Algorithms for Matching Problems Under Data Accessibility Constraints

Oussama Hanguir

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2022

Abstract

Algorithms for Matching Problems Under Data Accessibility Constraints

Oussama Hanguir

Traditionally, optimization problems in operations research have been studied in a complete information setting; the input/data is collected and made fully accessible to the user, before an algorithm is sequentially run to generate the optimal output. However, the growing magnitude of treated data and the need to make immediate decisions are increasingly shifting the focus to optimizing under incomplete information settings. The input can be partially inaccessible to the user either because it is generated continuously, contains some uncertainty, is too large and cannot be stored on a single machine, or has a hidden structure that is costly to unveil. Many problems providing a context for studying algorithms when the input is not entirely accessible emanate from the field of matching theory, where the objective is to pair clients and servers or, more generally, to group clients in disjoint sets. Examples include ride-sharing and food delivery platforms, internet advertising, combinatorial auctions, and online gaming.

In this thesis, we study three different novel problems from the theory of matchings. These problems correspond to situations where the input is hidden, spread across multiple processors, or revealed in two stages with some uncertainty. In particular, we present in Chapter 1 the necessary definitions and terminology for the concepts and problems we cover. In Chapter 2, we consider a two-stage robust optimization framework that captures matching problems where one side of the input includes some future demand uncertainty. We propose two models to capture the demand uncertainty: explicit and implicit scenarios. Chapters 3 and 4 see us switch our attention to matchings in hypergraphs. In Chapter 3, we consider the problem of learning hidden hypergraph matchings through membership queries. Finally, in Chapter 4, we study the problem of finding matchings in uniform hypergraphs in the massively parallel computation (MPC) model where the data (e.g. vertices and edges) is distributed across the machines and in each round, a machine performs local computation on its fragment of data, and then sends messages to other machines for the next round.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to thank my advisor, Clifford Stein, without whom this dissertation would not have been possible. Cliff taught me myriad valuable lessons on how to conduct relevant and fruitful research. His keen sense of analysis and synthesis, deep knowledge of the state-of-the-art, and limitless intellectual curiosity have helped me grow academically and personally. Thank you for your guidance and benevolence throughout these years.

I would also like to thank my thesis committee members: Vineet Goyal, Yuri Faenza, Eric Balkanski, and Will Ma. Their help throughout this PhD has made my research experience better in many ways. Vineet: I had the good fortune of working with you during the central years of my PhD. I have learned a lot from you on matchings and combinatorial optimization, and I am glad to see the results of our collaboration in Chapter 2 of this thesis. Yuri: thank you for teaching me Optimization II. Your excellent class on combinatorial optimization has given me the taste of graph problems and significantly nurtured my early interests in the field. Eric: I enjoyed taking your class on machine Learning for algorithm design. I am delighted that it led to our collaboration in Chapter 3, especially since it meant working on a very different type of problems. Thank you for your advice, encouragement, and spot-on ideas. Will: thank you for helping me explore the interface between video games and optimization, for the time you invested in our collaboration, and for the constructive feedback on my writing and presentations. Working with you was as much an enjoyment as a learning experience.

I express my thanks and gratitude to my friend and collaborator, Omar El Housni. I have learned immensely from you during the last five years. Thank you for being my mentor and for your help and advice both academically and in life in general. I want to thank Christopher Thomas Ryan as well; you have been a great mentor and collaborator. Thank you for always being available and for the fascinating discussions we shared.

Throughout my internships, I had the chance to work and learn from some fantastic people. My thanks go to Andrew Goldberg, Julien Michel, Titouan Jehl, and Baptiste Richard for their teachings, guidance, and life advice.

I learned a great deal from my fellow PhD students in IEOR. I will miss the incredible moments we have shared, from taking classes together to collaborating on research projects, sightseeing in New York, or just grabbing a cup of coffee. Thank you Shatian, Lingyi, Mali, Agathe, Steven, Achraf, Omar Mouchtaki, Hersh, Xuan, Goutham, Yuan, Mike, Noémie, and Jacob. My thanks also go to the amazing IEOR staff, past and present, for their dedicated presence and helpfulness, especially Lizbeth, Kristen, and Winsor.

A thesis cannot come to an end without moments of stress and self-doubt, but I have been very lucky to be surrounded by great people who stood by my side. Through their support and understanding, Mathilde, Zakariae, Jad, Ali, Hassan, Amine, Othmane, Rashed, Sarah, and Latifa, have kept me grounded and focused on the most important goal. I owe you enormous gratitude and recognition. Your backing has gone beyond this PhD and affected numerous aspects of my life journey.

Last but not least, I would like to thank my family for always encouraging me. I thank my sisters Hajar and Oumayma for their constant motivation and cheerfulness. Finally, from the bottom of my heart and soul, I would like to thank my parents Said and Mina for their unconditional love and continuous support. I dedicate my efforts to them.

Introduction

Traditionally, optimization problems in operations research have been studied in a complete information setting; the input/data is collected and made fully accessible to the user before an algorithm is sequentially run to generate the optimal output. However, the growing magnitude of treated data and the need to make immediate decisions are increasingly shifting the focus to optimizing under incomplete information settings. Such settings can take multiple forms: The input can be not entirely accessible to the user, either because 1) it's generated continuously and contains some uncertainty, 2) it's too large and cannot be stored on one single machine, 3) it has a hidden structure that is costly to unveil. These modern computational paradigms motivate the renewed interest in computation under uncertainty and under memory constraints.

Many problems providing a context for studying algorithms when the input is not entirely accessible emanate from the field of matching theory, where the objective is to pair clients and servers or, more generally, to group clients into disjoint sets. Examples include ride-sharing and food delivery platforms, internet advertising, combinatorial auctions, and online gaming. The study of matchings has historically played a key role in the development of influential algorithms for the optimization of other combinatorial problems. Furthermore, due to the general nature of matching problems, studying them under data accessibility constraints can provide similar fundamental insights for other optimization problems in similar settings [1, 2, 3].

In this thesis, we study three different novel problems from the theory of matchings. These problems correspond to situations where the input is hidden, spread across multiple processors, or revealed in two stages with some uncertainty. Our goal is to shed light on settings where learning or computing a matching is a challenging problem. We design and analyze learning and approximations algorithms with theoretical guarantees for each of these settings. In addition to theoretical results, we provide practical implementations for our algorithms on real-life data to understand their power and limitations and to gain more insights towards more efficient solutions. In the rest of this introduction, we motivate the settings of the problems we study. We then present the outline of this thesis and highlight our key contributions in each chapter.

The online and uncertain nature of the input, as well as the irrevocability of decisions, can manifest in multiple matching, scheduling or resource allocation applications. For example, to schedule a set of jobs, it is not feasible to wait for all jobs to arrive to compute the schedule that minimizes makespan. The jobs have to be scheduled either immediately or at most a short while after they come. Another example can be seen in advertising allocation, where users need to be matched to ads as soon as they open a webpage. Multiple paradigms and models are adopted to deal with these settings, such as online algorithms [4, 5], fully dynamic algorithms [6, 7], and robust optimization algorithms [8, 9, 10]. Robust optimization is a particularly widely used paradigm to handle uncertainty, where we consider an adversarial model of uncertainty, and the goal is to optimize over the worst-case realization from the uncertainty set. This setting arises in many business applications where real-time decisions are based on the information revealed thus far. For instance, in ride-sharing dispatch systems, a subset of matching decisions need to be made before the platform observes the uncertain demand in subsequent minutes. In Chapter 2, we study this problem of matching with uncertainty through the lens of a two-stage robust optimization model. Our goal is to match the current clients to servers (in the first stage) so that the cost of first stage matching and the worst-case cost over all scenarios for the second stage matching is minimized.

In the ride-sharing application of Chapter 2, we can naturally model the problem using a bipartite graph with drivers on one side and riders on the other. However, while graphs can provide a practical structure to capture and model a wide variety of problems, the order they impose on the data can be restrictive. For example, some social interactions are often complex and supra-dyadic, as opposed to dyadic or 2-ary relationships. In shared rides, travelers can share rides instead of each having their own vehicle on the road, which can reduce congestion and operating costs. However, in order to compute a matching with shared rides, one has to consider the possibility that more than three riders might be matched to the same driver. One way to generalize the concept of graphs and capture such complex relationships is to allow an edge to connect an arbitrary number of vertices. This example motivates the second part of our thesis, where we focus on the generalization of matchings to hypergraphs. In hypergraphs, an edge can join any number of vertices, and a matching is a set of edges in which every two edges are disjoint. We start in Chapter 3, by studying the problem of learning a hypergraph matching via edge detecting queries. In this problem, a learner queries subsets of vertices of a hidden hypergraph and observes whether these subsets contain an edge or not. Hypergraph learning via edge detecting queries has relevant applications in chemical reaction networks where, given a set of chemicals, the goal is to learn which subsets of chemicals react using a small number of experiments.

The abstract structure of hypergraphs gives important flexibility to model a vast range of real-world applications. However, this same structure means that sometimes hypergraphs can contain a huge number of edges, making it challenging to handle them on one processor. In general, this is another limitation of the classic paradigm of sequential algorithms; the growth of memory and speed of individual computers is being outpaced by the increasing amount of data that needs to be processed. Even if the speed at which sequential processors perform computations has been improving exponentially in recent year, the cost of this improvement is also growing. Furthermore, the need for faster algorithms to solve large-size optimization keeps increasing in a variety of applications. As a consequence, a considerable amount of research have sought more cost-effective improvements by adopting *parallel* computation paradigms. The study of optimization problems in parallel computation models can be traced back to *Parallel Random Access Machine* (PRAM) algorithms of 1980s [11, 12, 13, 14]. The PRAM model is an extension of the RAM model of sequential computation, where a number of processors operate synchronously and are able to randomly access a large shared memory. Since then, other models have been used, such as CONGEST, which is a distributed message-passing model with a processor assigned to each vertex and a limit on the amount of information sent along each edge per round [15]. The LOCAL model, introduced by Linial [16], consisted of a relaxation of CONGEST that allows for an arbitrary amount of data sent along each edge. Finally, the *MapReduce* framework was introduced in 2008 [17], and has since been one of the most popular frameworks for parallel computation. In Chapter 4, we adopt an abstraction of the MapReduce framework named the Massively Parallel Computation (MPC) model, and study the maximum cardinality matching problem in hypergraphs where all edges have the same size (called uniform hypergraphs). In the MPC model, the data (e.g., vertices and edges) is distributed across the machines, and in each round, a machine performs local computation on its fragment of data, and then sends messages to other machines for the next round.

Outline and contributions. Chapter 1 of this thesis presents the relevant definitions and concepts that we will regularly use in the following chapters. We present the most common settings for matching problems on graphs as well as hypergraphs, and we conclude by presenting the MPC parallel computation framework. The rest of the thesis is split into two main parts:

1. **Part 1: Two-stage matchings in graphs.** This part consists of Chapter 2 and is based on our work in [18]. In this chapter, we initiate the study of a *two-stage robust* framework for matching problems. In this setting, we are given a set of drivers D, a set of first stage riders R_1, a universe of potential second-stage riders R_2, and a set of second-stage scenarios $\mathcal{S} \subseteq P(R_2)$. We are given a metric distance between drivers and riders, and the goal is to find a subset of drivers $D_1 \subseteq D$ to match all the first stage riders R_1 such that the sum of the cost of first stage matching and worst-case cost of second stage matching (between $D \setminus D_1$ and the riders in the second stage scenario) is minimized. We primarily focus on the case where the first stage cost is the average weight of matching between D_1 and R_1, and the second stage cost is the bottleneck matching cost between $D \setminus D_1$ and scenario $S \in \mathcal{S}$. We refer to this variant as the *Two-Stage Robust Matching Bottleneck Problem* (**TSRMB**), but our results extend easily to other settings. We investigate this problem using the two common approaches to model uncertainty in robust optimization problems: either explicitly

enumerate all the realizations of the uncertain parameters, or specify them implicitly by a set of constraints over the uncertain parameters. For explicit uncertainty, we show that TSRMB is NP-hard even for two scenarios and NP-hard to approximate within a factor better than 2 for three or more scenarios. For the case of implicit uncertainty, we show that even when the number of scenarios is small, the problem is NP-hard to approximate within a factor better than 2. Given these hardness results, we focus on designing approximation algorithms for the TSRMB problem. We show that the greedy and myopic approach can be arbitrarily bad even on one scenario, and present a constant approximation algorithm for any constant number of explicit scenarios. We also tackle the problem under implicit uncertainty and present constant approximation algorithms under different mild assumptions. We conclude this part with an experimental study, where we implement our algorithms and test them on real-life taxi data. Our experimental results show that our two-scenarios algorithm improves significantly upon the greedy algorithm both in and out of sample.

2. **Part 2: Matchings in hypergraphs.** In the second part of this thesis, we focus on the generalization of matchings to hypergraphs. As we touched upon, hypergraphs are a powerful tool to capture more complex relationships that cannot be modeled on graphs, and matchings on hypergraphs (also called hypermatchings) arise in many business applications such as ride-sharing and airline scheduling.

   - In Chapter 3, we consider the problem of reconstructing/learning hypermatchings by using membership queries. In this problem, we (the learner) are given the vertex set V of a hidden hypergraph H and an edge-detecting oracle Q_H. Given a query $S \subseteq V$, the oracle answers $Q_H(S) = 1$ if S contains an edge from H; otherwise, $Q_H(S) = 0$. Our goal is to learn the edge set of H using a small number of queries [19]. When queries can be evaluated in parallel, we are interested in algorithms with low adaptivity. Adaptivity is measured by the number of sequential rounds an algorithm makes where, in each round, queries are evaluated in parallel. The applications of this problem to molecular biology have motivated a long line of work on hypergraph learning in the edge-detecting queries model. Still, previous attempts to learn general hypergraphs have been shown to necessitate an exponential number of queries. In Chapter 3, we identify non-trivial families of hypergraphs that can be learned without suffering from a query complexity that grows exponentially in the size of the edges. We show that hypermatchings and low-degree near-uniform hypergraphs with n vertices are learnable with $\text{poly}(n)$ queries. We focus on learning hypermatchings and give an $O(\log^3 n)$-round algorithm with $O(n \log^5 n)$ queries. We complement this upper bound by showing that there are no algorithms with $\text{poly}(n)$ queries that learn hypermatchings in $o(\log \log n)$ adaptive rounds. The main contribution behind our algorithms in this chapter is the concept of *unique-edge covering family*. Previous work on learning a hypergraph relies on constructing a collection of sets of vertices that aim to identify *non-edges*, which are sets of vertices that do not contain any edge from the hidden hypergraph. These families have a minimum size that is exponential in the edge sizes, and previous approaches require a number of queries that is at least the size of these families. In contrast to these existing approaches that focus on non-edges, we construct a unique-edge covering family: a collection of sets of vertices such that every edge of the hidden hypergraph is contained in at least one set of the collection, and this set does not contain any other edges. Since the number of edges in a hypergraph with maximum degree Δ and n vertices is at most Δn, there exists unique-edge covering families whose sizes depend on the maximum degree Δ instead of the maximum edge size.

- In Chapter 4, we transition from learning hypermatchings to computing them. Particularly, we are interested in the problem of finding maximum cardinality matchings in very large hypergraphs, large enough that we cannot solve the problem on one computer [20]. This problem generalizes maximum matchings in graphs and has numerous applications in partitioning problems, where we need to partition elements under strong constraints on what is an allowable partition. Examples include scheduling airline flight crews to airplanes as well as determining the winners in combinatorial auctions. We consider d-*uniform* hypergraphs where every hyperedge has the same cardinality d. In the d-Uniform Hypergraph Matching Problem (d-UHM), a d-uniform hypergraph is given, and one needs to find the maximum cardinality matching. We design and implement the first algorithms for the d-UHM in the MPC model. We will give three different algorithms, demonstrating different trade-offs between the model's parameters. Our algorithms are inspired by methods to find maximum matchings in graphs, but require developing significant new tools to address hypergraphs. Our contributions include: 1) A $O(\log n)$-round d-approximation algorithm that uses $O(nd)$ space per machine, 2) a 3-round, $O(d^2)$-approximation algorithm that uses $\tilde{O}(\sqrt{nm})$ space per machine, and 3) a 3-round algorithm that computes a subgraph containing a $(d-1+\frac{1}{d})^2$-approximation, using $\tilde{O}(\sqrt{nm})$ space per machine for linear hypergraphs, and $\tilde{O}(n\sqrt{nm})$ in general. For the third algorithm, we introduce the concept of *HyperEdge Degree Constrained Subgraph* (HEDCS), which can be of independent interest. We show that an HEDCS contains a fractional matching with a total value at least $|M^*|/(d-1+\frac{1}{d})$, where $|M^*|$ is the size of the maximum matching in the hypergraph. Moreover, we investigate the experimental performance of these algorithms both on random input and real instances. Our results support the theoretical bounds and confirm the trade-offs between the quality of approximation and the speed of the algorithms.

Chapter 1: An Introduction to Matchings and Hypergraphs

Optimization problems on graphs constitute an important class of combinatorial optimization problems. These graph problems arise in a wide range of fields, such as network design, facility location, scheduling, and clustering in data science applications. In his famous paper on the "Reducibility Among Combinatorial Problems" [21], Karp designates 10 graph problems among the total 21 decision problems he shows to be NP-complete. The ubiquity of using graphs to model decisions even extends to some of the remaining 11, which can also be formulated or easily transformed into graph problems.

Typically, in a graph optimization problem, we are given an undirected or directed graph with vertices (or nodes) and edges (or arcs). To every vertex/edge (or both), we are given a corresponding weight, which can be deterministic or uncertain. The goal is to select a subset of vertices/edges that satisfy some specific structure (cut, spanning tree, matching, etc.) such that the total weight of the chosen vertices/edges is optimized.

In this thesis, we focus on the specific structure of matchings, which requires that the selected edges are vertex disjoint. Graph matching is one of the most well-studied problems in combinatorial optimization. Matchings found their earliest motivation in transportation cost minimization [22], as well as optimizing assignments of jobs to candidates [23]. In subsequent years and decades, applications of matchings have arisen in many fields such as scheduling [24], ride-sharing platforms [18, 25], quantum computing [26], stable marriages [27, 28], and chemistry [29]. In the rest of this preliminary chapter, we present formal definitions for graphs and hypergraphs, as well as some of the most known matching problems that are relevant to our contributions. We finally introduce the parallel computation model under which we consider our hypergraph matching problem from Chapter 4.

