 2 approximation. Is this tight for MPSM conditioned on some hardness conjecture or can we do better? It surely seems like a natural bound. Regardless, can we achieve

a 2-approximation in linear time? Likewise, for MWPSM, a 2-approximation could be seen as the next major goal. All of this seems within reach, using existing ideas or different tools such as LP rounding techniques. Another direction would be to add edit operations. It seems that MWPSM could be adapted to handle the cost of substitutions. However, this is nontrivial since existing algorithms assume that letters which do not belong to preserved duos can be mapped at no penalty.

Finally, we propose variants of MWPSM that may admit a faster approximation than we have seen here. Suppose the weights are not arbitrary, but follow some "rules". [201] suggested the weight of a duo-preservation could be a function of the "closeness" of the mapping in terms of the positions of the characters in their respective strings. However, [201] and our work consider only arbitrary weights. One could imagine a weight function like $w(D_i^A, D_j^B) = n - |i - j|$ that does not require us to examine every edge in the duo graph. Of course, the function need not be so naive as any metric or geometric weight functions admit faster matching algorithms [228].

Bibliography

[1] J Gregory Caporaso, Christian L Lauber, William A Walters, Donna Berg-Lyons, James Huntley, Noah Fierer, Sarah M Owens, Jason Betley, Louise Fraser, Markus Bauer, et al. Ultra-high-throughput microbial community analysis on the Illumina HiSeq and MiSeq platforms. *The ISME journal*, 2012.

[2] Daniel J. Nasko, Sergey Koren, Adam M. Phillippy, and Todd J. Treangen. Refseq database growth influences the accuracy of k-mer-based lowest common ancestor species identification. *Genome Biology*, 2018.

[3] Susan R. Leonard, Mark K. Mammel, David W. Lacher, and Christopher A. Elkins. Application of metagenomic sequencing to food safety: Detection of shiga toxin-producing escherichia coli on fresh bagged spinach. *Applied and environmental microbiology*, 2015.

[4] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 2006.

[5] Allan Borodin, Christodoulos Karavasilis, and Denis Pankratov. An experimental study of algorithms for online bipartite matching, 2018.

[6] Jon Kleinberg and Éva Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *J. ACM*, 2002.

[7] Rucho v. Common Cause, No. 18-422, 588 U.S. ___ (2019).

[8] Brian Brubach. Fast Matching-based Approximations for Maximum Duo-Preservation String Mapping and its Weighted Variant. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2018.

[9] Brian Brubach. Further improvement in approximating the maximum duo-preservation string mapping problem. In *International Workshop on Algorithms in Bioinformatics (WABI)*, 2016.

[10] Negin Golrezaei, Hamid Nazerzadeh, and Paat Rusmevichientong. Real-time optimization of personalized assortments. *Management Science*, 2014.

[11] Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 2012.

[12] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Annual Symposium on Theory of Computing (STOC)*, 1990.

[13] Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011.

[14] Aranyak Mehta and Debmalya Panigrahi. Online matching with stochastic rewards. In *Annual Symposium on Foundations of Computer Science (FOCS)*, 2012.

[15] Brian Brubach, Nathaniel Grammel, and Aravind Srinivasan. Vertex-weighted online stochastic matching with patience constraints. *CoRR*, 2019.

[16] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *Annual Symposium on Theory of Computing (STOC)*, 2011.

[17] Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Online vertex-weighted bipartite matching: Beating 1-1/e with random arrivals. *CoRR*, 2018.

[18] Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *European Symposium on Algorithms (ESA)*. 2013.

[19] Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. New algorithms, better bounds, and a novel model for online stochastic matching. *CoRR*, 2016.

[20] Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. New algorithms, better bounds, and a novel model for online stochastic matching. *European Symposium on Algorithms (ESA)*, 2016.

[21] Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. Attenuate locally, win globally: An attenuation-based framework for online stochastic matching with timeouts. In *International Conference On Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2017.

[22] Aranyak Mehta, Bo Waggoner, and Morteza Zadimoghaddam. Online stochastic matching with unequal probabilities. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015.

[23] Nikhil R Devanur and Thomas P Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *ACM Conference on Economics and Computation (EC)*, 2009.