1.1 Matchings in graphs

Definition 1.1.1 (Graph). *An undirected graph $G = (V, E)$ is a structure consisting of a finite set V of vertices (also known as nodes) and a finite set $E \subseteq V \times V$ of edges such that each edge $e = (i, j)$ is associated with the pair of vertices i and j. If each edge e in graph G has an associated weight w_e , the graph G is called a weighted graph.*

Definition 1.1.2 (Bipartite graph). *A graph $G = (V, E)$ is bipartite if the vertex set V can be partitioned into two sets A and B, such that every edge contains exactly one vertex from A and one vertex from B, i.e. $E \subseteq A \times B$. In this case the graph is denoted $G = (A, B, E)$.*

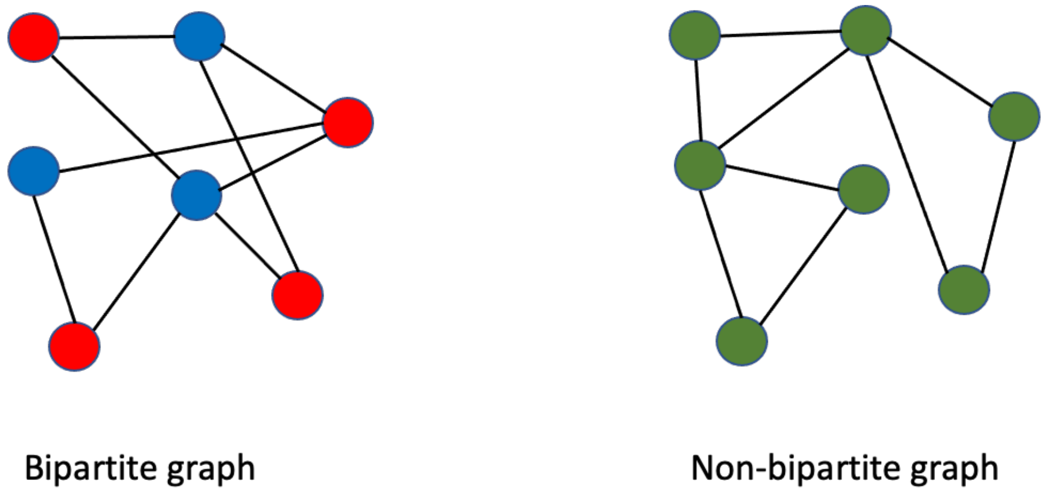

Figure 1.1: Example of a bipartite and a non-bipartite graph.

Definition 1.1.3 (Degree). *The degree of a node is the number of edges for which it is one of the endpoints. The maximum degree of a graph is the maximum degree over all of the nodes in the graph.*

Definition 1.1.4 (Matching). *Let $G = (V, E)$ be a graph. A matching in G is a set of edges $M \subseteq E$ such that for every $e, e' \in M$, there is no vertex v such that e and e' are both incident on v. The matching M is called perfect if for every $v \in V$, there is some $e \in M$ which is incident on v.*

Definition 1.1.5 (Maximal Matching). *A maximal matching is a matching to which no more edges can be added without increasing the degree of one of the nodes to two; it is a local maximum.*

1.1.1 Maximum Cardinality Matching

Maximizing the cardinality of matching is a fundamental problem in graph theory [30]. In this problem, we are given a graph G, and the objective is to compute a matching that contains the maximum number of edges. This problem has many applications in a variety of fields, such as image recognition [31], chemical structure analysis [32], and bioinformatics [33]. An important special case of the maximum cardinality matching problem is when G is bipartite. In this case, the problem can be efficiently solved with simpler algorithms than in the general case.

On bipartite graphs, we can efficiently compute a maximum matching using the Hopcroft–Karp algorithm (sometimes called the Hopcroft–Karp–Karzanov algorithm) [34, 35]. This algorithm repeatedly increases the size of an initial matching by finding augmenting paths. These paths are sequences of edges that alternate between edges in the matching and edges out of the matching, and where the initial and final edges are not in the partial matching. The Ford–Fulkerson algorithm [36], a greedy algorithm that computes the maximum flow in a flow network, can also be used to compute a maximum matching on bipartite graphs by converting them into flow networks.

On general, not necessarily bipartite graphs, the problem is more challenging. The blossom algorithm, developed by Jack Edmonds [37], similarly improves an initial empty matching by finding augmenting paths. Unlike bipartite matching, this algorithm deals with the possibility of an odd-length cycle in the graph (blossom). Blossoms are contracted to a single vertex, with the search continuing iteratively in the contracted graph.

1.1.2 Maximum Weight Matching

In the maximum weight matching problem, we are given a weighted graph, and the objective is to compute a matching in which the sum of edge weights is maximized. A special case of this problem is finding a maximum cardinality matching on an unweighted graph (equivalent to all edges having the same weight). This problem has various applications, such as computer vision [38], and chess tournaments [39].

The most common approach to solving the maximum weight matching problem in bipartite graphs is the primal-dual method, called the Hungarian method (Kuhn [40]). For general graphs, Edmonds [41] designed a $O(n^2m))$ time algorithm to find a maximum weight matching. A refined implementation of Edmonds' algorithm, by Gabow [42], runs in $O(mn + n^2 \log n)$ time. The idea is to build up feasible primal and dual solutions simultaneously and show that in the end, both solutions satisfy complementary slackness conditions, and hence by the duality theorem, the primal solution is a maximum weight matching. More elaborate and faster algorithms to both exactly solve and approximate the problem have been presented since (see [43, 44, 45]).

Another weighted matching problem is the minimum weight bipartite perfect matching problem, where, given a bipartite graph and a weight function w, we need to compute a perfect matching with minimum weight. This problem is sometimes also referred to as the assignment problem. The minimum weight perfect matching problem can be easily reduced to the maximum weight matching problem by the following modification of the weight vector: for each edge e set the new weight $w'_e := M - w_e$ where M is a big number, and replace the weight vector w with w'.

1.1.3 Bottleneck Matching

When a graph is weighted, it is often desirable to compute a matching that is optimal with respect to some criterion. A minimum weight perfect matching minimizes the sum of the weights of the edges of the matching, a most uniform matching minimizes the difference between the maximum weight and the minimum weight of edges of the matching, and a minimum deviation matching minimizes the difference between the average weight and the minimum weight of the matching. A criterion that has received particular attention is the bottleneck objective. The bottleneck matching problem is to find a maximum cardinality matching M such that the largest edge weight in M is as small as possible.

An example application for the bottleneck matching problem is the threat seduction problem [46], where the objective is to lure a threat away from an asset by employing a decoy that emits a signature that is indistinguishable from the asset.

The bottleneck matching problem is well-studied, and there exist many efficient algorithms to solve it [47, 48, 49, 50, 51]. In [47], for example, a threshold algorithm is presented where an initial threshold is set, and the algorithm checks if an assignment can be created with allocation costs smaller than the threshold. The threshold is iteratively increased until a valid assignment is found, which then corresponds to the optimal bottleneck matching.

In general, a bottleneck optimization problem on a graph with edge costs is the problem of finding a subgraph of a certain kind that minimizes the maximum edge cost in the subgraph. The bottleneck objective contrasts with the more common objective of minimizing the sum of edge costs. Several bottleneck problems have been considered, e.g., shortest path problems [52, 53], spanning tree and maximum cardinality matching [49], and traveling salesman problems [54] (see [55] for a compilation of graph bottleneck problems).

1.1.4 Online Matching

In a number of settings, the input is not known in advance but is instead revealed incrementally, even as the algorithm makes its decisions based on currently available information. Examples of such settings can range from ski rental [56] to portfolio selection [57]. When studying online optimization problems, we are interested in the competitive ratio of an online algorithm, which is simply the approximation ratio achieved by the algorithm. That is, the worst-case ratio between the cost of the solution found by the algorithm and the cost of an optimal solution that knows the entire input in advance. A considerable amount of interest has gone towards studying online matching problems and their generalizations, driven by new applications of ad allocation in internet advertising and ride-sharing [58, 59].

In online bipartite matching, we are given a known set of *servers* while a set of *clients* arrive online and upon arrival, each client can be matched to a server irrevocably. The online maximum cardinality matching problem was first studied by Karp *et al.* [60] in the adversarial model where the graph is unknown; when a client arrives it reveals its incident edges. Karp *et al.* [60] and Birnbaum and Mathieu [61] proved that the simple randomized RANKING algorithm achieves $(1 - 1/e)$ competitive ratio and this factor is the best possible performance. Since then, many online variants have been studied in great depth (see survey [62]). Applications include problems like AdWords [63, 64, 65], vertex-weighted [66, 67], edge-weighted [68, 69], stochastic matching [70, 71, 72, 73], random vertex arrival [74, 75, 76, 77], and batch arrivals [78, 79, 80].

In the *online bipartite metric matching problem*, servers and clients correspond to points from a metric space. Upon arrival, each client must be matched to a server irrevocably, at a cost equal to their distance. The objective is to find the minimum weight maximum cardinality matching. For general metric spaces, Khullet *et al.* [81] and Kalyanasundaram and Pruhs [82] proved that there is a tight bound of $(2n-1)$ on the competitiveness factor of deterministic online algorithms, where n is the number of servers. This raised the questions as to whether randomization could help obtain an exponential improvement for general metric spaces. Meyerson, *et al.* [83] and Bansal *et al.* [84] provided poly-logarithmic competitive randomized algorithms for the problem. Recently, Raghvendra [85] presented a $O(\log n)$-competitive algorithm in the random arrival model. In Chapter 2, our two-stage model can be seen as an online problem with group arrivals.

1.1.5 Two-stage Matching

In several optimization problems, the input is not fully available at the start but has some uncertain part that is revealed over time, and we need to make decisions that take this uncertainty into account. Decision making under uncertainty is a fundamental problem that arises in numerous business applications such as capacity planning [86, 87, 88], where retailers make capacity decisions while demand is sequentially manifesting. In facility location problems, manufacturers need to decide on the location of the stores before they observe the demand from customers [89, 90]. To handle optimization problems under uncertainty, we usually adopt either methods from the realm of Stochastic Optimization or from Robust Optimization. In the stochastic optimization approach, uncertainty is modeled as a probability distribution, and the goal is to optimize an expected objective (see [91] for a detailed discussion). On the other hand, in the robust optimization approach, an adversarial model of uncertainty is used; parts of the input stem from a known uncertainty set, and the goal is to optimize over the worst-case realization from this uncertainty set. This approach increasingly developed as a popular model for hedging against uncertainty [92, 93], due to the fact that designing an uncertainty set from historical data is significantly less challenging than estimating a joint probability distribution, especially for high-dimensional uncertainty. Furthermore, robust optimization leads to a tractable approach where a feasible static solution can be computed efficiently for a large class of problems.

Two-stage robust models constitute the cornerstone of robust optimization problems. In this setting, the decision maker acts in two steps: they make a first stage decision, observe the revealed uncertainty and then make a second stage (or recourse) decision. Several combinatorial optimization problems have been studied in this model, including Set Cover and Capacity Planning, [87, 88], Facility Location [94], and Network flow [95].

Two-stage matching problems with uncertainty, however, have not been studied extensively. They have been considered in the stochastic setting with uncertainty over the edges [96, 97]. Matuschke *et al.* [98] considered a two-stage version of the uni-chromatic problem (where there is no distinction between servers and clients).

In Chapter 2, we initiate the study of a two-stage robust approach for matching problems. We study the hardness of approximation of our two-stage problem under different cost functions and present constant approximation algorithms in several settings for both the implicit and explicit models of uncertainty. Furthermore, we test our algorithms on real-life taxi data from the city of Shenzhen and show that they significantly improve upon classical greedy solutions.

1.2 Hypergraphs

In our ride-sharing application of Chapter 2, we can naturally model the problem using a bipartite graph with drivers on one side and riders on the other, and with edge weights being the distances between riders and drivers.

However, while graphs provide a practical structure that can capture and model most settings, the order that they impose on the data can be restrictive. For example, some social interactions are often complex and supra-dyadic, as opposed to dyadic or 2-ary relationships; they usually involve more than two actors or take place with multi-user groups [99]. For example, in coauthorship networks, multiple researchers can coauthor one or more papers. One way to generalize the concept of graphs and capture such complex relationships is to allow an edge to connect an arbitrary number of vertices. Such edges are called hyperedges and correspond to subsets of the vertex set. A set of vertices together with a family of hyperedges constitutes a hypergraph.

Definition 1.2.1 (Hypergraph)**.** *A hypergraph H is a pair $H = (V, E)$ where V is the set of vertices and E is the set of hyperedges. A hyperedge $e \in E$ is a nonempty subset of the vertices. The cardinality of a hyperedge is the number of vertices it contains. When every hyperedge has the same cardinality d, the hypergraph is said to be d-uniform.*

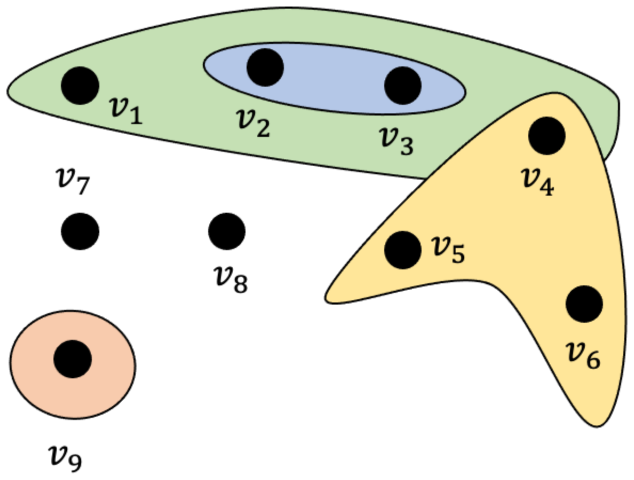

Figure 1.2: Example of hypergraph

Theoretically, hypergraphs are very abstract objects with a vastly general structure. Nonetheless, the last decades have seen a surge of using hypergraphs to model applications of real-world problems. Hypergraphs can be used in the context of transportation for vehicle-rotation-Planning [100, 101]. They have been adopted increasingly in the domain of artificial intelligence and machine learning [102, 103, 104]. Hypergraphs are also a powerful tool in computational chemistry and molecular biology where they are used to represent groups of chemicals and molecules that cause a reaction; vertices represent chemicals and molecules, and edges represent groups of vertices that cause a reaction [105, 106]. In [107] for example, it is shown that a comparison between different hypergraphs can help identify signs of viral infection. In the context of the COVID-19 pandemic, hypergraph modeling can help differentiate between COVID-19 and other lung diseases in the CT imaging procedure [108].

One disadvantage of the modeling power and flexibility of hypergraphs is that a number of combinatorial optimization problems that are solvable in polynomial time on graphs become NP-hard when considered on hypergraphs. The example we consider in Chapter 4 is the d-Uniform Hypergraph Matching (d-UHM) problem: given an n-vertex hypergraph H where every hyperedge is of size d, find a maximum cardinality set of disjoint hyperedges. For $d \geq 3$, the problem of finding the maximum matching is NP-complete and was one of Karp's 21 NP-complete problems [21]. This problem cannot be efficiently approximated within a factor better than $\Omega(d/\log d)$ unless P = NP [109].

We are interested in the problem of finding matchings in very large hypergraphs, large enough that we cannot solve the problem on one computer. We adopt a MapReduce-like model of modern parallel computation named the Massively Parallel Computation (MPC) model [110] (formally defined in the subsection below). This model is widely used to solve different graph problems such as matching, vertex cover [111, 112, 113, 114, 115], independent set [115, 116], as well as many other algorithmic problems. In this model, we consider approximation algorithms to find maximum matchings both on general and *linear* hypergraphs.

Definition 1.2.2 (Linear Hypergraph)**.** *A hypergraph is linear if the intersection of any two hyperedges has at most one vertex.*

1.3 MPC Computation Model

Many combinatorial optimization problems require the processing of graphs that contain tens of billions of edges, which can be too large to be contained on a single processor. A regular approach to handling large inputs and to scale algorithms is to run them in a distributed setting. This consists in partitioning the data among multiple processors that will perform computations in parallel.

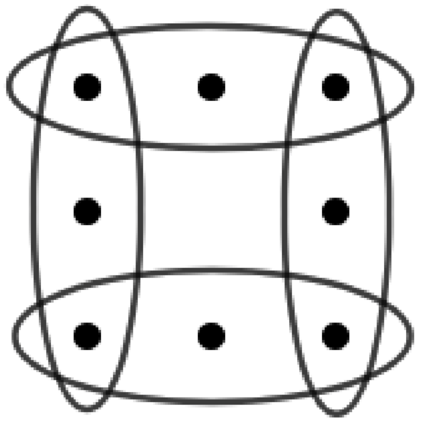

Figure 1.3: Example of a linear hypergraph.

MapReduce, a framework for implementing distributed algorithms, was introduced in 2008 [17], and has since been one of the most popular frameworks for parallel computation. A MapReduce program runs in a sequence of rounds. In each round, we transform a set of key-value pairs into another set of key-value pairs through main phases: 1) Map phase: specifying a map function that is applied to filter and sort the key-value pairs. 2) Reduce phase: performing a summary operation on the intermediate key-value pairs to produce the output of the round.

With the large-scale adoption of MapReduce and its variants such as Hadoop and Spark, the interest within the computer science community to develop a theoretical model that captures parallel computation frameworks has grown. The first was proposed by Karloff *et al.* [117]. Since then, multiple new models and variants along with new algorithms have been presented [118, 119, 111, 120]. After multiple modeling efforts, the community has converged on a model named the *Massively Parallel Computation (MPC)* model, which is believed to best capture parallel computation frameworks.

The MPC model is widely used to solve different graph problems such as matching, vertex cover [111], independent set [115, 116], as well as many other algorithmic problems [112, 113, 114, 115]. This model is often defined by three parameters: the input data size N, the number of machines k available for computation, and the memory size per machine of s. Usually, algorithms in this model satisfy $k \cdot s = \tilde{O}(N)$, which means that the total space in the system is only a polylogarithmic factor more than the input size. The computation proceeds in rounds. At the beginning of each round, the data (e.g., vertices and edges) is distributed across the machines. In each round, a machine performs local computation on its data (of size s), and then sends messages to other machines for the next round. Crucially, the total amount of communication sent or received by a machine is bounded by s, its space. For example, a machine can send one message of size s, or s messages of size 1. It cannot, however, broadcast a size s message to every machine. Each machine treats the received messages as the input for the next round.

In practice, the memory per machine s is less of a choice and more of a constraint, imposed by the problem instances and the available computational resources. Typically, s is polynomially smaller than N, e.g. $s = O(N^c)$ for some constant $c < 1$. Our model limits also the number of machines to be substantially sublinear in the size of the input. On the other hand, no restrictions are placed on the computational power of any individual machine. The main complexity measure is therefore the memory per machine and the number of rounds R required to solve a problem, which we consider to be the "parallel time" of the algorithm.

While parallel computing frameworks allow the processing of graphs/hypergraphs with trillions of edges, they also introduce new challenges. One main challenge is that each machine only sees a small piece of the input at a time, which creates the need to handle inter-machine communication very carefully to avoid redundant computations. In Chapter 4, we introduce multiple parallel algorithms to compute hypergraph matchings with approximations guarantees. We will see that our algorithms also highlight the possible trade-offs between the parameters of the MPC model.

Chapter 2: Two-Stage Robust Matching

In this chapter, we consider a two-stage robust optimization framework that captures matching problems where one side of the input includes some future demand uncertainty. We propose two models to capture the demand uncertainty: explicit and implicit scenarios. Our models are inspired from the ridesharing settings, where platforms need to match the riders (almost) as soon as the request arrives with only partial knowledge about future ride requests. This chapter is based on the article [18] written in collaboration with Omar El Housni, Vineet Goyal, and Clifford Stein. The article is published in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX)*, 2021.

2.1 Introduction

Matching demand (riders) with supply (drivers) is a fundamental problem for ride-hailing platforms such as Uber, Lyft and DiDi, who continually need to match drivers to current riders efficiently with only partial knowledge of future ride requests. An efficient matching results in lower wait times for riders and more business for drivers. With new riders arriving every few seconds, these plateforms need to constantly compute matchings between riders and drivers. A common approach in practice is batched matching: instead of matching each request sequentially as it arrives, aggregate the requests for a small amount of time (typically one to two minutes) and match all the requests to available drivers in one batch [121, 122, 80]. However, computing this batch matching myopically without considering future requests can lead to a highly sub-optimal outcome for some subsequent riders. Motivated by this shortcoming, and by the possibility of using historical data to hedge against future uncertainty, we study a two-stage framework for matching problems where the future demand uncertainty is modeled as a set of scenarios that are specified explicitly or implicitly. The goal is to compute a matching between the available drivers and current batch of riders such that the total worst-case cost of first stage and second stage matching is minimized. More specifically, we consider an adversarial model of uncertainty where the adversary observes the first stage matching of our algorithms and presents a worst-case scenario from the list of specified scenarios in the second stage. We primarily focus on the case where the first stage cost is the average weight of the first stage matching, and the second stage cost is the highest edge weight in the second stage matching. This is motivated by the goal of computing a low-cost first stage matching while also minimizing the waiting time for any ride in the worst-case scenario in the second stage. We also consider other metrics for the total cost and present related results.

Two-stage robust optimization is a popular model for hedging against uncertainty [92, 93]. Several combinatorial optimization problems have been studied in this model, including Set Cover and Capacity Planning, [87, 88], Facility Location [94] and Network flow [95]. Two-stage matching problems with uncertainty, however, have not been studied extensively. A review of the literature on matchings in both online and two-stage settings is presented in Section 2.2.

In this chapter, besides initiating the study of our two-stage robust approach for matching problems, we study the hardness of approximation of this problem under different cost functions and present constant approximation algorithms in several settings for both the implicit and explicit models of uncertainty. Furthermore, we test our algorithms on real-life taxi data from the city of Shenzhen and show that they significantly improve upon classical greedy solutions.

2.1.1 Problem definition.

We consider the following *Two-stage Robust Matching Problem*. We are given a set of drivers D, a set of first stage riders R_1, a universe of potential second stage riders R_2 and a set of second stage scenarios $\mathcal{S} \subseteq \mathcal{P}(R_2)$[1]. We are given a metric distance d on $V = R_1 \cup R_2 \cup D$. The goal is to find a subset of drivers $D_1 \subseteq D$ ($|D_1| = |R_1|$) to match all the first stage riders R_1 such that the sum of cost of first stage matching and worst-case cost of second stage matching (between $D \setminus D_1$

[1]$\mathcal{P}(R_2)$ is the power set of R_2, the set of all subsets of R_2.

and the riders in the second stage scenario) is minimized. More specifically,

$$\min_{D_1 \subset D} \left\{ cost_1(D_1, R_1) + \max_{S \in \mathcal{S}} cost_2(D \setminus D_1, S) \right\}.$$

The first stage decision is denoted D_1 and its cost is $cost_1(D_1, R_1)$. Similarly, $cost_2(D \setminus D_1, S)$ is the second stage cost for scenario S, and $\max\{cost_2(D \setminus D_1, S) \mid S \in \mathcal{S}\}$ is the worst-case cost over all possible scenarios. Let $|R_1| = m$, $|R_2| = n$. We denote the objective function for a feasible solution D_1 by

$$f(D_1) = cost_1(D_1, R_1) + \max_{S \in \mathcal{S}} cost_2(D \setminus D_1, S).$$

We assume that there are sufficiently many drivers to satisfy both first and second stage demand. Given an optimal first-stage solution D_1^*, we denote

$$OPT_1 = cost_1(D_1^*, R_1), \quad OPT_2 = \max\{cost_2(D \setminus D_1^*, S) \mid S \in \mathcal{S}\}, \quad OPT = OPT_1 + OPT_2.$$

As we mentioned earlier, we primarily focus on the setting where the first stage cost is the average weight of matching between D_1 and R_1, and the second stage cost is the bottleneck matching cost between $D \setminus D_1$ and S. The bottleneck matching problem is to find a maximum matching that minimizes the length of the longest edge. We refer to this variant as the *Two-Stage Robust Matching Bottleneck Problem* (**TSRMB**). We also consider several other cost variants and present results in Section 2.8. Formally, let M_1 be the minimum weight perfect matching between R_1 and D_1, and given a scenario S, let M_2^S be the bottleneck matching between the scenario S and the available drivers $D \setminus D_1$, then the cost functions for the TSRMB are:

$$cost_1(D_1, R_1) = \frac{1}{m} \sum_{(i,j) \in M_1} d(i,j), \quad \text{and} \quad cost_2(D \setminus D_1, S) = \max_{(i,j) \in M_2^S} d(i,j).$$

The difference between the first and second stage metric is motivated by the fact that the plateform has access to the current requests and can exactly compute the cost of matching these first stage requests. On the other hand, to ensure the robustness of the solution over the second stage uncertainty, we require for all second stage assignments to have low waiting times by accounting for the maximum wait time in every scenario. Note that we choose the first stage cost to be the average matching weight instead of the total weight for homogeneity reasons, so that first and second stage costs have comparable magnitudes.

Scenario model of uncertainty. Two common approaches to model uncertainty in robust optimization problems are to either explicitly enumerate all the realizations of the uncertain parameters or to specify them implicitly by a set of constraints over the uncertain parameters. In this chapter, we consider both models. In the *explicit model*, we are given a list of scenarios: $\mathcal{S} = \{S_1, \dots, S_p\}$. In the *implicit model*, we consider the setting where we are a given a universe of second stage riders, R_2 and any subset of size less than k can be a scenario. Therefore, the set of scenarios is $\mathcal{S} = \{S \subset R_2 \text{ s.t. } |S| \leq k\}$ for a given k. Note that the total number of possible scenarios is exponential in k; however, they are specified implicitly. This model is widely used in the Robust Optimization literature [123, 124] and is known as budget of uncertainty or cardinality constraint set.

2.1.2 Our results and techniques

Hardness. We show that TSRMB is NP-hard even for two scenarios and NP-hard to approximate within a factor better than 2 for three or more scenarios. For the case of implicit model of uncertainty, we show that even when the number of scenarios is small, specifically $k = 1$, the problem is NP-hard to approximate within a factor better than 2. Given these hardness results, we focus on designing approximation algorithms for the TSRMB problem.

A natural candidate to address two-stage problems is the greedy approach that minimizes only the first stage cost without considering the uncertainty in the second stage. However, we show that this myopic approach can be bad, namely $\Omega(m) \cdot OPT$.

Approximations algorithms. We first consider the case of a small number of explicit scenarios. This model is motivated by the desire to use historical data from past riders as our list of explicit scenarios. Our main result in this case is a constant approximation algorithm for TSRMB with two scenarios (Theorem 2.5.1). We further generalize the ideas of this algorithm to show a constant approximation for TSRMB with a fixed number of scenarios (Theorem 2.5.2). Our approximation does not depend on the number of first stage riders or the size of scenarios but scales with the number of scenarios. In particular, we have the following theorems.

Theorem 2.5.1 (restated). *There is an algorithm that yields a 5-approximation to the TSRMB problem with 2 scenarios.*

Theorem 2.5.2 (restated). *There is an algorithm that yields a $O(p^{1.59})$-approximation to the TSRMB with p explicit scenarios.*

The main idea in our algorithms is to reduce the TSRMB problem with multiple scenarios to an instance with a single *representative scenario* while losing only a small approximation factor. We then solve the single scenario instance (which can be done exactly in polynomial time) and recover a constant-factor approximation for our original problem. The challenge in constructing a single representative scenario is to find the right trade-off between effectively capturing the demand of all second stage riders and keeping the cost of this scenario close to the optimal cost of the original instance.

For the implicit model of uncertainty, the scenarios can be exponentially many in k, which makes even the evaluation of the total cost of a feasible solution challenging and not necessarily achievable in polynomial time. Our analysis depends on the imbalance between supply and demand. In fact, when the number of drivers is very large compared to riders, the problem is less interesting in practice. However, the problem becomes interesting when the supply and demand are comparable. In this case, drivers might need to be shared between different scenarios. This leads us to define the notion of surplus $\ell = |D| - |R_1| - k$, which is the maximum number of drivers that we can afford not to use in a solution. As a warm-up, we first show that if the surplus is equal to zero, (in this case all the drivers need to be used), using any scenario as a representative scenario and solving the single scenario instance gives a 3-approximation to TSRMB. The problem becomes significantly more challenging even with a small surplus. We show that under a reasonable assumption on the size of scenarios, there is a constant approximation to the TSRMB in the regime when the surplus ℓ is smaller than the demand k (Theorem 2.6.1). This result is quite involved and requires several different new ideas and techniques to overcome the exponential number of scenarios.

Theorem 2.6.1 (restated). *There is an algorithm that yields a* 17*-approximation to the TSRMB problem with implicit scenarios, when* $\ell < k$ *and* $k \leq \sqrt{\frac{n}{2}}$.

The algorithm in Theorem 2.6.1 finds a clustering of drivers and riders that yields a simplified instance of TSRMB which can be solved within a constant factor. We show that we can cluster the riders into a ball (riders close to each others) and a set of *outliers* (riders far from each others) and apply some of our ideas from the analysis of two scenario on these two sets. Finally, since the evaluation problem is challenging because of the exponentially many scenarios, our algorithm constructs a set of a polynomial number of proxy scenarios on which we can evaluate any feasible solution within a constant approximation. When the surplus ℓ is larger than the demand k, it becomes harder to group the set of potential outliers into one single scenario, and our algorithm might fail to compute a constant approximation. We also address the case of arbitrary surplus if each scenario has only a single rider ($k = 1$). While this case has only polynomially many scenarios of size 1 each, it is still NP-hard and we use different techniques to get a constant-factor approximation (Theorem 2.6.2). In particular, we establish a connection between our problem and q-supplier problem [55] which we use as a subroutine to design a constant approximation algorithm in this case.

Theorem 2.6.2 (restated). *There is an algorithm that yields a 15-approximation to the TSRMB with implicit scenarios in the case of* $k = 1$.

Extensions and variants. While the majority of the chapter focuses on the TSRBM problem, we also initiate the study of several other cost functions for two-stage matching problems both for adversarial and stochastic second stage scenarios. In particular, we consider the Two-Stage Stochastic Matching Bottleneck (TSSMB), where the first stage cost is the average weight of the matching, and the second stage is the expectation of the bottleneck matching cost over all scenarios (each scenarios can occur with a specified probability). We also consider the Two-Stage Robust Matching problem (TSRM), where the first and second stage costs correspond both to the total weight of the matchings. Finally, we consider the Two-Stage Robust Bottleneck Bottleneck problem (TSRBB), where the first and second stage costs both correspond to the bottleneck matching cost. We study the hardness of these variants, and make a first attempt to present approximation algorithms under specific settings.

Experimental study. We implement our algorithms and test them on real-life taxi data from the city of Shenzhen [125]. Our experimental results show that our two-scenarios algorithm improves significantly upon the greedy algorithm both in and out of sample. Furthermore, the experiments show that while the second stage bottleneck of our algorithm is significantly less than the bottleneck of the greedy algorithm, the total weight of the matchings provided by the two algorithms are roughly similar. This implies that our algorithm reduces the maximal second stage wait time, without adding to the overall average wait time. For example, we show for the instances we consider in our experiments, that our two-scenarios algorithm reduces the maximum wait time of the second-stage riders by an average of 30%. See Section 2.7 for more details.

Outline. The chapter is organized as follows. We review relevant literature in Section 2.2. In Section 2.3, we introduce some preliminary results on the hardness of TSRMB. We study the performance of the greedy approach and finally present a subroutine to solve the deterministic TSRMB with one scenario. In Section 2.5, we study TSRMB with explicit scenarios. In Section 2.6, we consider the case of implicit scenarios. Section 2.8 explores other variants of the two-stage robust matching problem with different cost functions. We present our numerical experiments on a set of real-life taxi data from the city of Shenzhen in Section 2.7.

2.2 Literature Review

Online bipartite matching. Finding a maximum cardinality bipartite matching is one of the classical problems in algorithmic graph theory and combinatorial optimization as it arises naturally in several applications such as resource allocation, scheduling, and online advertising. The online version of this problem has received a considerable amount of attention over the years (see survey [62]). In this setting, we are given a known set of *servers* while a set of *clients* arrive online and upon arrival, each client can be matched to a server irrevocably. The online matching problem was first studied by Karp *et al.* [60] in the adversarial model where the graph is unknown; when a client arrives it reveals its incident edges. Karp *et al.* [60] and Birnbaum and Mathieu [61] proved that the simple randomized RANKING algorithm achieves $(1 - 1/e)$ competitive ratio and this factor is the best possible performance. Since then, many online variants have been studied in great depth (see survey [62]). This includes problems like AdWords [63, 64, 65], vertex-weighted [66, 67], edge-weighted [68, 69], stochastic matching [70, 71, 72, 73], random vertex arrival [74, 75, 76, 77], and batch arrivals [78, 79, 80].

Online minimum weight matching. In the *online bipartite metric matching problem*, servers and clients correspond to points from a metric space. Upon arrival, each client must be matched to a server irrevocably, at a cost equal to their distance. The objective is to find the minimum weight maximum cardinality matching. For general metric spaces, Khullet *et al.* [81] and Kalyanasundaram and Pruhs [82] proved that there is a tight bound of $(2n-1)$ on the competitiveness factor of deterministic online algorithms, where n is the number of servers. In the random arrival model, a natural question is whether randomization could help obtain an exponential improvement for general metric spaces. Meyerson, *et al.* [83] and Bansal *et al.* [84] provided poly-logarithmic competitive randomized algorithms for the problem. Recently, Raghvendra [85] presented a $O(\log n)$-competitive algorithm in the random arrival model.

Two-stage stochastic combinatorial optimization. Within two-stage stochastic optimization, matching has been studied under various models and different objectives. Kong and Schaefer [126] introduce the stochastic two-stage maximum matching problem. They prove that the problem is NP-hard when the number of scenarios is an input of the problem and provide $1/2$-approximation algorithm. Escoffier *et al.* [97] further study this problem, strengthen the hardness results, and slightly improve the approximation ratio. Katriel *et al.* [96] study two stochastic minimum weight maximum matching problems in bipartite graphs. In their two variants, the uncertainty is respectively on the second stage cost of the edges and on the set of vertices to be matched. Feng and Niazadeh [79] study K-stage variants of vertex weighted bipartite b-matching and AdWords problems, where online vertices arrive in K batches. More recently, Feng *et al.* [127] initiate the study and present online competitive algorithms for vertex-weighted two-stage stochastic matching as well as two-stage joint matching and pricing.

Two-stage robust combinatorial optimization. Within two-stage robust optimization, matchings have not been studied extensively. Matuschke *et al.* proposed a two-stage robust model for minimum weight matching with recourse [98]. In the first stage, a perfect matching between $2n$ given nodes must be selected; in the second stage $2k$ new nodes are introduced. The goal is to produce α-competitive matchings at the end of both stages, and such that the number of edges removed from the first stage matching is at most βk. Our model for TSRMB is different in 3 main aspects: 1) In our model the second stage vertices come from an uncertainty set whereas in their model the only information given is the number of second stage vertices. 2) We do not allow any recourse and our first stage matching is irrevocable. 3) Our second stage cost is the bottleneck weight instead of the total weight. In general, a bottleneck optimization problem on a graph with edge costs is the problem of finding a subgraph of a certain kind that minimizes the maximum edge cost in the subgraph. The bottleneck objective contrasts with the more common objective of minimizing the sum of edge costs. Several Bottleneck problems have been considered, e.g. Shortest Path Problem [52, 53], Spanning Tree and Maximum Cardinality Matching [49], and TSP problems [54] (see [55] for a compilation of graph bottleneck problems).

2.3 Preliminaries

In this section, we study the hardness of approximation for TSRMB. We also examine the challenges with the natural greedy approach for solving TSRMB. We finally present a subroutine to solve the single scenario case that we will use later on in our general algorithms.

NP-hardness. We show that TSRMB is NP-hard under both the implicit and explicit models of uncertainty. In the explicit model, TSRMB is NP-hard even for two scenarios. Moreover, we show that it is NP-hard to approximate within a factor better than 2 even for three scenarios. In the explicit model with a polynomial number of scenarios, it is clear that the problem is in NP. However, in the implicit model, even though the problem can be described with a polynomial size input, it is not clear that we can compute the total cost function in polynomial time since there could be exponentially many scenarios. We also show that it is NP-hard to approximate TSRMB with an implicit model of uncertainty within a factor better than 2 even when $k = 1$. The exact statements of our hardness of approximation results are given in the following theorems. We provide below the proof of Theorem 2.3.1. The proof of Theorem 2.3.2 follows a similar approach and is deferred to Appendix A.1.

Theorem 2.3.1. *In the explicit model of uncertainty, TSRMB is NP-hard even when the number of scenarios is equal to 2. Furthermore, when the number of scenarios is ≥ 3, there is no $(2-\epsilon)$-approximation algorithm for any fixed $\epsilon > 0$, unless P = NP.*

Theorem 2.3.2. *In the implicit model of uncertainty, even when $k = 1$, there is no $(2-\epsilon)$-approximation algorithm for TSRMB for any fixed $\epsilon > 0$, unless $P = NP$.*

Proof of Theorem 2.3.1. Before proving the theorem, we start by presenting the 3-Dimensional Matching and Set Cover problems, that we use in our reduction to show Theorem 2.3.1. The problem is known to be strongly NP-hard [128, 129].