[24] Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Automata, Languages and Programming*. 2009.

[25] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2008.

[26] Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and S Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *Annual Symposium on Foundations of Computer Science (FOCS)*, 2009.

[27] Vahideh H Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. *Math. Oper. Res.*, 2012.

[28] Patrick Jaillet and Xin Lu. Online stochastic matching: New algorithms with better bounds. *Math. Oper. Res.*, 2013.

[29] Bernhard Haeupler, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In *Internet and Network Economics*. 2011.

[30] Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra. When LP is the cure for your matching woes: Improved bounds for stochastic matchings. In *European Symposium on Algorithms (ESA)*, 2010.

[31] Marek Adamczyk, Fabrizio Grandoni, and Joydeep Mukherjee. Improved approximation algorithms for stochastic matching. In *European Symposium on Algorithms (ESA)*, 2015.

[32] Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *International Conference on Automata, Languages, and Programming (ICALP)*, 2009.

[33] Alok Baveja, Amit Chavan, Andrei Nikiforov, Aravind Srinivasan, and Pan Xu. Improved bounds in stochastic matching and optimization. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. (APPROX/RANDOM)*, 2015.

[34] Anupam Gupta and Viswanath Nagarajan. A stochastic probing problem with applications. In *Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2013.

[35] Nikhil R Devanur, Kamal Jain, Balasubramanian Sivan, and Christopher A Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. In *ACM Conference on Economics and Computation (EC)*, 2011.

[36] Nikhil R Devanur, Balasubramanian Sivan, and Yossi Azar. Asymptotically optimal algorithm for stochastic adwords. In *ACM Conference on Economics and Computation (EC)*, 2012.

[37] Saeed Alaei, MohammadTaghi Hajiaghayi, and Vahid Liaghat. Online prophet-inequality matching with applications to ad allocation. In *ACM Conference on Economics and Computation (EC)*, 2012.

[38] Saeed Alaei, MohammadTaghi Hajiaghayi, and Vahid Liaghat. The online stochastic generalized assignment problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. (APPROX/RANDOM)*. 2013.

[39] Jon Feldman, Nitish Korula, Vahab Mirrokni, S Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *Internet and network economics*. 2009.

[40] Bala Kalyanasundaram and Kirk R Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 2000.

[41] Reshef Meir, Yiling Chen, and Michal Feldman. Efficient parking allocation as online bipartite matching with posted prices. In *International Conference On Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2013.

[42] Sepehr Assadi, Sanjeev Khanna, and Yang Li. The stochastic matching problem with (very) few queries. In *ACM Conference on Economics and Computation (EC)*, 2016.

[43] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *Annual Symposium on Foundations of Computer Science (FOCS)*, 2015.

[44] Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. Attenuate locally, win globally: An attenuation-based framework for online stochastic matching with timeouts. In *International Conference On Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2017.

[45] Vineet Goyal and Rajan Udwani. Online Matching with Stochastic Rewards: Optimal Competitive Ratio via Path Based Formulation. *CoRR*, 2019.

[46] Buddhima Gamlath, Sagar Kale, and Ola Svensson. Beating greedy for stochastic bipartite matching. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019.

[47] Bahman Bahmani and Michael Kapralov. Improved bounds for online stochastic matching. In *European Symposium on Algorithms (ESA)*. 2010.

[48] Bela Bollobas and Graham Brightwell. The width of random graph orders. *The Mathematical Scientist*, 1995.

[49] Andrew Mastin and Patrick Jaillet. Greedy online bipartite matching on random graphs. *CoRR*, 2013.

[50] Allan Borodin, Christodoulos Karavasilis, and Denis Pankratov. Greedy Bipartite Matching in Random Type Poisson Arrival Model. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. (APPROX/RANDOM)*, 2018.

[51] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Innovations in Theoretical Computer Science Conference (ITCS)*, 2012.

[52] Sugato Basu, Ian Davidson, and Kiri Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. 2008.

[53] Xiaojin Zhu. Semi-supervised learning literature survey, 2006.

[54] Jian Zhang and Rong Yan. On the value of pairwise constraints in classification and consistency. In *International Conference on Machine Learning (ICML)*, 2007.