3-Dimensional Matching (3-DM): Given three sets U, V, and W of equal cardinality n, and a subset T of $U \times V \times W$, is there a subset M of T with $|M| = n$ such that whenever (u, v, w) and

(u', v', w') are distinct triples in M, $u \neq u'$, $v \neq v'$, and $w \neq w'$?

Consider an instance of the 3-Dimensional Matching Problem. We can use it to construct (in polynomial time) an instance of TSRMB with 2 scenarios as follows. We create two scenarios of size n: $S_1 = U$ and $S_2 = V$. We then set $D = T$, such that every driver corresponds to a triple in T. For every $w \in W$, let $d_T(w)$ be the number of sets in T that contain w. We create $d_T(w) - 1$ first stage riders, that are all copies of w. The total number of first stage riders is therefore $|R_1| = |T| - n$. We now specify the distances between drivers and riders in our graph:

- For $(w, e) \in R_1 \times D$, $d(w, e) = \begin{cases} 1 & \text{if } w \in e \\ 3 & \text{otherwise.} \end{cases}$
- For $(u, e) \in S_1 \cup S_2 \times D$, $d(u, e) = \begin{cases} 1 & \text{if } u \in e \\ 3 & \text{otherwise.} \end{cases}$
- For $u, v \in R_1 \cup S_1 \cup S_2, d(u, v) = \min_{e \in D} d(u, e) + d(v, e)$.
- For $e, f \in D, d(e, f) = \min_{u \in R_1 \cup S_1 \cup S_2} d(u, e) + d(u, f)$.

This choice of distances induces a metric graph. We claim that there exists a 3-dimensional matching if and only if there exists a solution to this TSRMB instance with total cost equal to 2. Suppose that $M = \{e_1, \ldots, e_n\} \subset T$ is a 3-Dimensional matching. Let $e_1, \ldots, e_n$ be the drivers that correspond to M in the TSRMB instance. We show that by using $D_1 = D \setminus \{e_1, \ldots, e_n\}$ as a first stage decision, we ensure that the total cost for the TSRMB instance is equal to 2. For any rider u in scenario S_1, by definition of M, there exits a unique edge $e_i \in M$ that covers u. The corresponding driver $e_i \notin D_1$ can be matched to u with a distance equal to 1. Furthermore, e_i cannot be matched to any other rider in S_1 with a cost less than 1. Similarly, for any rider v in scenario S_2, since there exits a unique edge $e_j \in M$ that covers v, the corresponding driver can be matched to v with a cost of 1. The second stage cost is therefore equal to 1. As for the first stage cost, we know by definition of M, that every element $w \in W$ is covered exactly once. Therefore, for every $w \in W$, there exists $d_T(w) - 1$ edges that contain w in $T \setminus M$. This means that every 1st stage rider can be matched to a driver in D_1 with a cost equal to 1. Hence the total cost of this two-stage matching is equal to 2.

Suppose now that there exists a solution to the TSRMB instance with a cost equal to 2. This means that the first and second stage costs are both equal to 1. Let $M = \{e_1, \ldots, e_n\}$ be the set of drivers used in the second stage of this solution. We show that M is a 3-dimensional matching. Let $e_i = (u, v, w)$ and $e_j = (u', v', w')$ be distinct triples in M. Since the second stage cost is equal to 1, the driver e_i (resp. e_j) must be matched to u (resp. u') in S_1. Since we have exactly n second stage drivers and n riders in S_1, this means that e_i and e_j have to be matched to different second stage riders in S_1. Therefore we get $u' \neq u$. Similarly we see that $v' \neq v$. Assume now that $w = w'$, this means that the TSRMB solution has used two drivers (triples) e_i and e_j that contain w in the second stage. It is therefore impossible to match all the $d_T(w) - 1$ copies of w in the first stage with a cost equal to 1. Therefore $w \neq w'$. The above construction can be performed in polynomial time of the 3-DM input, and therefore shows that TSRMB with two scenarios is NP-hard.

Now, to show that TSRMB is hard to approximate within a factor better than 2, we consider three scenarios. Consider an instance of 3-DM. We can use it to construct an instance of TSRMB with 3 scenarios as follows. We create 3 scenarios this time, each of size n: $S_1 = U$, $S_2 = V$ and $S_3 = W$. We set $D = T$, and create $|R_1| = |T| - n$ first stage riders. We set the distances as follows:

- For $(w, e) \in R_1 \times D$, $d(w, e) = 1$.
- For $(u, e) \in S_1 \cup S_2 \cup S_3 \times D$, $d(u, e) = \begin{cases} 1 & \text{if } u \in e \\ 3 & \text{otherwise.} \end{cases}$
- For $u, v \in R_1 \cup S_1 \cup S_2 \cup S_3$, $d(u, v) = \min_{e \in D} d(u, e) + d(v, e)$.
- For $e, f \in D, d(e, f) = \min_{u \in R_1 \cup S_1 \cup S_2 \cup S_3} d(u, e) + d(u, f)$.

This choice of distances induces a metric graph. Similarly to the proof of 2 scenarios, we can show that there exists a 3-dimensional matching if and only if there exists a TSRMB solution with cost equal to 2. Furthermore, any solution for this TSRMB instance have either total cost of 2 or 4 (the first stage cost is always equal to 1). We show that if a $(2-\epsilon)$-approximation (for some $\epsilon > 0$) to the TSRMB exists then 3-Dimensional Matching is decidable. We know that this instance of TSRMB has a solution with total cost equal to 2 if and only if there is a 3-dimensional matching. Furthermore, if there is no 3-dimensional matching, the cost of the optimal solution to TSRMB must be 4. Therefore, if an algorithm guarantees a ratio of $(2-\epsilon)$ and a 3-dimensional matching exists, the algorithm delivers a solution with total cost equal to 2. If there is no 3-dimensional matching, then the solution produced by the algorithm has a total cost of 4. This concludes that if a $(2-\epsilon)$-approximation (for some $\epsilon > 0$) to the TSRMB exists then 3-Dimensional Matching is decidable. □

Greedy Approach. A natural greedy approach is to choose the optimal matching for the first stage riders R_1 without considering the uncertainty in second stage in any way. We show via a counterexample that this greedy approach could lead to a bad solution for TSRMB with a total cost that scales linearly with m (cardinality of R_1) while OPT is a constant, even when there is only one scenario.

Counterexample. Consider the line example depicted in Figure 2.1, where we have m first stage riders and $m+1$ drivers that alternate on a line with distances 1 and $1-\epsilon$. There is only one second stage rider at the right endpoint of the line. A greedy matching would minimize the first stage cost by matching the first stage riders using the dashed edges, with an average weight of $1-\epsilon$. When the second stage scenario is revealed, the rider can only be matched with the farthest driver for a cost of $1+(2-\epsilon)m$. Therefore the total cost of the greedy approach is $(2-\epsilon)(m+1)$, while the optimal cost is clearly equal to 2. This example shows that the cost of the greedy algorithm for TSRMB could be far away from the optimal cost with an approximation ratio that scales with the dimension of the problem. The same observation generalizes to any number of scenarios by simply duplicating the second stage rider. Therefore any attempt to have a good approximation to the TSRMB needs to consider the second stage riders. In particular, we have the following lemma.

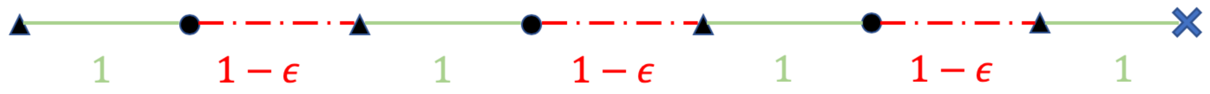

Figure 2.1: Example instance on the line. Riders in first stage are depicted by black dots and drivers are indicated as black triangles. The second stage rider is depicted as a blue cross. First and second stage optimum are depicted by solid green edges. Matching the first stage riders greedily leads to a significant increase in the second stage cost.

Lemma 2.3.1. *The cost of the Greedy algorithm can be* $\Omega(m) \cdot OPT$.

2.4 Single Scenario.

The *deterministic* version of the TSRMB problem, i.e., when there is only a single scenario in the second stage, can be solved exactly in polynomial time. This is a simple preliminary result which we need for the general case. Let S denote a single second stage scenario. The instance (R_1, S, D) of TSRMB is then simply given by

$$\min_{D_1 \subset D} \left\{ cost_1(D_1, R_1) + cost_2(D \setminus D_1, S) \right\}.$$

Since the second stage problem is a bottleneck problem, the value of the optimal second stage cost w is one of the edge weights between D and S. We iterate over all possible values of w (at most $|S| \cdot |D|$ values), delete all edges between R_2 and D with weights strictly higher than w and set the weight of the remaining edges between S and D to zero. This reduces the problem to finding a minimum weight maximum cardinality matching. Below, are presented the details of our algorithm. We refer to it as *TSRMB-1-Scenario* (or Algorithm 1) in the rest of the chapter.

We define the bottleneck graph of w to be $BOTTLENECKG(w) = (R_1 \cup S \cup D, E_1 \cup E_2)$ where $E_2 = \{(i,j) \in D \times S,\ d(i,j) \le w\}$ and $E_1 = \{(i,j) \in D \times R_1\}$. Furthermore, we assume that there are q edges $\{e_1, \ldots, e_q\}$ between S and D with weights $w_1 \le w_2 \le \ldots \le w_q$.

Note that the arg min in the last step of Algorithm 1 is only taken over values of i for which there was no certificate of failure.

Algorithm 1 TSRMB-1-Scenario(R_1, S, D)

```
Input: First stage riders $R_1$, scenario $S$ and drivers $D$.
Output: First stage decision $D_1$.
1: for $i \in \{1, \ldots, q\}$ do
2:     $G_i := BOTTLENECKG(w_i)$.
3:     Set all weights between $D$ and $S$ in $G_i$ to be 0.
4:     $M_i :=$ minimum weight maximum cardinality matching on $G_i$.
5:     if $R_1 \cup S$ is not completely matched in $M_i$ then
6:         output certificate of failure.
7:     else
8:         $D_1^i :=$ first stage drivers in $M_i$.
9: return $D_1 = \arg\min_{D_1^i : 1 \le i \le q} \left\{ cost_1(D_1^i, R_1) + cost_2(D \setminus D_1^i, S) \right\}$.
```

Lemma 2.4.1. *TSRMB-1-Scenario (Algorithm 1) provides an exact solution to TSRMB with a single scenario.*

Proof. Let OPT_1 and OPT_2 be the first and second stage cost of an optimal solution, and $i \in \{1, \ldots, q\}$ such that $w_i = OPT_2$. In this case, G_i contains all the edges of this optimal solution. By setting all the edges in E_2 to 0, we are able to compute a minimum weight maximum cardinality matching between $R_1 \cup S$ and D that matches both R_1 and S and minimizes the weight of the edges matching R_1. The first stage cost of this matching is less than OPT_1, the second stage cost is clearly less than OPT_2 because we only allowed edges with weight less than OPT_2 in G_i. $\square$

We also observe that we can use binary search in Algorithm 1 to iterate over the edge weights. For an iteration i, a failure to find a minimum weight maximum cardinality matching on G_i that matches both R_1 and S implies that we need to try an edge weight higher than w_i. On the other hand, if M_i matches R_1 and S such that D_1^i gives a smaller total cost, then the optimal bottleneck value is lower than w_i.

2.5 Explicit Scenarios

In this section, we consider TSRMB under the explicit model of uncertainty where we have an explicit list of scenarios for the second-stage and we optimize over the worst case scenario realization. We first present a constant factor approximation for TSRMB for the case of two scenarios. We then extend our result to the case of any fixed number of scenarios. However, the approximation factor scales with the number of scenarios p as $O(p^{1.59})$. The idea of our algorithm is to reduce the instance of TSRMB with p scenarios to an instance with only a single representative scenario by losing a small factor and then use Algorithm 1 to solve the single scenario instance. To illustrate the core ideas of our algorithm, we focus on the case of two scenarios first and then extend it to a constant number of scenarios.

2.5.1 Two scenarios

Consider two scenarios $\mathcal{S} = \{S_1, S_2\}$. First, we can assume without loss of generality that we know the exact value of OPT_2 which corresponds to one of the edges connecting second stage riders R_2 to drivers D (we can iterate over all the weights of second stage edges). We construct a representative scenario that serves as a proxy for S_1 and S_2 as follows. In the second stage, if a pair of riders $i \in S_1$ and $j \in S_2$ are served by the same driver in the optimal solution, then they should be close to each other. Therefore, we can consider a single representative rider for each such pair. While it is not easy to guess all such pairs, we can approximately compute the representative riders by solving a maximum matching on $S_1 \cup S_2$ with edges less than $2OPT_2$. More formally, let G_I be the induced bipartite subgraph of G on $S_1 \cup S_2$ containing only edges between S_1 and S_2 with weight less than or equal to $2OPT_2$. We compute a maximum cardinality matching M between S_1 and S_2 in G_I, and construct a representative scenario containing S_1 as well as the unmatched riders of S_2. We solve the single scenario problem on this representative scenario using Algorithm 1 and return its optimal first stage solution. We show in Theorem 2.5.1 that this solution leads to 5-approximation for our problem. Our algorithm is described below.

Theorem 2.5.1. *Algorithm 2 yields a solution with total cost less than $OPT_1 + 5OPT_2$ for TSRMB with 2 scenarios.*

Recall that OPT_1 and OPT_2 are respectively the first-stage and second-stage cost of an optimal solution for our TSRMB problem with two scenarios. The proof of Theorem 2.5.1 relies on the

Algorithm 2 Two explicit scenarios.

```
Input: First stage riders $R_1$, two scenarios $S_1$ and $S_2$, drivers $D$ and value of $OPT_2$.
Output: First stage decision $D_1$.
1: Let $G_I$ be the induced subgraph of $G$ on $S_1 \cup S_2$ with only the edges between $S_1$ and $S_2$ of
   weights less than $2OPT_2$ .
2: Set $M$ := maximum cardinality matching between $S_1$ and $S_2$ in $G_I$.
3: Set $S_2^{Match} := \{r \in S_2 \mid \exists\, s \in S_1 \text{ s.t } (s,r) \in M\}$ and $S_2^{Unmatch} = S_2 \setminus S_2^{Match}$.
4: return $D_1$ := TSRMB-1-Scenario($R_1, S_1 \cup S_2^{Unmatch}, D$).
```

following structural lemma where we show that the set D_1 returned by Algorithm 2 yields a total cost at most $(OPT_1 + 3OPT_2)$ when evaluated only on the single representative scenario $S_1 \cup S_2^{Unmatch}$.

Lemma 2.5.1. *Let D_1 be the set of first stage drivers returned by Algorithm 2. Then,*

$$cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_1 \cup S_2^{Unmatch}) \leq OPT_1 + 3OPT_2.$$

Proof. To prove the lemma, it is sufficient to show the existence of a matching M_a between $R_1 \cup S_1 \cup S_2^{Unmatch}$ and D with a total cost less than $OPT_1 + 3OPT_2$. This would imply that the optimal solution D_1 of TSRMB-1-Scenario($R_1, S_1 \cup S_2^{Unmatch}, D$) has a total cost less than $OPT_1 + 3OPT_2$ and concludes the proof of the lemma. We show the existence of M_a by construction.

- **Step 1.** We first match R_1 with their mates in the optimal solution of TSRMB. Hence, the first stage cost of our constructed matching M_a is OPT_1.

- **Step 2.** Now, we focus on $S_2^{Unmatch}$. Let $S_2^{Unmatch} = S_{12} \cup S_{22}$ be a partition of $S_2^{Unmatch}$ where S_{12} contains riders with a distance less than $2OPT_2$ from S_1 and S_{22} contains riders with a distance strictly bigger than $2OPT_2$ from S_1, where the distance from a set is the minimum distance to any element of the set. A rider in S_{22} cannot share any driver with a rider from S_1 in the optimal solution of TSRMB, because otherwise, the distance between these riders will be less than $2OPT_2$ by using the triangle inequality. Therefore we can match S_{22} to their mates in the optimal solution and add them to M_a, without using the optimal drivers of S_1. We pay less than OPT_2 for matching S_{22}.

- **Step 3.** We still need to simultaneously match riders in S_1 and S_{12} to finish the construction of M_a. Notice that some riders in S_{12} might share their optimal drivers with riders in S_1. We can assume without loss of generality that all riders in S_{12} share their optimal drivers with S_1 (otherwise we can match them to their optimal drivers without affecting S_1). Denote $S_{12} = \{r_1, \ldots, r_q\}$ and $S_1 = \{s_1, \ldots, s_k\}$. For each $i \in [q]$ let's say $s_i \in S_1$ is the rider that shares its optimal driver with r_i. We show that $q \leq |M|$. In fact, every rider in S_{12} shares its optimal driver with a different rider in S_1, and is therefore within a distance $2OPT_2$ from S_1 by the triangle inequality. But since S_{12} is not covered by the maximum cardinality matching M, this implies by the maximality of M that there are q other riders from S_2^{Match} that are covered by M. Hence $q \leq |M|$. Finally, let $\{t_1, \ldots, t_q\} \subset S_2^{Match}$ be the mates of $\{s_1, \ldots, s_q\}$ in M, i.e., $(s_i, t_i) \in M$ for all $i \in [q]$. Recall that $d(s_i, t_i) \leq 2OPT_2$ for all $i \in [q]$. In what follows, we describe how to match S_{12} and S_1:
  - For $i \in [q]$, we match r_i to its optimal driver and s_i to the optimal driver of t_i. This is possible because the optimal driver of t_i cannot be the same as the optimal driver of r_i since both r_i and t_i are part of the same scenario S_2. Therefore, we pay a cost OPT_2 for the riders r_i and a cost $3OPT_2$ (follows from the triangle inequality) for the riders s_i where $i \in [q]$.
  - We still need to match $\{s_{q+1}, \ldots, s_k\}$. Consider a rider s_j with $j \in \{q+1, \ldots, k\}$. If the optimal driver of s_j is not shared with any $t_i \in \{t_1, \ldots, t_q\}$, then this optimal driver is still available and can be matched to s_j with a cost less than OPT_2. If the optimal driver of s_j is shared with some $t_i \in \{t_1, \ldots t_q\}$, then s_j is also covered by M. Otherwise M can be augmented by deleting (s_i, t_i) and adding (r_i, s_i) and (s_j, t_i). Therefore s_j is covered by M and has a mate $\tilde{t}_j \in S_2^{Match} \setminus \{t_1, \ldots, t_q\}$. Furthermore, the driver assigned to $\tilde{t}_j$ is still available. We can then match s_j to the optimal driver of $\tilde{t}_j$. Similarly if the optimal driver of some $s_{j'} \in \{s_{q+1}, \ldots, s_k\} \setminus \{s_j\}$ is shared with $\tilde{t}_j$, then $s_{j'}$ is covered by M. Otherwise $(r_i, s_i, t_i, s_j, \tilde{t}_j, s_{j'})$ is an augmenting path in M. Therefore $s_{j'}$ has a mate in M and we can match $s_{j'}$ to the optimal driver of its

mate. We keep extending these augmenting paths until all the riders in $\{s_{q+1}, \dots, s_k\}$ are matched. Furthermore, the augmenting paths $(r_i, s_i, t_i, s_j, \tilde{t}_j, s_{j'} \dots)$ starting from two different riders $r_i \in S_{12}$ are vertex disjoint. This ensures that every driver is used at most once. Again, by the triangle inequality, the edges that match $\{s_{q+1}, \dots, s_k\}$ in our solution have weights less then $3OPT_2$.

Putting it all together, we have constructed a matching M_a where the first stage cost is exactly OPT_1 and the second-stage cost is less than $3OPT_2$ since the edges used for matching $S_1 \cup S_2^{Unmatch}$ in M_a have a weight less than $3OPT_2$. Therefore, the total cost of M_a is less than $OPT_1 + 3OPT_2$.

□

Proof of Theorem 2.5.1. Let D_1 be the set of drivers returned by Algorithm 2. Lemma 2.5.1 implies that

$$cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_1) \leq \; OPT_1 + 3OPT_2 \tag{2.1}$$

and

$$cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_2^{Unmatch}) \leq \; OPT_1 + 3OPT_2.$$

We have $S_2 = S_2^{Match} \cup S_2^{Unmatch}$. If the scenario S_2 is realized, we use the drivers that were assigned to S_1 in the matching constructed in Lemma 2.5.1 to match S_2^{Match}. This is possible with edges of weights less than $cost_2(D \setminus D_1, S_1) + 2OPT_2$ because by definition S_2^{Match} are connected to S_1 within edges of weight less than $2OPT_2$. Therefore,

$$cost_2(D \setminus D_1, S_2) \leq \max\left\{cost_2(D \setminus D_1, S_2^{Unmatch}),\; cost_2(D \setminus D_1, S_1) + 2OPT_2\right\}$$

and therefore

$$cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_2) \leq OPT_1 + 5OPT_2. \tag{2.2}$$

From (2.1) and (2.2), we conclude that

$$cost_1(D_1, R_1) + \max_{S \in \{S_1, S_2\}} cost_2(D \setminus D_1, S) \leq OPT_1 + 5OPT_2.$$

□

2.5.2 Constant number of scenarios

We now consider the case of an explicit list of p scenarios, i.e., $\mathcal{S} = \{S_1, S_2, \ldots, S_p\}$. Building upon the ideas from Algorithm 2, we present $O(p^{1.59})$-approximation to TSRMB with p scenarios. The idea of our algorithm is to construct the representative scenario recursively by processing pairs of "scenarios" at each step. Hence, we need $O(\log_2 p)$ iterations to reduce the problem to an instance of a single scenario. At each iteration, we show that we only lose a multiplicative factor of 3 so that the final approximation ratio is $O(3^{\log_2 p}) = O(p^{1.59})$. We present details in Algorithm 3. Theorem 2.5.2 states the theoretical guarantee. We note that the approximation guarantee of our algorithm grows in a sub-quadratic manner with the number of scenarios p and it is an open question if there exists an algorithm for TSRMB with an approximation guarantee that does not depend on the number of scenarios.

Algorithm 3 p explicit scenarios.

```
Input: First-stage riders $R_1$, scenarios $\{S_1, S_2, \ldots, S_p\}$, drivers $D$ and value of $OPT_2$.
Output: First stage decision $D_1$.
1: Initialize $\hat{S}_j := S_j$ for $j = 1, \ldots, p$.
2: for $i = 1, \ldots, \log_2 p$ do
3:     for $j = 1, 2, \ldots, \frac{p}{2^i}$ do
4:         $\sigma(j) = j + \frac{p}{2^i}$
5:         $M_j$ := maximum cardinality matching between $\hat{S}_j$ and $\hat{S}_{\sigma(j)}$ with edges of weight less
           than $2 \cdot 3^{i-1} \cdot OPT_2$.
6:         $\hat{S}^{Match}_{\sigma(j)} := \{r \in \hat{S}_{\sigma(j)} \mid \exists\, s \in \hat{S}_j$ s.t $(s, r) \in M_j\}$.
7:         $\hat{S}^{Unmatch}_{\sigma(j)} := \hat{S}_{\sigma(j)} \setminus \hat{S}^{Match}_{\sigma(j)}$
8:         $\hat{S}_j = \hat{S}_j \cup \hat{S}^{Unmatch}_{\sigma(j)}$.
9: return $D_1$ := TSRMB-1-Scenario($R_1, \hat{S}_1, D$).
```

Theorem 2.5.2. *Algorithm 3 yields a solution with total cost of $O(p^{1.59}) \cdot OPT$ for TSRMB with an explicit list of p scenarios.*

Proof of Theorem 2.5.2. The algorithm reduces the number of considered "scenarios" by half in every iteration, until only one scenario remains. In iteration i, we have $\frac{p}{2^{i-1}}$ scenarios that we aggregate in $\frac{p}{2^i}$ pairs, namely $(\hat{S}_j\ ,\ \hat{S}_{\sigma(j)})$ for $j \in \{1, 2, \ldots, \frac{p}{2^i}\}$. For each pair, we construct a single representative scenario which plays the role of the new $\hat{S}_j$ at the start of the next iteration $i + 1$.

Claim 2.5.1. *There exists a first stage decision D_1^*, such that at every iteration $i \in \{1, \ldots, \log_2 p\}$, we have for all $j \in \{1, 2, \ldots, \frac{p}{2^i}\}$:*

(1) R_1 can be matched to D_1^* with a first stage cost of OPT_1.

(2) $\hat{S}_j \cup \hat{S}_{\sigma(j)}^{Unmatch}$ can be matched to $D \setminus D_1^*$ with a second stage cost less than $3^i \cdot OPT_2$.

(3) There exists a matching between $\hat{S}_{\sigma(j)}^{Match}$ and $\hat{S}_j$ with all edge weights less than $2 \cdot 3^{i-1} \cdot OPT_2$.

Proof of Claim 2.5.1. Statement (3) follows from the definition of $\hat{S}_{\sigma(j)}^{Match}$ in Algorithm 3. Let's show (1) and (2) by induction over i.

- **Initialization:** for $i = 1$, let's take any two scenarios $\hat{S}_j = S_j$ and $\hat{S}_{\sigma(j)} = S_{\sigma(j)}$. We know that these two scenarios can be matched to drivers of the optimal solution in the original problem with a cost less than OPT_2. In the proof of Lemma 2.5.1, we show that if we use the optimal first stage decision D_1^* of the original problem, then we can match $\hat{S}_j$ and $\hat{S}_{\sigma(j)}^{Unmatch}$ simultaneously to $D \setminus D_1^*$ with a cost less than $3OPT_2$.

- **Maintenance.** Assume the claim is true for all values less than $i \leq \log_2 p - 1$. We show it is true for $i + 1$. Since the claim is true for iteration i, we know that at the start of iteration $i + 1$, for $j \in \{1, \ldots, \frac{p}{2^i}\}$, $\hat{S}_j$ can be matched to $D \setminus D_1^*$ with a cost less than $3^i \cdot OPT_2$. We can therefore consider a new TSRMB problem with $\frac{p}{2^i}$ scenarios, where using D_1^* as a first stage decision ensures a second stage optimal value less than $\widehat{OPT}_2 = 3^i \cdot OPT_2$. By the proof of

Lemma 2.5.1, and by using D_1^* as a first stage decision in this problem, we ensure that for $j \in \{1, \dots, \frac{p}{2^{i+1}}\}$, $\hat{S}_j$ and $\hat{S}_{\sigma(j)}^{Unmatch}$ can be simultaneously matched to $D \setminus D_1^*$ with a cost less than $3\widehat{OPT}_2 = 3^{i+1} \cdot OPT_2$.

□

From Claim 2.5.1, we have in the last iteration $i = \log_2 p$,

- R_1 can be matched to D_1^* with a first stage cost of OPT_1.
- $\hat{S}_1$ can be matched to $D \setminus D_1^*$ with a second stage cost less than $3^{\log_2 p} \cdot OPT_2$.

Computing the single scenario solution for $\hat{S}_1$ will therefore yield a first stage decision D_1 that gives a total cost less than $OPT_1 + 3^{\log_2 p} \cdot OPT_2$ when the second stage is evaluated on the scenario $\hat{S}_1$. We now bound the cost of D_1 on the original scenarios $\{S_1, \dots, S_p\}$. Consider a scenario $S \in \{S_1, \dots, S_p\}$. The riders in $S \cap \hat{S}_1$ can be matched to some drivers in $D \setminus D_1$ with a cost less than $OPT_1 + 3^{\log_2 p} \cdot OPT_2$. As for other riders of $S \setminus \hat{S}_1$, they are not part of $\hat{S}_1$ because they have been matched and deleted at some iteration $i < \log_2 p$. Consider riders r in $S \setminus \hat{S}_1$ that were matched and deleted from a representative scenario at some iteration, then by statement (3) in Claim 2.5.1, each r can be connected to a different rider in $\hat{S}_1 \setminus (\hat{S}_1 \cap S)$ within a path of length at most

$$\sum_{t=1}^{\log_2 p} 2 \cdot 3^{t-1} \cdot OPT_2 = (3^{\log_2 p} - 1) \cdot OPT_2.$$

We know that R_1 and $\hat{S}_1$ can be matched respectively to D_1 and $D \setminus D_1$ with a total cost less than $OPT_1 + 3^{\log_2 p} \cdot OPT_2$. Therefore, we can match R_1 and S respectively to D_1 and $D \setminus D_1$ with a total cost less than

$$OPT_1 + 3^{\log_2 p} \cdot OPT_2 + (3^{\log_2 p} - 1) \cdot OPT_2 = O(3^{\log_2 p}) \cdot OPT = O(p^{\ln 3/\ln 2}) \cdot OPT = O(p^{1.59}) \cdot OPT.$$

Therefore, the worst-case total cost of the solution returned by Algorithm 3 is $O(p^{1.59}) \cdot OPT$. □

2.6 Implicit Scenarios

In this section, we consider TSRMB under the implicit description of scenarios $\mathcal{S} = \{S \subset R_2 \text{ s.t. } |S| \leq k\}$. This uncertainty model is widely used, however, it poses a challenge because the number of scenarios can be exponential. Therefore, even computing the worst case second stage cost, for a given first stage solution, might not be possible in polynomial time and we can no longer assume that we can guess OPT_2. We can show that the worst case occurs at scenarios of size exactly k, and hence we will focus on the implicit model of uncertainty where $S \subset R_2$ and $|S| = k$. Our analysis for this model of uncertainty will depend on the balance between supply (drivers) and demand (riders). In particular, we introduce the notion of surplus l defined as the excess in the number of available drivers for matching first-stage riders and a second-stage scenario:

$$\ell = |D| - |R_1| - k.$$

As a warm-up, we start by studying the case of no surplus ($\ell = 0$). Then, we address the more general case with a small surplus of drivers. We end this section by examining the case of scenarios with a single rider ($k = 1$) and arbitrary surplus.

2.6.1 Warm-up: no surplus

When the number of drivers is equal to the number of first stage riders plus the size of scenarios (i.e., $\ell = 0$), we show a 3-approximation to TSRMB by simply solving a single scenario TSRMB with any of the scenarios. In fact, because there is no surplus, all scenarios are matched to the same drivers in the optimal solution. Hence, between any two scenarios, there exists a matching where all edge weights are less than $2OPT_2$. So by solving TSRMB with only one of these scenarios, we can recover a solution and bound the cost of the other scenarios within $OPT_1 + 3OPT_2$ using the triangular inequality. For completeness, we present below our algorithm for the no surplus case. We show its worst-case guarantee in Lemma 2.6.1.

Algorithm 4 Implicit scenarios with no surplus.

```
Input: First stage riders $R_1$, second stage riders $R_2$, size $k$ and drivers $D$.
Output: First stage decision $D_1$.
1: $S_1$ := a second stage scenario of size $k$.
2: $D_1$ := TSRMB-1-Scenario($R_1, S_1, D$).
3: return $D_1$.
```

Lemma 2.6.1. *Algorithm 4 yields a solution with total cost less than $OPT_1 + 3OPT_2$ for TSRMB with implicit scenarios and no surplus.*

Proof. Let OPT_1 and OPT_2 be the first and second stage cost of the optimal solution. Let $f(D_1)$ be the total cost of the solution returned by the algorithm. We claim that $f(D_1) \leq OPT_1 + 3OPT_2$. It is clear that $cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_1) \leq OPT_1 + OPT_2$. Let $S \in \mathcal{S}$ be another scenario. Because $|D| = |R_1| + k$, the optimal solution uses exactly the same k drivers to match all the second stage scenarios. This implies that we can use the triangular inequality to find a matching between S and S_1 of bottleneck cost less than $2OPT_2$. Hence for any scenario S,

$$cost_1(D_1, R_1) + cost_2(D \setminus D_1, S) \leq cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_1) + 2OPT_2 \leq OPT_1 + 3OPT_2.$$

□

We note that when the surplus is strictly greater than 0, Algorithm 4 no longer yields a constant approximation and its worst case performance can be as bad as $\Omega(m)$. In particular, consider the example in Figure 2.2, with $k = 1$ and two second stage riders. The single scenario solution for S_1 uses only dashed edges in the first stage, and therefore uses the optimal second stage driver of S_2. Hence, if S_2 is realized, the cost of matching S_2 to the closest available driver is $\Omega(m)$. By symmetry, solving the single scenario problem for S_2 yields a $\Omega(m)$ bottleneck cost for S_1.

2.6.2 Small surplus

As observed in the example of Figure 2.2, the TSRMB problem becomes challenging even with a unit surplus of drivers and Algorithm 4 could be arbitrarily bad. Motivated by this, we focus

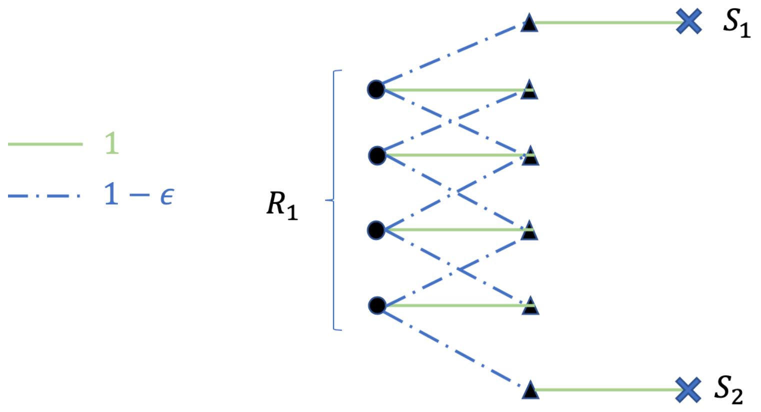

Figure 2.2: Example instance with a surplus of one. Riders in first stage are depicted by black dots and drivers are indicated as black triangles. The two second stage riders are depicted as blue crosses. First and second stage optimum are depicted as solid green edges. $\mathcal{S} = \{S_1, S_2\}$ and $k = 1$.

on the case of a small surplus ℓ. In particular, we assume that $\ell < k$, i.e., the excess in the total available drivers is smaller than the size of any scenario. We present a constant approximation algorithm in this regime for the implicit model of uncertainty where the size of scenarios is relatively small with respect to the size of the universe ($k = O(\sqrt{n})$). This technical assumption is needed for our analysis but it is not too restrictive and still captures the regime where the number of scenarios can be exponential. Our algorithm attempts to cluster the second stage riders in different groups (a *ball* and a set of *outliers*) in order to reduce the number of possible worst-case configurations. We then solve a sequence of instances with representative riders from each group. In what follows, we present our construction for these groups of riders.

Our construction. First, we show that many of the riders are contained in a ball with radius $3OPT_2$. The center of this ball δ can be found by enumerating over all drivers and selecting the one with the least maximum distance to its closest k second-stage riders, i.e.,

$$\delta = \operatorname*{arg\,min}_{\delta' \in D} \max_{r \in R_k(\delta')} d(\delta', r) \tag{2.3}$$

where $R_k(\delta')$ is the set of the k closest second stage riders to δ'. Formally, we have the following lemma.

Lemma 2.6.2. *Suppose $k \leq \sqrt{\frac{n}{2}}$ and $\ell < k$ and let δ be the driver given by* (2.3)*. Then, the ball $\mathcal{B}$*

centered at δ with radius $3OPT_2$ contains at least $n-\ell$ second stage riders. Moreover, the distance between any of these riders and any rider in $R_k(\delta)$ is less than $4OPT_2$.

Proof of Lemma 2.6.2. Let δ be the driver given by (2.3). We claim that the k closest riders to δ are all within a distance less than OPT_2 from δ. Consider D_2^* to be the $k+\ell$ drivers left for the second stage in the optimal solution. Every driver in D_2^* can be matched to a set of different second stage riders over different scenarios. Let us rank the drivers in D_2^* according to how many different second stage riders they are matched to over all scenarios, in descending order. Formally, let $D_2^* = \{\delta_1, \delta_2, \dots, \delta_{k+\ell}\}$ and let $R^*(\delta_i)$ be the second stage riders that are matched to δ_i in the optimal solution in some scenario, such that

$$|R^*(\delta_1)| \geq \ldots \geq |R^*(\delta_{k+\ell})|.$$

We claim that $|R^*(\delta_1)| \geq k$. In fact, we have $\sum_{i=1}^{k+\ell} |R^*(\delta_i)| \geq n$ because every second stage rider is matched to at least one driver in some scenario. Therefore

$$|R^*(\delta_1)| \geq \frac{n}{k+\ell} \geq \frac{n}{2k} \geq k.$$

We know that all the second stage riders in $R^*(\delta_1)$ are within a distance less than OPT_2 from δ_1. Therefore $\max_{r \in R_k(\delta_1)} d(\delta_1, r) \leq OPT_2$. But we know that by definition of δ,

$$\max_{r \in R_k(\delta)} d(\delta, r) \leq \max_{r \in R_k(\delta_1)} d(\delta_1, r) \leq OPT_2$$

This proves that the k closest second stage riders to δ are within a distance less than OPT_2. Let $R(\delta)$ be the set of all second stage riders that are within a distance less than OPT_2 from δ. Recall that $R_k(\delta)$ is the set of the k closest second stage riders to δ. In the optimal solution, the scenario $R_k(\delta)$ is matched to a set of at least new $k-1$ drivers $\{\delta_{i_1}, \dots \delta_{i_{k-1}}\} \subset D_2^* \setminus \{\delta\}$. We show a lower bound on the size of $R(\delta)$ and the number of riders matched to $\{\delta_{i_1}, \dots \delta_{i_{k-1}}\}$ over all scenarios in the optimal solution.