[55] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *International Conference on Machine Learning (ICML)*, 2003.

[56] Mikhail Bilenko, Sugato Basu, and Raymond J Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *International Conference on Machine Learning (ICML)*, 2004.

[57] Yale Chang, Junxiang Chen, Michael H Cho, Peter J Castaldi, Edwin K Silverman, and Jennifer G Dy. Multiple clustering views from multiple uncertain experts. In *International Conference on Machine Learning (ICML)*, 2017.

[58] Yucen Luo, Tian Tian, Jiaxin Shi, Jun Zhu, and Bo Zhang. Semi-crowdsourced clustering with deep generative models. In *Neural Information Processing Systems (NeurIPS)*, 2018.

[59] R. C. Edgar. Uparse: highly accurate otu sequences from microbial amplicon reads. *Nat Methods*, 2013.

[60] S.L. Westcott and P.D. Schloss. OptiClust, an improved method for assigning amplicon-based sequence data to operational taxonomic units. 2017.

[61] E. Stackebrandt and B. M. Goebel. Taxonomic note: A place for dna-dna reassociation and 16s rrna sequence analysis in the present species definition in bacteriology. *International Journal of Systematic and Evolutionary Microbiology*, 1994.

[62] Mohammadreza Ghodsi, Bo Liu, and Mihai Pop. Dnaclust: accurate and efficient clustering of phylogenetic marker genes. *BMC bioinformatics*, 2011.

[63] Robert C Edgar. Updating the 97% identity threshold for 16s ribosomal rna otus. *Bioinformatics*, 2018.

[64] Cheng Chen, Lan Zheng, Venkatesh Srinivasan, Alex Thomo, Kui Wu, and Anthony Sukow. Conflict-aware weighted bipartite b-matching and its application to e-commerce. *IEEE Trans. on Knowl. and Data Eng.*, 2016.

[65] Matthew Paul Johnson, Fei Fang, and Milind Tambe. Patrol strategies to maximize pristine forest area. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[66] Yue Yin, Bo An, and Manish Jain. Game-theoretic resource allocation for protecting large public events. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[67] Kareem Amin, Satinder Singh, and Michael P Wellman. Gradient methods for Stackelberg security games. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2016.

[68] Nitin Kamra, Umang Gupta, Fei Fang, Yan Liu, and Milind Tambe. Policy learning for continuous space security games using neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[69] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *International Conference on Machine Learning (ICML)*, 2001.

[70] Stefan Schroedl, Kiri Wagstaff, Seth Rogers, Pat Langley, and Christopher Wilson. Mining GPS traces for map refinement. *Data Mining and Knowledge Discovery*, 2004.

[71] Ian Davidson, S. S. Ravi, and Leonid Shamis. *A SAT-based Framework for Efficient Constrained Clustering.* 2010.

[72] Ian Davidson and S. S. Ravi. The complexity of non-hierarchical clustering with instance and cluster level constraints. *Data Mining and Knowledge Discovery*, 2007.

[73] Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 1985.

[74] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 1985.

[75] Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 1986.

[76] Moses Charikar, Sudipto Guha, ÃĽva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 2002.

[77] Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms (TALG)*, 2017.

[78] Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1979.

[79] Danny Z. Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and knapsack center problems. *Algorithmica*, 2016.

[80] David G. Harris, Thomas Pensyl, Aravind Srinivasan, and Khoa Trinh. A Lottery Model for Center-Type Problems with Outliers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. (APPROX/RANDOM)*, 2017.

[81] Robert Krauthgamer and Tim Roughgarden. Metric clustering via consistent labeling. *Theory of Computing*, 2011.

[82] Brian Brubach, Darshan Chakrabarti, John P. Dickerson, Samir Khuller, Aravind Srinivasan, and Leonidas Tsepenekas. A pairwise fair and community-preserving approach to k-center clustering. In *International Conference on Machine Learning (ICML)*, 2020.

[83] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1992.

[84] Oren Zamir, Oren Etzioni, Omid Madani, and Richard M. Karp. Fast and intuitive clustering of web documents. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 1997.