Claim 2.6.1. $\left|R(\delta) \bigcup_{j=1}^{k-1} R^*(\delta_{i_j})\right| \geq n - \ell$

Proof. Suppose the opposite, suppose that at least $\ell + 1$ riders from R_2 are not in the union. Let F be the set of these $\ell + 1$ riders. Since $\ell + 1 \leq k$, we can construct a scenario S that includes F. In the optimal solution, and in particular, in the second stage matching of S, at least one rider from F needs to be matched to a driver from $\{\delta, \delta_{i_1}, \dots \delta_{i_{k-1}}\}$. Otherwise there are only ℓ second stage drivers left to match all of F. Therefore there exists $r \in F$ such that either $r \in R(\delta)$ or there exists $j \in \{1, \dots, k-1\}$ such that $r \in R^*(\delta_{i_j})$. This shows that $r \in R(\delta) \bigcup_{j=1}^{k-1} R^*(\delta_{i_j})$, which is a contradiction. Therefore, at most ℓ second stage riders are not in the union. □

Claim 2.6.2. *For any rider $r \in R(\delta) \bigcup_{j=1}^{k-1} R^*(\delta_{i_j})$, we have*

$$d(r, \delta) \leq 3OPT_2$$

Proof. If $r \in R(\delta)$ then by definition we have $d(r, \delta) \leq OPT_2$. Now suppose $r \in R^*(\delta_{i_j})$ for $j \in [k-1]$. Let r' be the rider from scenario $R_k(\delta)$ that was matched to δ_{i_j} in the optimal solution.

$$d(r, \delta) \leq d(r, \delta_{i_j}) + d(\delta_{i_j}, r') + d(r', \delta) \leq 3OPT_2.$$

□

From Claim 2.6.2, we see that the ball centered at δ, with radius $3OPT_2$, contains at least $n - \ell$ second stage riders in $R(\delta) \bigcup_{j=1}^{k-1} R^*(\delta_{i_j})$. This proves the first part of the lemma. The second part is proved in the next claim.

Claim 2.6.3. *For $r_1 \in R_k(\delta)$ and $r_2 \in R_k(\delta) \bigcup_{j=1}^{k-1} R^*(\delta_{i_j})$, we have*

$$d(r_1, r_2) \leq 4OPT_2$$

Proof. Let $r_1 \in R_k(\delta)$. If $r_2 \in R_k(\delta)$ then $d(r_1, r_2) \leq d(r_1, \delta) + d(\delta, r_2) \leq 2OPT_2$. If $r_2 \in R^*(\delta_{i_j})$

for some j, and r' is the rider from scenario $R_k(\delta)$ that was matched to δ_{i_j}

$$d(r_1, r_2) \leq d(r_1, \delta) + d(\delta, r') + d(r', \delta_{i_j}) + d(\delta_{i_j}, r_2) \leq 4OPT_2.$$

□

□

Now, let us focus on the rest of second stage riders. We introduce the following definition. We say that a rider $r \in R_2$ is an *outlier* if $d(\delta, r) > 3OPT_2$. Denote $\{o_1, o_2, \ldots, o_\ell\}$ the farthest ℓ riders from δ with $d(\delta, o_1) \geq d(\delta, o_2) \geq \ldots \geq d(\delta, o_\ell)$. Note that by Lemma 2.6.2, the $n - \ell$ riders in $\mathcal{B}$ are not outliers and the only potential outliers could be in $\{o_1, o_2, \ldots, o_\ell\}$. Let j^* be the threshold such that $o_1, o_2, \ldots, o_{j^*}$ are outliers and $o_{j^*+1}, \ldots, o_\ell$ are not, with the convention that $j^* = 0$ if there is no outlier. There are $\ell+1$ possible values for j^*. We call each of these possibilities a *configuration*. For $j = 0, \ldots, \ell$, let C_j be the configuration corresponding to threshold candidate j. Note that C_0 is the configuration where there is no outlier and C_{j^*} is the correct configuration (See Figure 2.3).

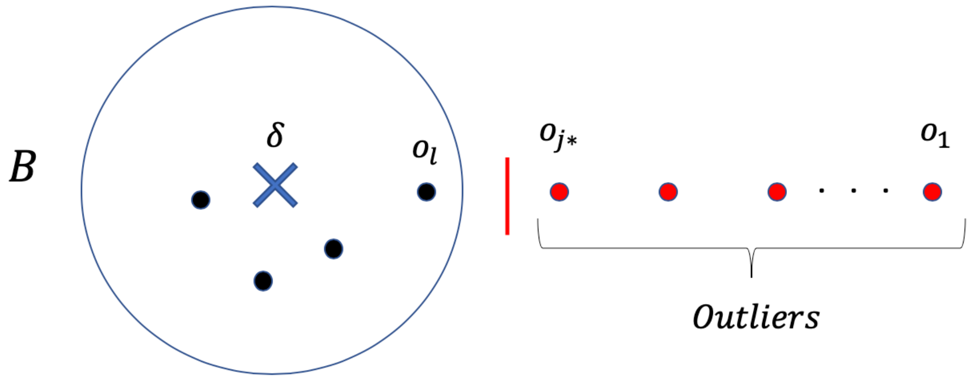

Figure 2.3: Configuration C_{j^*}

Now, we are ready to describe our algorithm. Recall that $R_k(\delta)$ are the closest k second-stage riders to δ. For the sake of simplicity, we denote $S_1 = R_k(\delta)$ and $S_2 = \{o_1 \ldots o_\ell\}$. Note that S_2 is a feasible scenario since $\ell < k$. For every configuration C_j, we form a representative scenario using

S_1 and $\{o_1 \dots o_j\}$. We solve TSRMB with this single representative scenario $S_1 \cup \{o_1 \dots o_j\}$ and denote $D_1(j)$ the corresponding optimal solution, i.e.,

$$D_1(j) = \text{TSRMB-1-Scenario}(R_1, S_1 \cup \{o_1 \dots o_j\}, D).$$

Since we cannot evaluate the cost of $D_1(j)$ on all scenarios because we can have exponentially many, we evaluate the cost of $D_1(j)$ on the two proxy scenarios S_1 and S_2. We finally show that the candidate $D_1(j)$ with minimum cost over these two scenarios, gives a constant approximation to our original problem (See Theorem 2.6.1). The details of our algorithm are summarized below.

Algorithm 5 Implicit scenarios with small surplus and $k \leq \sqrt{\frac{n}{2}}$.

```
Input: First stage riders $R_1$, second stage riders $R_2$, size $k$ and drivers $D$.
Output: First stage decision $D_1$.
1: Set $\delta$ := driver given by (2.3).
2: Set $S_1$:= the closest $k$ second stage riders to $\delta$.
3: Set $S_2 := \{o_1, \dots, o_\ell\}$ the farthest $\ell$ second stage riders from $\delta$ ($o_1$ being the farthest).
4: for $j = 0, \dots, \ell$ do
5:     $D_1(j) :=$ TSRMB-1-Scenario$(R_1, S_1 \cup \{o_1 \dots o_j\}, D)$.
6: return $D_1 = \arg\min_{D_1(j):\ j \in \{0,\dots,\ell\}} cost_1\big(D_1(j), R_1\big) + \max_{S \in \{S_1, S_2\}} cost_2\big(D \setminus D_1(j), S\big)$.
```

Theorem 2.6.1. *Algorithm 5 yields a solution with total cost less than $3OPT_1 + 17OPT_2$ for TSRMB with implicit scenarios when $k \leq \sqrt{\frac{n}{2}}$ and $\ell < k$.*

Proof of Theorem 2.6.1. We present here a sketch of the proof. The complete details and proofs of the claims appear in Appendix A.2. For all $j \in \{0, \dots, \ell\}$, denote

$$\Omega_j = cost_1\big(D_1(j), R_1\big)$$

$$\Delta_j = cost_2\big(D \setminus D_1(j), S_1 \cup \{o_1, \dots, o_j\}\big)$$

$$\beta_j = cost_1\big(D_1(j), R_1\big) + \max_{S \in \{S_1, S_2\}} cost_2\big(D \setminus D_1(j), S\big)$$

Recall f the objective function of TSRMB. In particular,

$$f\big(D_1(j)\big) = cost_1\big(D_1(j), R_1\big) + \max_{S \in \mathcal{S}} cost_2\big(D \setminus D_1(j), S\big)$$

Our proof is based on the following two claims. Claim 2.6.4 establishes a bound on the cost of $D_1(j^*)$ when evaluated on the proxy scenarios S_1 and S_2 and on all the scenarios in $\mathcal{S}$. Recall that j^* is the threshold index for the outliers as defined earlier in our construction. Claim 2.6.5 bounds the cost of $f(D_1(j))$ for any j. The proofs of both claims are presented in Appendix A.2.

Claim 2.6.4. $\Omega_{j^*} + \Delta_{j^*} \leq OPT_1 + OPT_2.$ *and* $f(D_1(j^*)) \leq OPT_1 + 5OPT_2.$

Claim 2.6.5. *For all* $j \in \{0, \ldots, l\}$ *we have,* $\beta_j \leq f(D_1(j)) \leq \max\{\beta_j + 4OPT_2,\ 3\beta_j + 2OPT_2\}.$

Suppose Algorithm 5 returns $D_1(\tilde{j})$ for some $\tilde{j}$. From Claim 2.6.5 and the minimality of $\beta_{\tilde{j}}$:

$$f\big(D_1(\tilde{j})\big) \leq \max\{\beta_{\tilde{j}} + 4OPT_2, 3\beta_{\tilde{j}} + 2OPT_2\} \leq \max\{\beta_{j^*} + 4OPT_2, 3\beta_{j^*} + 2OPT_2\}.$$

From Claim 2.6.4 and Claim 2.6.5, we have $\beta_{j^*} \leq f\big(D_1(j^*)\big) \leq OPT_1 + 5OPT_2.$ We conclude that,

$$f\big(D_1(\tilde{j})\big) \leq \max\big\{OPT_1 + 9OPT_2, 3OPT_1 + 17OPT_2\big\} = 3OPT_1 + 17OPT_2.$$

□

2.6.3 Arbitrary surplus with $k = 1$

In this part, we consider TSRMB when the surplus can be arbitrary and each of the second stage scenarios has a single rider ($k = 1$). We present a constant approximation algorithm for this case. Recall that TSRMB is NP-hard to approximate within a factor better than 2 even when $k = 1$. In this case, the second stage objective function aims to minimize the maximum distance from the remaining drivers to the second stage riders. We show that our problem is closely related to an instance of the p-supplier problem [55, 130]. This is a a variant of the p-center problem on a bipartite graph where centers can only belong to one side of the graph. The idea of our algorithm is to save a set of drivers to the second-stage by solving a p-supplier problem for the second stage riders (using the 3-approximation algorithm in [55]). Moreover, we reduce this set by pruning drivers that are close to each others within a threshold distance that depends on OPT_2. Note that we can assume that we know OPT_2 since the number of scenarios is exactly n and therefore we can evaluate any feasible solution in polynomial time. We show in Theorem 2.6.2 that the solution returned by Algorithm 6 gives a constant approximation.

Algorithm 6 Implicit scenarios with arbitrary surplus and $k = 1$.

```
Input: First stage riders $R_1$, second stage riders $R_2$, drivers $D$ and value of $OPT_2$.
Output: First stage decision $D_1$.
1: Set $p := |D| - |R_1|$.
2: Solve the $p$-supplier problem on the bipartite graph $D \cup R_2$, with centers in $D$ using the 3-
   approximation algorithm in [55].
3: Set $D_2$:= set of centers in the solution of the above $p$-supplier problem.
4: for $\delta \in D_2$ do
5:     if there exists another driver $\delta' \in D_2$ such that $d(\delta, \delta') \leq 8OPT_2$. then
6:         $D_2 := D_2 \setminus \{\delta'\}$.
7: $M$ := minimum weight maximum cardinality matching $R_1$ and $D \setminus D_2$.
8: return $D_1$ := drivers used in $M$.
```

Theorem 2.6.2. *Algorithm 6 yields a solution with total cost less than $OPT_1 + 15OPT_2$ for TSRMB with implicit scenarios and $k = 1$.*

Before presenting the proof of Theorem 2.6.2, let us introduce the p-supplier problem. The problem consists of n points in a metric space, that are partitioned into a client set C and a set of facilities F. Additionally, we are given a bound $p \leq |F|$. The objective is to open a set $S \subset F$ of p facilities that minimizes the maximum distance of a client to its closest open facility. The p-supplier problem is a generalization of the p-center problem, where the client and facility sets are identical (see [55, 131] for more details). We use the 3-approximation algorithm presented in [55] as a subroutine in Step 2 of Algorithm 6.

To prove the theorem, we present the following two claims:

Claim 2.6.6. *For all $r \in R_2$, there exists $\delta \in D_2$ s.t. $d(r, \delta) \leq 11OPT_2$.*

Proof of Claim 2.6.6. Let D_1^* be an optimal solution for the TSRMB problem with $k = 1$. $D \setminus D_1^*$ is a feasible solution for the p-supplier problem on the bipartite graph $D \cup R_2$, with centers in D. The p-supplier cost for $D \setminus D_1^*$ is equal to OPT_2. Therefore, the 3-approximation computed in Step 2 of the algorithm has a cost less than $3OPT_2$. This implies that initially, and before Step 4 of the algorithm, for every $r \in R_2$ there exists $\delta \in D_2$ s.t. $d(r, \delta) \leq 3OPT_2$. If a driver δ' is deleted in the loop of Step 4, then it is because there exists $\delta \in D_2$ s.t. $d(\delta, \delta') \leq 8OPT_2$. This implies that all the riders that were within distance $3OPT_2$ from δ' are now within a distance less than $3OPT_2 + d(\delta, \delta') \leq 11OPT_2$ from δ. □

Claim 2.6.7. *For $\delta \in D_2$, there exists $r \in R_2$ s.t. $d(r, \delta) \leq 3OPT_2$.*

Proof of Claim 2.6.7. Initially, every driver in D_2 has at least one rider in R_2 that is within a distance less than $3OPT_2$. Since the drivers that are not deleted from D_2 can only increase the set of riders to which they are the closest, every driver in D_2 always keeps at least one rider within a distance less than $3OPT_2$. □

Proof of Theorem 2.6.2. We show that we can match R_1 to $D \setminus D_2$ with a first stage cost less than $OPT_1 + 4OPT_2$. Let $r \in R_1$. If the optimal first stage driver of r is not in D_2, we simply match r to this optimal driver. On the other hand, suppose there exist $r_i, r_j \in R_1$ such that the optimal first stage drivers of r_i and r_j, respectively δ_i and δ_j, are both used in D_2. We show that we can match r_i and r_j to two different drivers in $D \setminus D_2$ within a distance less than $OPT_1 + 4OPT_2$. Since $\delta_i, \delta_j \in D_2$, there exists two different second stage riders $s_i \in R_2$ and $s_j \in R_2$ such that

$$d(\delta_i, s_i) \leq 3OPT_2,$$
$$d(\delta_j, s_j) \leq 3OPT_2.$$

It is clear that $s_i \neq s_j$, because otherwise $d(\delta_i, \delta_j) \leq 6OPT_2$ and either δ_i or δ_j would have been deleted from D_2. Let δ_i^2 and δ_j^2 be the optimal second stage drivers for s_i and s_j respectively.

We argue that $\delta_i^2 \neq \delta_j^2$. Suppose $\delta_i^2 = \delta_j^2$, then

$$d(s_i, s_j) \leq d(\delta_i^2, s_i) + d(\delta_j^2, s_j) \leq 2OPT_2,$$

and

$$d(\delta_i, \delta_j) \leq d(\delta_i, s_i) + d(s_i, s_j) + d(s_j, \delta_j) \leq 8OPT_2, \tag{2.4}$$

but (2.4) implies that either δ_i or δ_j would have been deleted in step 3 of the algorithm. Therefore $\delta_i^2 \neq \delta_j^2$, and r_i (resp. r_j) can be matched to δ_i^2 (resp. δ_j^2) within a distance less than

$$d(r_i, \delta_i^2) \leq d(r_i, \delta_i) + d(\delta_i, s_i) + d(s_i, \delta_i^2) \leq OPT_1 + 4OPT_2.$$

We showed the existence of a matching between R_1 and D_1 with an average weight less than $OPT_1 + 4OPT_2$. We know from Claim 2.6.6 that the second stage cost is less than $11OPT_2$. Therefore the total cost of the first stage decision D_1 is less than $OPT_1 + 15OPT_2$. $\square$

2.7 Numerical Experiments

In this section, we present an empirical comparison of Algorithm 2 with the greedy algorithm. We use a taxi data set from the city of Shenzhen to create realistic instances of the TSRMB problem.

2.7.1 Data

The data is collected for a month in the city of Shenzhen [125][2]. This data contains the GPS records of taxis in Shenzhen. The details of the data set are summarized in Table 2.1, where the sample rate means the interval between two adjacent GPS records. A trajectory is constructed by following one taxi between a pick-up ("Occupied" value change from 0 to 1) and a drop-off. A snapshot of the data is presented in Table 2.2.

[2]The raw trajectory record can be found in `https://github.com/cbdog94/STL`.

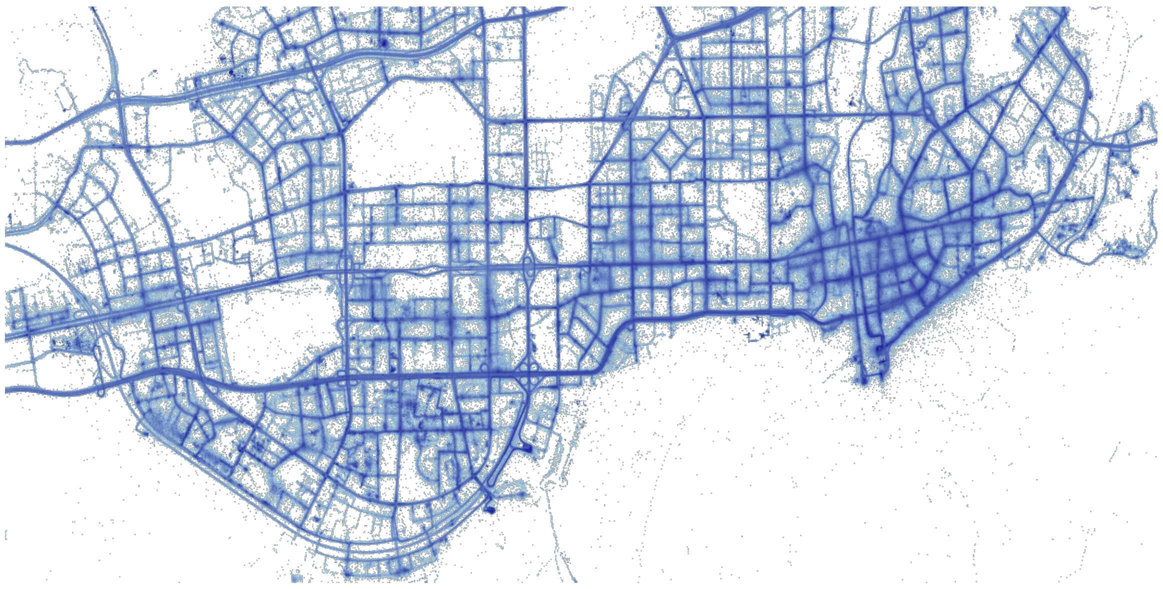

Figure 2.4: Union of the trajectories in Downtown Shenzhen on 09/17/2009

Size	# Taxis	# Trajectories	Sample rate	Avg trip time
32.7 GB	9,475	6,068,516	10-30 s	863s

Table 2.1: Details of the taxi trajectory data

Taxi ID	Time	Longitude	Latitude	Speed	Direction	Occupied
B97U79	2009-09-23 21:30:00	113.80275	22.66913	66	157	0
B97U79	2009-09-23 21:30:20	113.80137	22.67106	18	157	1

Table 2.2: Example of the taxi trajectory data

2.7.2 Experiment Setup

We focus on the GPS records of downtown Shenzhen, with $|\text{longitude} - 114.075| \leq 0.075$ and $|\text{latitude} - 22.54| \leq 0.03$ (See Figure 2.4). In a specific time range, we locate the riders by following taxis and observing when the occupied entry changes from 0 to 1. This change means that a pickup occurred and the rider's location is estimated to be the same as the taxi location at the time of pickup. For different days d of the month, and different times t of the day, we consider the pickups that were made in $[t, t + 1\text{min}]$ to be the first stage riders R_1. For the second stage riders, we construct two scenarios S_1, S_2 using the pickups that occurred in $[t + 1min, t + 2min]$ in $d - 7$ and $d - 14$ respectively, which represent the same day as d in the two previous weeks. We also construct the realized scenario S^*, which contains the pickups in $[t+1min, t+2min]$ of day d. We use the taxis of day d that were not occupied in the past 5 minutes before t to sample the set of drivers D. The edge weights correspond to the distances between drivers and riders. Note that in the data set, the number of all available drivers in the past 5 minutes is considerably higher than the number of pickups. Hence, to simulate a busier time, we randomly sample $2.5 \times |R_1|$ drivers in every instance. For every instance, we preform 10 random driver samples, solve the problem for every sample, and report the average. We report results from different times of the 17th day of the month, with S_1 and S_2 constructed from the 3rd and the 10th day respectively.

2.7.3 Evaluation metrics and experimental results

We use Algorithm 2 to solve the TSRMB problem with second stage scenarios S_1 and S_2. We denote $Alg(S_1, S_2)$ the total cost of the solution returned by Algorithm 2 where the second stage cost is the worst-case cost over the scenarios $\{S_1, S_2\}$. We call this case *In-sample*. We denote $Gr(S_1, S_2)$ the total worst-case cost of the greedy solution that myopically solves the first stage and uses the remaining drivers to match S_1 or S_2. We compare the in-sample performance of Algorithm 2 with the greedy algorithm by computing the ratio $Gr(S_1, S_2)/Alg(S_1, S_2)$.

We also evaluate Algorithm 2 on out-of-sample data. In particular, we consider the solution (first stage drivers D_1) returned by Algorithm 2 and use $D \setminus D_1$ to satisfy the realized scenario S^*. We call this case *Out-of-Sample*. The idea is to use S_1 and S_2 as a prediction for S^*. $Alg(S^*)$ denotes the total cost of our solution on the realized scenario S^*. $Gr(S^*)$ denotes the total cost of the greedy solution that myopically solves the first stage and uses the remaining drivers to match S^*. Finally, $OPT(S^*)$ denotes the cost of the optimal solution that knows offline the scenario S^*, i.e., $OPT(S^*)$ is the cost of TSRMB problem with a single scenario S^* which we compute using Algorithm 1. We compare the out-of-sample performance of Greedy and Algorithm 2 by computing the ratios $Gr(S^*)/OPT(S^*)$ and $Alg(S^*)/OPT(S^*)$. The in-sample and out-of-sample performances for different times on 09/17 are presented in Table 2.3. The columns “1st Stage” and “2nd Stage” denote the time range of the first and second stage respectively.

1st Stage	2nd Stage	$\|D\|$	$\|R_1\|$	$\|S^*\|$	Out-of-sample		In-sample
					$\frac{Gr(S^*)}{OPT(S^*)}$	$\frac{Alg(S^*)}{OPT(S^*)}$	$\frac{Gr(S_1,S_2)}{Alg(S_1,S_2)}$
09:00-01	09:01-02	215	86	97	1.52	1.34	1.52
10:00-01	10:01-02	187	75	54	1.73	1.31	1.42
11:00-01	11:01-02	210	84	78	1.50	1.34	1.30
12:00-01	12:01-02	215	86	91	1.51	1.44	1.31
13:00-01	13:01-02	205	82	93	1.52	1.34	1.30
14:00-01	14:01-02	342	137	138	1.68	1.59	1.74
15:00-01	15:01-02	355	142	120	1.59	1.29	1.55
16:00-01	16:01-02	345	138	132	1.48	1.30	1.41
17:00-01	17:01-02	295	118	113	1.36	1.26	1.18
18:00-01	18:01-02	287	115	102	1.46	1.36	1.38
19:00-01	19:01-02	300	120	112	1.33	1.20	1.37
20:00-01	20:01-02	307	123	134	1.94	1.64	1.60
21:00-01	21:01-02	370	143	147	1.77	1.38	1.40

Table 2.3: In-sample and out-of-sample comparison between Greedy, Algorithm 2.

1st Stage	2nd Stage	$\|D\|$	$\|R_1\|$	$\|S^*\|$	$Gr_2(S^*)/Alg_2(S^*)$	Total Matching Ratio
09:00-01	09:01-02	215	86	97	1.20	0.99
10:00-01	10:01-02	187	75	54	1.60	0.95
11:00-01	11:01-02	210	84	78	1.28	0.97
12:00-01	12:01-02	215	86	91	1.11	0.99
13:00-01	13:01-02	205	82	93	1.44	0.98
14:00-01	14:01-02	342	137	138	1.11	0.97
15:00-01	15:01-02	355	142	120	1.45	0.96
16:00-01	16:01-02	345	138	132	1.27	0.97
17:00-01	17:01-02	295	118	113	1.11	0.99
18:00-01	18:01-02	287	115	102	1.14	0.97
19:00-01	19:01-02	300	120	112	1.28	0.97
20:00-01	20:01-02	307	123	134	1.39	1.00
21:00-01	21:01-02	370	143	147	1.40	0.98

Table 2.4: Comparison of Algorithm 2 and Greedy on out-of-sample w.r.t the total matching weight and the second stage bottleneck.

Furthermore, in Table 2.4, we compare the second stage cost (bottleneck cost) of Greedy and of our solution on the out-of-sample scenario S^*. In particular, we report the ratio $Gr_2(S^*)/Alg_2(S^*)$, where $Gr_2(S^*)$ is the second stage cost if we use Greedy and the second stage scenario is S^* and $Alg_2(S^*)$ is our second stage cost for scenario S^* after we have used the solution returned by Algorithm 2. We also compute the ratio between the total weight of the greedy solution on S^*, and the total weight of the solution given by our algorithm when evaluated on S^*. This ratio is presented in the column "Total Matching Ratio".

2.7.4 Discussion

We observe from Table 2.3 that our two-scenarios algorithm improves significantly upon the greedy algorithm both in-sample and out-of-sample. In-sample, our algorithm improves the total cost by an average of 42%. Out-of-sample, Table 2.3 shows that the greedy algorithm can be sub-optimal within 58% on average as compared to the optimal that knows offline the realization of the second-stage. Our two-scenarios algorithm performs significantly better and is only 36% higher than the optimal on average. Since Greedy, by definition, returns the best first stage cost, the two-scenarios algorithm can only improve upon Greedy by reducing the second stage bottleneck without considerably increasing the first stage cost. In particular, we observe from Table 2.4 that the second stage bottleneck of our algorithm on the realized scenario S^* is significantly less than the bottleneck of the greedy algorithm (by 30% on average), while the total weight of the matching provided by the two algorithms is roughly similar. If we think of the edge weights between drivers and riders as the wait times, then our results show that we substantially reduce the maximal second stage wait time, while the average wait time over the two stages is almost unchanged (only 2.4% higher in average). Our algorithm introduces more fairness in the distribution of the wait time between first and second stage, by reducing the maximum wait time, without materially affecting the overall average wait time.

2.8 Other cost metrics

In this section, we initiate the study of other variants of two-stage matching problems, under both robust and stochastic models of uncertainty and for different cost functions. We define these problems, study their hardness of approximation and design approximation algorithms in some specific cases. We summarize our results below and defer all the details to Appendix A.3 and Appendix A.4.

1. **Two-Stage Stochastic Matching Bottleneck Problem** (TSSMB). In this problem, the first stage cost is the same as the TSRMB (e.g. the average matching weight). However, we

assume that we have an explicit list of scenarios $\mathcal{S} = \{S_1, \ldots, S_p\}$. Scenario S_i is realized with probability p_i. The objective is to minimize the function

$$\min_{D_1 \subset D} \left\{ cost_1(D_1, R_1) + \sum_{i=1}^{p} p_i \cdot cost_2(D \setminus D_1, S_i) \right\},$$

where $cost_2(D \setminus D_1, S_i)$ is the bottleneck matching cost between $D \setminus D_1$ and scenario S_i. In Appendix A.3, we show this problem is NP-hard to approximate within a factor better than $4/3$. We also provide an algorithm that yields a 3-approximation when there is no surplus.

2. **Two-Stage Robust Matching Problem** (TSRM). In this problem, the cost of the first stage is the total weight of the first stage matching, and the second stage cost is the total weight of the worst case matching over scenarios. We present the formal definition of this problem in Appendix A.4, and show it is NP-hard even with two scenarios. Kalyanasundaram and Pruhs [82] consider the online version of this problem, and show that the greedy algorithm is 3-competitive for two stages and therefore yields a 3-approximation in the worst-case as well. We further improve this result and show a $7/3$-approximation when there is no surplus.

3. **Two-Stage Robust Bottleneck Bottleneck Problem** (TSRBB). The only difference from the TSRMB is that the first stage cost is the bottleneck of the first stage matching. That is, the cost functions for the TSRBB are:

$$cost_1(D_1, R_1) = \max_{(i,j) \in M_1} d(i,j), \quad \text{and} \quad cost_2(D \setminus D_1, S) = \max_{(i,j) \in M_2^S} d(i,j).$$

All our hardness and approximation algorithms from the TSRMB easily carry to this problem.

Chapter 3: Learning Hypermatchings

In the second part of this thesis, we switch our focus from graphs to the more general concept of hypergraphs. Our ultimate goal is to tackle the problem of finding maximum matchings in hypergraphs, which we will cover in Chapter 4. This chapter first concentrates on the problem of learning hidden hypergraph matchings through membership queries. This chapter is based on the article [19] written in collaboration with Eric Balkanski and Shatian Wang. The article is set to appear in the *Conference on Learning Theory (COLT)*, 2022.

3.1 Introduction and Problem Formulation

As we highlighted in Chapter 1, hypergraphs are a powerful tool in computational chemistry and molecular biology where they are used to represent groups of chemicals and molecules that cause a reaction. In such hypergraphs, vertices represent chemicals and molecules, and edges represent groups of vertices that cause a reaction. The chemicals and molecules that cause a reaction are however often unknown a priori, and learning such groups is a central problem of interest that has motivated a long line of work on hypergraph learning in the edge-detecting queries model [132, 133, 134, 135, 136, 137]. In this model, a learner queries subsets of vertices and, for each queried subset, observes whether it contains an edge or not. When the vertices represent chemicals, the learner queries groups of chemicals and observes a reaction if a subgroup reacts. The goal is to learn the edges with a small number of queries, i.e., a small number of experiments.

An important application of learning chemical reaction networks is to learn effective drug combinations (drugs are vertices and effective drug combinations are hyperedges). In particular, there has recently been a lot of interest in finding drug combinations that reduce cancer cell viability. For example, as part of AstraZeneca's recent DREAM Challenge, the effectiveness of a large number of drug combinations was tested against different cancer cell lines [138]. Despite this interest, as noted by Flobak et al. [139], "our knowledge about beneficial targeted drug combinations is still limited, partly due to the combinatorial complexity", especially since the time required to culture, maintain and test cell lines is significant. This combinatorial complexity motivates the need for novel computational methods that efficiently explore drug combinations.

One main obstacle to discovering effective combinations that involve $d > 2$ vertices is that $\Omega((2m/d)^{d/2})$ queries are required to learn hypergraphs with m edges of maximum size d [134]. Since there is no efficient algorithm for learning general hypergraphs with large maximum edge size, the main question is the following: which families of hypergraphs can we learn with a number of queries that does not grow exponentially in the size of the edges?

Our results. We develop the first algorithms with $\mathrm{poly}(n, m)$ queries for learning non-trivial families of hypergraphs that have a super-constant number of edges m of super-constant size. The main family of such hypergraphs that we consider are hypermatchings, i.e., hypergraphs such that every vertex is in at most one edge. Hypermatchings arise in a number of applications such as crew scheduling [140], coalitions in multi-agent systems [141] as well as combinatorial auctions [142], and learning hypermatchings generalizes the problem of learning matchings that is studied in [143, 144]. In addition to their query complexity, we are also interested in the adaptive complexity of our algorithms, which is the number of adaptive rounds of queries required when the algorithm can perform multiple queries in parallel in each round. Our first main result is an algorithm with near-linear query complexity and poly-logarithmic adaptive complexity for learning hypermatchings.

Theorem. *There is a $O(\log^3 n)$-adaptive algorithm with $O(n \log^5 n)$ queries that, for any hypermatching M over n vertices, learns M with high probability.*

The adaptivity of the algorithm can be improved to $O(\log n)$ at the expense of $O(n^2 \log^3 n)$ queries. We complement our algorithm by showing that there are no $o(\log \log n)$-adaptive algorithms that learn hypermatchings using $\mathrm{poly}(n)$ queries.

Theorem. *There is no $o(\log\log n)$-adaptive algorithm with polynomial query complexity that learns an arbitrary hypermatching M over n vertices, even with small probability.*

Moving beyond hypermatchings, our techniques can be applied to learn hypergraphs whose vertices have bounded maximum degree $\Delta \geq 2$ ($\Delta = 1$ for hypermatchings). For such hypergraphs, the previously mentioned $\Omega((2m/d)^{d/2})$ lower bound on the number of queries needed by any algorithm implicitly also implies that $\Omega((m/\rho)^{\rho})$ queries are required, even when $\Delta = 2$, where $\rho \geq 1$ is the ratio between the maximum and minimum edge sizes [134]. Thus, an exponential dependence on this near-uniform edge sizes parameter ρ is, in general, unavoidable. A non-adaptive algorithm can be given with query complexity that depends on the maximum degree Δ and the near-uniform parameter ρ of the hypergraph H we wish to learn.

Our learning algorithms rely on novel constructions of collections of queries that satisfy a simple property that we call unique-edge covering, which is a general approach for constructing learning algorithms with a query complexity that does not grow exponentially in the size of the edges. We believe that an interesting direction for future work is to identify, in addition to hypermatchings, other relevant families of hypergraphs that are learnable with $\text{poly}(n, m)$ queries, even when both the maximum edge size d and the edge size ratio ρ are non-constant.

Technical overview. Previous work on learning a hypergraph $H = (V, E)$ relies on constructing a collection of sets of vertices called an *independent covering family* [133] or a *cover-free family* [136, 145, 146, 135]. These families aim to identify *non-edges*, which are sets of vertices $T \subseteq V$ that do not contain an edge $e \in E$. More precisely, both of these families are a collection $\mathcal{S}$ of sets of vertices such that every non-edge set T of size d is contained in at least one set $S \in \mathcal{S}$ that does not contain an edge. These families have a minimum size that is exponential in d and these approaches require a number of queries that is at least the size of these families. In contrast to these existing approaches that focus on non-edges, we construct a collection $\mathcal{S}$ of sets of vertices such that every edge e is contained in at least one set $S \in \mathcal{S}$ that does not contain any other edge of H. We call such a collection a *unique-edge covering family*. Since the number of edges in a hypergraph H with maximum degree Δ is at most Δn, there exists unique-edge covering families whose sizes depend on the maximum degree Δ of H instead of the maximum edge size d.

The algorithm for learning a hypermatching M proceeds in phases. In each phase, we construct a unique-edge covering family of a subgraph of M containing all edges of size that is in a specified range. This unique-edge covering family is constructed with i.i.d. sampled sets that contain each vertex, independently, with probability depending on the edge size range. The edge size range is widened at the end of each phase. One main challenge with hypermatchings is to obtain a near-linear query complexity. We do so by developing a subroutine which, given a set S that covers a unique edge of size at most s, finds this unique edge with $O(s \log |S|)$ queries. For the lower bound for hypermatchings, there is a simple construction that shows that there are no non-adaptive algorithms for learning hypermatchings with $\text{poly}(n)$ queries. The technical part of interest is extending this construction and its analysis to hold for $\Omega(\log \log n)$ rounds. Finally, for low-degree near-uniform hypergraphs H, we construct a unique-edge covering family in a similar manner to those for hypermatchings ($\Delta = 1$). The main technical difficulty for hypergraphs with maximum degree $\Delta > 1$ is in analyzing the number of samples required to construct a unique-edge covering family, which is significantly more involved than for hypermatchings due to overlapping edges.