[85] Amit Datta, Michael Carl Tschantz, and Anupam Datta. Automated experiments on ad privacy settings. *Proceedings on Privacy Enhancing Technologies*, 2015.

[86] A Srinivasan, CJ Galbán, TD Johnson, TL Chenevert, BD Ross, and SK Mukherji. Utility of the k-means clustering algorithm in differentiating apparent diffusion coefficient values of benign and malignant neck pathologies. *American Journal of Neuroradiology*, 2010.

[87] Gustavo Malkomes, Matt J. Kusner, Wenlin Chen, Kilian Q. Weinberger, and Benjamin Moseley. Fast distributed k-center clustering with outliers on massive data. In *Neural Information Processing Systems (NeurIPS)*, 2015.

[88] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Neural Information Processing Systems (NeurIPS)*. 2017.

[89] Ioana O. Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R. Schmidt, and Melanie Schmidt. On the Cost of Essentially Fair Clusterings. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. (APPROX/RANDOM)*, 2019.

[90] Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In *International Conference on Machine Learning (ICML)*, 2019.

[91] United States Census Bureau, Population Division. *"American FactFinder – Results"*, 2016, accessed May, 2018.

[92] Thornburg v. Gingles, No. 83-1968, 478 U.S. 30 (1986).

[93] Yan Y. Liu, Wendy K. Tam Cho, and Shaowen Wang. PEAR: a massively parallel evolutionary computation approach for political redistricting optimization and analysis. *Swarm and Evolutionary Computation*, 2016.

[94] Micah Altman and M McDonald. The promise and perils of computers in redistricting. *Duke Journal of Constitutional Law and Public Policy*, 2010.

[95] Roland G. Fryer and Richard Holden. Measuring the compactness of political districting plans. *The Journal of Law and Economics*, 2011.

[96] Micah Altman. Modeling the effect of mandatory district compactness on partisan gerrymanders. *Political Geography*, 1998.

[97] Vincent Cohen-Addad, Philip N. Klein, and Neal E. Young. Balanced centroidal power diagrams for redistricting. In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2018.

[98] Itai Ashlagi and Peng Shi. Improving community cohesion in school choice via correlated-lottery implementation. *Operations Research*, 2014.

[99] Thomas M. Menino. The Honorable Mayor Thomas M. Menino state of the city address, January 2012.

[100] Brian Brubach, Aravind Srinivasan, and Shawn Zhao. Meddling metrics: the effects of measuring and constraining partisan gerrymandering on voter incentives. In *ACM Conference on Economics and Computation (EC)*, 2020.

[101] United States Senate. S. 1745 – 102nd Congress: Civil Rights Act of 199, 1991. `https://www.govtrack.us/congress/bills/102/s1745`.

[102] Supreme Court of the United States. 13-1371 – Texas Department of Housing and Community Affairs v. The Inclusive Communities Project, Inc., January 2015.

[103] Samir Khuller and Yoram J. Sussmann. The capacitated k-center problem. In *European Symposium on Algorithms (ESA)*, 1996.

[104] Cristina G. Fernandes, Samuel P. de Paula, and Lehilton L. C. Pedrosa. Improved approximation algorithms for capacitated fault-tolerant k-center. *Algorithmica*, 2018.

[105] Rong Ge, Martin Ester, Byron J. Gao, Zengjian Hu, Binay Bhattacharya, and Boaz Ben-Moshe. Joint cluster analysis of attribute data and relationship data: The connected k-center problem, algorithms and applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2008.

[106] Samir Khuller, Robert Pless, and Yoram J. Sussmann. Fault tolerant k-center problems. *Theoretical Computer Science*, 2000.

[107] Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. (APPROX/RANDOM)*, 2008.

[108] Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The non-uniform k-center problem. In *International Conference on Automata, Languages, and Programming (ICALP)*, 2016.

[109] Andrew Lim, Brian Rodrigues, Fan Wang, and Zhou Xu. k-center problems with minimum coverage. In *Computing and Combinatorics*, 2004.

[110] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In *Annual Symposium on Theory of Computing (STOC)*, 1997.

[111] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Annual Symposium on Theory of Computing (STOC)*, 2003.

[112] Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. In *Automata, Languages and Programming*, 2001.

[113] Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. *International Conference on Automata, Languages, and Programming (ICALP)*, 2018.

[114] Suman K Bera, Deeparnab Chakrabarty, and Maryam Negahbani. Fair algorithms for clustering. In *Neural Information Processing Systems (NeurIPS)*, 2019.

[115] Matthäus Kleindessner, Samira Samadi, Pranjal Awasthi, and Jamie Morgenstern. Guarantees for spectral clustering with fairness constraints. In *International Conference on Machine Learning (ICML)*, 2019.

[116] Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *International Conference on Machine Learning (ICML)*, 2019.

[117] Richard Zemel, Yu Wu, Kevin Swersky, Toniann Pitassi, and Cynthia Dwork. Learning fair representations. In *International Conference on Machine Learning (ICML)*, 2013.

[118] David G. Harris, Shi Li, Thomas Pensyl, Aravind Srinivasan, and Khoa Trinh. Approximation algorithms for stochastic clustering. *Journal of Machine Learning Research*, 2019.

[119] Christopher Jung, Sampath Kannan, and Neil Lutz. Service in Your Neighborhood: Fairness in Center Location. In *Symposium on Foundations of Responsible Computing (FORC)*, 2020.

[120] Sepideh Mahabadi and Ali Vakilian. (individual) fairness for k-clustering. In *International Conference on Machine Learning (ICML)*, 2020.

[121] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Annual Symposium on Theory of Computing (STOC)*, 1998.

[122] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *International Conference on Very Large Data Bases (VLDB)*, 1999.

[123] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Annual Symposium on Computational Geometry (SoCG)*, 2004.

[124] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 1985.

[125] Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 1993.

[126] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Annual Symposium on Foundations of Computer Science (FOCS)*, 1996.

[127] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Annual Symposium on Theory of Computing (STOC)*, 2003.

[128] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 1990.

[129] J.E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 1985.

[130] Jurij MiheliÄIJ and Borut Robic. Solving the k-center problem efficiently with a dominating set algorithm. *CIT*, 2005.

[131] Jesus Garcia-Diaz, Jairo Sanchez-Hernandez, Ricardo Menchaca-Mendez, and Rolando Menchaca-Mendez. When a worse approximation factor gives better performance: a 3-approximation algorithm for the vertex k-center problem. *Journal of Heuristics*, 2017.

[132] Mark Daskin. A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results. *Communications of the Operations Research Society of Japan*, 2000.

[133] Sourour Elloumi, Martine Labbé, and Yves Pochet. A new formulation and resolution method for the p-center problem. *INFORMS Journal on Computing*, 2004.

[134] Nenad Mladenovic, Martine Labbé, and Pierre Hansen. Solving the p-center problem with tabu search and variable neighborhood search. *Networks*, 2003.

[135] Ronny Kohavi and Barry Becker. UCI machine learning repository, 1996.

[136] Brian Brubach, Aravind Srinivasan, and Shawn Zhao. Meddling metrics: the effects of measuring and constraining partisan gerrymandering on voter incentives. In *ACM Conference on Economics and Computation (EC)*, 2020.

[137] Ivan Ryan and Warren D. Smith. Splitline districtings of all 50 states + DC + PR. `https://rangevoting.org/SplitLR.html`. [Online; accessed 15-August-2019].

[138] Maria Chikina, Alan Frieze, and Wesley Pegden. Assessing significance in a markov chain without mixing. *Proceedings of the National Academy of Sciences*, 2017.

[139] Gregory Herschlag, Han Sung Kang, Justin Luo, Christy Vaughn Graves, Sachet Bangia, Robert Ravier, and Jonathan C Mattingly. Quantifying gerrymandering in north carolina. *arXiv preprint arXiv:1801.03783*, 2018.

[140] Wendy K Tam Cho and Yan Y Liu. Sampling from complicated and unknown distributions: Monte carlo and markov chain monte carlo methods for redistricting. *Physica A: Statistical Mechanics and its Applications*, 2018.

[141] Wendy K Tam Cho. Technology-enabled coin flips for judging partisan gerrymandering. *Southern California law review*, 2019.

[142] League of Women Voters of Pennsylvania v. Commonwealth of Pennsylvania, No. 159 MM (2018).

[143] Allan Gibbard et al. Manipulation of voting schemes: a general result. *Econometrica*, 1973.

[144] Mark Allen Satterthwaite. Strategy-proofness and arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of economic theory*, 1975.

[145] Lily Hu, Nicole Immorlica, and Jennifer Wortman Vaughan. The disparate effects of strategic manipulation. In *ACM Conference on Fairness, Accountability, and Transparency (ACM FAccT)*, 2019.

[146] Allan Borodin, Omer Lev, Nisarg Shah, and Tyrone Strangway. Big city vs. the great outdoors: voter distribution and how it affects gerrymandering. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018.

[147] Arizona State Legislature v. Arizona Independent Redistricting Commission, No. 13-1314, 576 U.S. ___ (2015).

[148] United States Census Bureau. My Congressional District. `https://www.census.gov/mycd/`, 2017. [Online; accessed 30-August-2019].

[149] Moon Duchin. Geometry v. gerrymandering. *Scientific American*, 2018.

[150] Shaw v. Reno, No. 92-357, 509 U.S. 630 (1993).

[151] Moon Duchin, Taissa Gladkova, Eugene Henninger-Voss, Ben Klingensmith, Heather Newman, and Hannah Wheelen. Locating the representational baseline: Republicans in massachusetts. *arXiv preprint arXiv:1810.09051*, 2018.

[152] Nicholas Stephanopoulos and Eric McGhee. Partisan gerrymandering and the efficiency gap. *University of Chicago Law Review*, 2015.

[153] Gill v. Whitford, No. 16-1161, 585 U.S. ___ (2018).

[154] Mira Bernstein and Moon Duchin. A formula goes to court: Partisan gerrymandering and the efficiency gap. *Notices of the AMS*, 2017.

[155] Benjamin Plener Cover. Quantifying partisan gerrymandering: An evaluation of the efficiency gap proposal. *Stan. L. Rev.*, 2018.

[156] Aaron Bycoffe, Ella Koeze, David Wasserman, and Julia Wolfe. The Atlas Of Redistricting. `https://projects.fivethirtyeight.com/redistricting-maps/`, 2018. [Online; published 25-January-2018; accessed 15-August-2019].

[157] Jowei Chen, Jonathan Rodden, et al. Unintentional gerrymandering: Political geography and electoral bias in legislatures. *Quarterly Journal of Political Science*, 2013.

[158] Vieth v. Jubelirer, No. 02-1580, 541 U.S. 267 (2004).

[159] Wesley Pegden, Ariel D Procaccia, and Dingli Yu. A partisan districting protocol with provably nonpartisan outcomes. *arXiv preprint arXiv:1710.08781*, 2017.

[160] Corbett A Grainger. Redistricting and polarization: Who draws the lines in california? *The Journal of Law and Economics*, 2010.

[161] Brian Brubach, Jay Ghurye, Mihai Pop, and Aravind Srinivasan. Better greedy sequence clustering with fast banded alignment. In *International Workshop on Algorithms in Bioinformatics (WABI)*, 2017.

[162] Brian Brubach and Jay Ghurye. A Succinct Four Russians Speedup for Edit Distance Computation and One-against-many Banded Alignment. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2018.

[163] Gerard Muyzer, Ellen C De Waal, and Andre G Uitterlinden. Profiling of complex microbial populations by denaturing gradient gel electrophoresis analysis of polymerase chain reaction-amplified genes coding for 16S rRNA. *Applied and environmental microbiology*, 1993.

[164] James R White, Saket Navlakha, Niranjan Nagarajan, Mohammad-Reza Ghodsi, Carl Kingsford, and Mihai Pop. Alignment and clustering of phylogenetic markers-implications for microbial diversity studies. *BMC bioinformatics*, 2010.

[165] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1994.

[166] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 2006.

[167] Robert C Edgar. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, 2010.

[168] Mohammadreza Ghodsi, Bo Liu, and Mihai Pop. DNACLUST: accurate and efficient clustering of phylogenetic marker genes. *BMC bioinformatics*, 2011.

[169] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, 1966.

[170] Robert C Edgar. Updating the 97% identity threshold for 16s ribosomal rna otus. *Bioinformatics*, 2018.

[171] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 1981.

[172] Eugene W Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1986.

[173] William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 1980.

[174] William J. Masek and Michael S. Paterson. How to compute string-edit distances quickly. 1983.

[175] Gene Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 1999.

[176] Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

[177] Julie D Thompson, Toby Gibson, Des G Higgins, et al. Multiple sequence alignment using ClustalW and ClustalX. *Current protocols in bioinformatics*, 2002.

[178] DOTREE Plotree and DOTGRAM Plotgram. PHYLIP-phylogeny inference package (version 3.2). *cladistics*, 1989.

[179] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 1990.

[180] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Annual Symposium on Theory of Computing (STOC)*, 2015.

[181] Karl Bringmann and Marvin Kunnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Annual Symposium on Foundations of Computer Science (FOCS)*, 2015.

[182] Youngho Kim, Joong Chae Na, Heejin Park, and Jeong Seop Sim. A space-efficient alphabet-independent four-russians' lookup table and a multithreaded four-russians' edit distance algorithm. *Theoretical Computer Science*, 2016.

[183] Maxime Crochemore, Gad M. Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM J. Comput.*, 2003.

[184] Yelena Frid and Dan Gusfield. A simple, practical and complete o-time algorithm for rna folding using the four-russians speedup. *Algorithms for Molecular Biology*, 2010.

[185] Claus-Peter Schnorr. An algorithm for transitive closure with linear expected time. *SIAM Journal on Computing*, 1978.

[186] Gregory Bard. Matrix inversion (or lup-factorization) via the method of four russians, in $\theta(n^3/logn)$ time. *LMS J. Comput. Math*, 2008.

[187] J. W. Fickett. Fast optimal alignment. *Nucleic Acids Res.*, 1984.

[188] Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1986.

[189] Esko Ukkonen. Algorithms for approximate string matching. *Inf. Control*, 1985.

[190] Klaus Schulz and Stoyan Mihov. Fast string correction with levenshtein-automata. *INTERNATIONAL JOURNAL OF DOCUMENT ANALYSIS AND RECOGNITION*, 2002.

[191] Stoyan Mihov and Klaus U. Schulz. Fast approximate search in large dictionaries. *Comput. Linguist.*, 2004.

[192] Ahmed Hassan, Sara Noeman, and Hany Hassan. Language independent text correction using finite state automata. In *International Joint Conference on Natural Language Processing*, 2008.

[193] Gene Myers. A four russians algorithm for regular expression pattern matching. *J. ACM*, 1992.

[194] Sun Wu, Udi Manber, and Eugene Myers. A subquadratic algorithm for approximate regular expression matching. *Journal of algorithms*, 1995.

[195] Jeanette P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM J. Comput.*, 1998.

[196] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 1987.

[197] Aggarwal and J. Park. Notes on searching in multidimensional monotone arrays. In *Annual Symposium on Foundations of Computer Science (FOCS)*, 1988.

[198] Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Trans. Algorithms (TALG)*, 2007.

[199] Krister M. Swenson, Mark Marron, Joel V. Earnest-Deyoung, and Bernard M. E. Moret. Approximating the true evolutionary distance between two genomes. *J. Exp. Algorithmics*, 2008.

[200] Xin Chen, Jie Zheng, Zheng Fu, Peng Nan, Yang Zhong, S. Lonardi, and Tao Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2005.

[201] Saeed Mehrabi. Approximating weighted duo-preservation in comparative genomics. In *Computing and Combinatorics*, 2017.

[202] RC Hardison. Comparative genomics. *PLoS Biol*, 2003.

[203] A.R. Mushegian. *Foundations of Comparative Genomics*. 2007.

[204] Bartlomiej Dudek, Pawel Gawrychowski, and Piotr Ostropolski-Nalewaja. A family of approximation algorithms for the maximum duo-preservation string mapping problem. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2017.

[205] Wenbin Chen, Zhengzhang Chen, Nagiza F. Samatova, Lingxi Peng, Jianxiong Wang, and Maobin Tang. Solving the maximum duo-preservation string mapping problem with linear programming. *Theoretical Computer Science*, 2014.

[206] Nicolas Boria, Adam Kurpisz, Samuli Leppänen, and Monaldo Mastrolilli. Improved approximation for the maximum duo-preservation string mapping problem. In *International Workshop on Algorithms in Bioinformatics (WABI)*, 2014.

[207] Nicolas Boria, Gianpiero Cabodi, Paolo Camurati, Marco Palena, Paolo Pasini, and Stefano Quer. A 7/2-approximation algorithm for the maximum duo-preservation string mapping problem. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2016.

[208] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. In *Analysis and Design of Algorithms for Combinatorial Problems*. 1985.

[209] Yao Xu, Yong Chen, Guohui Lin, Tian Liu, Taibo Luo, and Peng Zhang. A (1.4 + epsilon)-approximation algorithm for the 2-max-duo problem. In *International Conference on Algorithms and Computation (ISAAC)*, 2017.

[210] Stefano Beretta, Mauro Castelli, and Riccardo Dondi. Parameterized tractability of the maximum-duo preservation string mapping problem. *CoRR*, 2015.

[211] Christian Komusiewicz, Mateus de Oliveira Oliveira, and Meirav Zehavi. Revisiting the parameterized complexity of maximum-duo preservation string mapping. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2017.

[212] Marek Chrobak, Petr Kolman, and Jiří Sgall. The greedy algorithm for the minimum common string partition problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. (APPROX/RANDOM)*, 2004.

[213] Avraham Goldstein, Petr Kolman, and Jie Zheng. Minimum common string partition problem: Hardness and approximations. In *International Conference on Algorithms and Computation (ISAAC)*, 2004.

[214] Petr Kolman and Tomasz Waleń. Reversal distance for strings with duplicates: Linear time approximation using hitting set. In *Workshop on Approximation and Online Algorithms (WAOA)*, 2007.

[215] Petr Kolman and Tomasz Waleń. Approximating reversal distance for strings with bounded number of duplicates. *Discrete Applied Mathematics*, 2007.

[216] Laurent Bulteau, Guillaume Fertin, Christian Komusiewicz, and Irena Rusu. A fixed-parameter algorithm for minimum common string partition with few duplications. In *International Workshop on Algorithms in Bioinformatics (WABI)*, 2013.

[217] Peter Damaschke. Minimum common string partition parameterized. In *International Workshop on Algorithms in Bioinformatics (WABI)*, 2008.

[218] Haitao Jiang, Binhai Zhu, Daming Zhu, and Hong Zhu. Minimum common string partition revisited. *Journal of Combinatorial Optimization*, 2012.

[219] Laurent Bulteau and Christian Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014.

[220] S. M. Ferdous and M. Sohel Rahman. Solving the minimum common string partition problem with the help of ants. In Ying Tan, Yuhui Shi, and Hongwei Mo, editors, *Advances in Swarm Intelligence: 4th International Conference, ICSI 2013, Harbin, China, June 12-15, 2013, Proceedings, Part I*, 2013.

[221] Christian Blum, José A. Lozano, and Pinacho Davidson. Mathematical programming strategies for solving the minimum common string partition problem. *European Journal of Operational Research*, 2015.

[222] S. M. Ferdous and M. Sohel Rahman. An integer programming formulation of the minimum common string partition problem. *PLoS ONE*, 2015.

[223] Isaac Goldstein and Moshe Lewenstein. Quick greedy computation for minimum common string partition. *Theoretical Computer Science*, 2014.

[224] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 2014.

[225] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 1987.

[226] James B. Orlin. Max flows in O(nm) time, or better. In *Annual Symposium on Theory of Computing (STOC)*, 2013.

[227] Haim Kaplan and Nira Shafrir. The greedy algorithm for edit distance with moves. *Information Processing Letters*, 2006.

[228] Pankaj K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *Annual Symposium on Theory of Computing (STOC)*, 2014.