3.2 Preliminaries

As defined in Chapter 1, a hypergraph $H = (V, E)$ consists of a set of n vertices V and a set of m edges $E \subseteq 2^V$. We abuse notation in this chapter and write $e \in H$ for edges $e \in E$. For the problem of learning a hypergraph H, the learner is given V and an edge-detecting oracle Q_H. Given a query $S \subseteq V$, the oracle answers $Q_H(S) = 1$ if S contains an edge e, i.e. there exists $e \in H$ such that $e \subseteq S$; otherwise, $Q_H(S) = 0$. The goal is to learn the edge set E using a small number of queries. When queries can be evaluated in parallel, we are interested in algorithms with low adaptivity. The *adaptivity* of an algorithm is the number of sequential rounds it makes where, in each round, queries are evaluated in parallel. An algorithm is non-adaptive if its adaptivity is 1.

We recall that degree of a vertex v is the number of edges $e \in H$ such that $v \in e$. The maximum degree Δ of H is the maximum degree of a vertex $v \in V$. When $\Delta = 1$, every vertex is in at most one edge and H is called a hypermatching, which we also denote by M. The rank d of H is the maximum size of an edge $e \in H$, i.e., $d := \max_{e \in H} |e|$. The edge size ratio of a hypergraph H is $d/\min_{e \in H} |e|$. A graph is ρ-near-uniform and is uniform if $d/\min_{e \in H} |e| \leq \rho$ and $d/\min_{e \in H} |e| = 1$ respectively.

3.3 Literature Review

The problem of learning a hidden hypergraph with edge-detecting queries was first introduced by Torney [132] for complex group testing, where we have a set of n items, and m defective groups that are known as positive complexes, and we can choose an arbitrary group $S \subseteq [n]$ and ask whether S contains at least one positive complex. This problem is also known as exact learning from membership queries of a monotone DNF with at most m monomials, where each monomial contains at most d variables [147, 134, 148].

Regarding hardness results, non-adaptive algorithms for learning hypergraphs with m edges of size at most d require $\Omega(N(m,d) \log n)$ queries where $N(m,d) = \frac{m+d}{\log \binom{m+d}{d}} \binom{m+d}{d}$ [135, 146]. Angluin et al. [134] show that $\Omega((2m/d)^{d/2})$ queries are required by any (adaptive) algorithm for learning general hypergraphs, and then extended this lower bound in [133] to $\Omega((2m/(\delta+2))^{1+\delta/2})$ for learning near-uniform hypergraphs with maximum edge size difference $\delta = d - \min_{e \in H} |e|$.

Regarding adaptive algorithms, Angluin et al. [133] give an algorithm that learns a hypergraph with $O(2^{O((1+\frac{\delta}{2})d)} \cdot m^{1+\frac{\delta}{2}} \cdot poly(\log n))$ queries in $O((1+\delta) \min\{2^d(\log m + d)^2, (\log m + d)^3\})$ rounds with high probability. When $m < d$, Abasi et al. [135] give a randomized algorithm that achieves a query complexity of $O(dm \log n + (d/m)^{m-1+o(1)})$ and show an almost matching $\Omega(dm \log n + (d/m)^{m-1})$ lower bound. When $m \geq d$, they present a randomized algorithm that asks $(cm)^{d/2+0.75} + dm \log n$ queries for some constant c, almost matching the $\Omega((2m/d)^{d/2})$ lower bound of [134]. Chodoriwsky and Moura [149] develop an adaptive algorithm with $O(m^d \log n)$ queries. Adaptive algorithms with $O\left(md(\log_2 n + (md)^d)\right)$ queries are constructed in [150]. Regarding non-adaptive algorithms, Chin et al. [151] and Abasi et al. [136] give deterministic non-adaptive algorithms with an almost optimal query complexity $N(m, d)^{1+o(1)} \log n$. The problem of learning a hypergraph when a fraction of the queries are incorrect is studied in [152] and [137]. We note that, to the best of our knowledge, all existing algorithms require a number of queries that is exponential in d, or exponential in m when $m < d$. We develop the first algorithms using $\text{poly}(n)$ queries for learning non-trivial families of hypergraphs that have arbitrarily large edge sizes and a super-constant number of edges m.

Finally, learning a matching has been studied in the context of graphs ($d = 2$) [143, 144]. In particular, Alon et al. [143] provide a non-adaptive algorithm with $O(n \log n)$ queries. To the best of our knowledge, there is no previous work on learning hypermatchings, which generalizes matchings to hypergraphs of maximum edge size $d > 2$. Our lower bound shows that, in contrast to algorithms for learning matchings, algorithms for learning hypermatchings require multiple adaptive rounds. Similar to the algorithm in [143] for learning matchings, our algorithm for learning hypermatchings has a near-linear query complexity.

3.4 Learning algorithm for hypermatchings

In this section, we study the problem of learning hypermatchings, i.e., hypergraphs of maximum degree $\Delta = 1$. We present an algorithm that, with high probability, learns a hypermatching with $O(n \log^5 n)$ queries in $O(\log^3 n)$ rounds. The number of rounds can be improved to $O(\log n)$, at the expense of an $O(n^2 \log^3 n)$ query complexity. In Section 3.5, we show that there is no $o(\log \log n)$-round algorithm that learns hypermatchings with $\text{poly}(n)$ queries. Full proofs are deferred to Appendix B.1.1 and B.1.2.

A central concept we introduce for our learning algorithms is the definition of a unique-edge covering family of a hypergraph H, which is a collection of sets such that every edge $e \in H$ is contained in at least one set that does not contain any other edge.

Definition 3.4.1. *A collection $\mathcal{S} \subseteq 2^V$ is a* ***unique-edge covering family*** *of H if, for every $e \in H$, there exists $S \in \mathcal{S}$ s.t. e is the unique edge contained in S, i.e. $e \subseteq S$ and $e' \not\subseteq S$ for all $e' \in H, e' \neq e$.*

Efficiently searching for the unique edge in a set. We first show that, given a unique-edge covering family $\mathcal{S}$ of a hypermatching M, we can learn M. We observe that a set of vertices $S \subseteq V$ contains a unique edge if and only if $Q_M(S) = 1$ and $Q_M(S \setminus \{v\}) = 0$ for some $v \in S$ and that if it contains a unique edge, this edge is $e = \{v \in S : Q_M(S \setminus \{v\}) = 0\}$. This observation immediately leads to the following simple algorithm called FINDEDGEPARALLEL.

Algorithm 7 FINDEDGEPARALLEL(S), returns e if S contains a unique edge e

```
Input: set $S \subseteq V$ of vertices
1: if $Q_M(S) = 1$ and $\exists v \in S$ s.t. $Q_M(S \setminus \{v\}) = 0$ then return $\{v \in S : Q_M(S \setminus \{v\}) = 0\}$
2: return None
```

The main lemma for FINDEDGEPARALLEL follows immediately from the above observation.

Lemma 3.4.1. *For any hypermatching M and set $S \subseteq V$,* FINDEDGEPARALLEL *is a non-adaptive algorithm with $|S| + 1$ queries that, if S contains a unique edge e, returns e and None otherwise.*

Proof. It is clear that FINDEDGEPARALLEL(S) makes at most $|S| + 1$ queries. If S does not contain an edge, then $Q_M(S) = 0$ and the algorithm returns None. If S contains at least two edges, then there is no intersection between the edges because M is a matching. Therefore, for every $v \in S$, $Q_M(S \setminus \{v\}) = 1$, and the algorithm returns None. The only case left is when S contains a unique edge e. In this case $Q_M(S) = 1$ and $e = \{v \in S, Q_M(S \setminus \{v\} = 0\}$. In this case, FINDEDGEPARALLEL(S) return e. □

In order to obtain a near-linear query complexity for learning a hypermatching, the query complexity of FINDEDGEPARALLEL needs to be improved. Next, we describe an $O(\log |S|)$-round algorithm, called FINDEDGEADAPTIVE, which finds the unique edge e in a set S with $O(s \log |S|)$ queries, assuming $|e| \leq s$. The algorithm recursively partitions S into two arbitrary sets S_1 and S_2 of equal size. If $Q_M(S_1) = Q_M(S_2) = 1$, then S contains at least two edges. If $Q_M(S_1) = 1$ and $Q_M(S_2) = 0$ (or similarly $Q_M(S_1) = 0$ and $Q_M(S_2) = 1$), then, assuming S contains a single edge, this edge is contained in S_1 and we recurse on S_1. The most interesting case is if $Q_M(S_1) = Q_M(S_2) = 0$. Assuming S contains a single edge e, this implies that both S_1 and S_2 contain vertices in e and we recurse on both S_1 and S_2. When recursing on S_1, the set S_2 needs to be included in future queries to find vertices in $e \cap S_1$ (and vice-versa when recursing on S_2). The algorithm thus also specifies an additional argument T for the recursive calls. This argument is initially the empty set, in the case where $Q_M(S_1) = Q_M(S_2) = 0$ the set S_2 is added to T when recursing on S_1 and vice-versa, and T is included in all queries in the recursive calls.

We denote the pairs $(S', T) \in \mathcal{S}$ at iteration i of the while loop by $\mathcal{S}^i = \{(S^i_j, T^i_j)\}_j$. In the case where S contains a single edge e and $|e| \leq s$, the algorithm maintains two invariants. The first is that all sets S^i_j contain at least one vertex $v \in e$. This invariant explains why, in the base case $|S^i_j| = 1$, we add the single vertex in S^i_j to the solution $\hat{e}$. The second invariant is that, for all vertices $v \in e$ and all iterations i, there exists a unique j such that v is contained in set S^i_j. Thanks to both invariants, when $|\mathcal{S}^i| > s$, we have that either set S contains more than one edge or that it contains a single edge of size greater than s, in which case the algorithm returns None. This stopping condition ensures that at most $2s$ queries are performed at each iteration and that the algorithm uses at most $2s \log_2 |S| + (s + 1)$ total queries.

Algorithm 8 FINDEDGEADAPTIVE(S, s), returns e if S contains a unique edge e and $|e| \leq s$

```
Input: set $S \subseteq V$, edge size $s$
 1: if $Q_M(S) = 0$ then return None
 2: $\mathcal{S} \leftarrow \{(S, \emptyset)\}, \hat{e} \leftarrow \{\}$
 3: while $|\mathcal{S}| > 0$ do
 4:     if $|\mathcal{S}| > s$ then return None
 5:     $\mathcal{S}' \leftarrow \{\}$
 6:     for each $(S', T) \in \mathcal{S}$ do (in parallel)
 7:         $S_1, S_2 \leftarrow$ partition $S'$ into two sets of size $|S'|/2$
 8:         if $|S'| = 1$ then add $v \in S'$ to $\hat{e}$
 9:         else if $Q_M(S_1 \cup T) = 1$ and $Q_M(S_2 \cup T) = 1$ then return None
10:         else if $Q_M(S_1 \cup T) = 1$ and $Q_M(S_2 \cup T) = 0$ then add $(S_1, T)$ to $\mathcal{S}'$
11:         else if $Q_M(S_1 \cup T) = 0$ and $Q_M(S_2 \cup T) = 1$ then add $(S_2, T)$ to $\mathcal{S}'$
12:         else if $Q_M(S_1 \cup T) = 0$ and $Q_M(S_2 \cup T) = 0$ then add $(S_1, S_2 \cup T), (S_2, S_1 \cup T)$ to $\mathcal{S}'$
13:     $\mathcal{S} \leftarrow \mathcal{S}'$
14: if $|\hat{e}| > s$, or $Q_M(\hat{e}) = 0$, or $Q_M(\hat{e} \setminus \{v\}) = 1$ for some $v \in \hat{e}$ then return None
15: return $\hat{e}$
```

We show that if FINDEDGEADAPTIVE(S, s) returns $\hat{e}$, then $\hat{e}$ is an edge. If there is a unique edge $e \subseteq S$ and $|e| \leq s$, then the algorithm returns $\hat{e} = e$.

Lemma 3.4.2. *If there is a unique edge $e \in M$ such that $e \subseteq S$ and if $|e| \leq s$, then* FINDEDGEADAPTIVE(S, s) *returns the edge e. Otherwise, it either returns None or an edge $e \in M$.* FINDEDGEADAPTIVE(S, s) *uses at most $2s \log_2 |S| + (s + 1)$ queries in at most $\log_2 |S|$ rounds.*

Before proving Lemma 3.4.2, we first show the invariants in FINDEDGEADAPTIVE. Let $\mathcal{S}^i = \{(S^i_1, T^i_1), \ldots, (S^i_\ell, T^i_\ell)\}$ be $\mathcal{S}$ at the beginning of the ith iteration of the while loop of FINDEDGEADAPTIVE(S, s).

Lemma 3.4.3. *Assume there is a unique edge $e \in M$ such that $e \subseteq S$ and that $|e| \leq s$, then, for every vertex $v \in e$ and at every iteration i of the while loop of* FINDEDGEADAPTIVE(S, s) *we have that*

1. *either $v \in \hat{e}$ or $v \in S^i_j$ for some $j \in [\ell]$*
2. *$Q_M(S^i_j \cup T^i_j) = 1$ for all $j \in [\ell]$*
3. *$e \cap S^i_j \neq \emptyset$ for all $j \in [\ell]$*

Proof of Lemma 3.4.3. We prove the lemma by induction on i. Base case: At the beginning of the first iteration, $\mathcal{S}^1 = \{(S, \emptyset)\}$ and $\hat{e} = \emptyset$, and we have the following 1) For every $v \in e$ we have $v \in S$. 2) $Q_M(S \cup \emptyset) = 1$. 3) $e \cap S = e \neq \emptyset$.

Assume that the statement of the lemma holds at the beginning of iteration $i \geq 1$. We will show that it holds at the beginning of iteration $i + 1$. Let $\mathcal{S}^{i+1} = \{(S^{i+1}_1, T^{i+1}_1), \ldots, (S^{i+1}_k, T^{i+1}_k)\}$ be $\mathcal{S}$ at the beginning of the $(i + 1)$th iteration of the while loop of FINDEDGEADAPTIVE(S, s).

1. Consider a vertex $v \in e$. If at the beginning of the ith iteration we had $v \in \hat{e}$, then $v \in \hat{e}$ at the beginning of the $(i + 1)$th iteration as well. Suppose now that $v \in S^i_j$ for some $j \in [\ell]$ at the start of iteration i. If $|S^i_j| = 1$ then v is added to $\hat{e}$ at Step 8. Assume $|S^i_j| > 1$, and let $T = T^i_j$. Then S^i_j is partitioned into two sets S_1 and S_2. If $Q_M(S_1 \cup T) = 1$ and $Q_M(S_2 \cup T) = 1$ then this contradicts the fact that S contains a unique edge. In fact, without loss of generality we can assume that $v \in S_1$, therefore $S_2 \cup T$ cannot include e and the only way $Q_M(S_2 \cup T) = 1$ is if S contains another edge. If $Q_M(S_1 \cup T) = 1$ and $Q_M(S_2 \cup T) = 0$ then we know that

$v \in S_1$ and (S_1, T) is added to $\mathcal{S}^{i+1}$. If $Q_M(S_1 \cup T) = 0$ and $Q_M(S_2 \cup T) = 1$ then we know that $v \in S_2$ and (S_2, T) is added to $\mathcal{S}^{i+1}$. If $Q_M(S_1 \cup T) = 0$ and $Q_M(S_2 \cup T) = 0$ then both $(S_1, S_2 \cup T)$ and $(S_2, S_1 \cup T)$ are added to $\mathcal{S}^{i+1}$, and we have either $v \in S_1$ or $v \in S_2$.

2. From steps 9 to 11 in FINDEDGEADAPTIVE(S, s), a couple (S_j^{i+1}, T_j^{i+1}) is only added to $\mathcal{S}^{i+1}$ when $Q_M(S_j^{i+1} \cup T_j^{i+1}) = 1$. In step 12, we have $Q_M(S_1 \cup T) = 0$ and $Q_M(S_2 \cup T) = 0$, and both $(S_1, S_2 \cup T)$ and $(S_2, S_1 \cup T)$ are added to $\mathcal{S}^{i+1}$. But we know from the induction assumption that at the beginning of iteration i we had $(S_1 \cup S_2, T) \in \mathcal{S}^i$, therefore by 2. $Q_M(S_1 \cup S_2 \cup T) = 1$. This ensures that also in this case, (S_j^{i+1}, T_j^{i+1}) is only added to $\mathcal{S}^{i+1}$ when $Q_M(S_j^{i+1} \cup T_j^{i+1}) = 1$.

3. We know from the induction hypothesis $e \cap S_j^i \neq \emptyset$ for all $j \in [\ell]$. For a fixed $j \in [\ell]$, S_j^i contains a vertex v from e. In the while loop, S_j^i is partitioned into S_1 and S_2, such that v is either in S_1 or in S_2. If $Q_M(S_1 \cup T_j^i) = 1$, then $v \in S_1$, (S_1, T_j^i) is added to $\mathcal{S}^{i+1}$ and $S_1 \cap e \neq \emptyset$. Similarly, if $Q_M(S_2 \cup T_j^i) = 1$, then $v \in S_2$, (S_2, T_j^i) is added to $\mathcal{S}^{i+1}$ and $S_2 \cap e \neq \emptyset$. Assume now that $Q_M(S_1 \cup T_j^i) = Q_M(S_2 \cup T_j^i) = 0$, then $(S_1, S_2 \cup T_j^i)$ and $(S_2, S_1 \cup T_j^i)$ are added to $\mathcal{S}^{i+1}$ and we need to show that $S_1 \cap e \neq \emptyset$ and $S_2 \cap e \neq \emptyset$. We know from 2. that $Q_M(S_1 \cup S_2 \cup T_j^i) = 1$. Therefore $Q_M(S_1 \cup T_j^i) = Q_M(S_2 \cup T_j^i) = 0$ implies that there is a vertex of e both in S_1 and S_2, therefore $S_1 \cap e \neq \emptyset$ and $S_2 \cap e \neq \emptyset$. This concludes this part and shows that whenever (S_j^{i+1}, T_j^{i+1}) is added to $\mathcal{S}^{i+1}$, we have $S_j^{i+1} \cap e \neq \emptyset$.

□

We are now ready to prove Lemma 3.4.2.

Proof of Lemma 3.4.2. First, assume that there is a unique edge $e \in M$ such that $e \subseteq S$ and that $|e| \leq s$. Consider $v \in S$. If $v \in e$, then by Lemma 3.4.3, we have that at every iteration i of the while loop. Either $v \in \hat{e}$ or $v \in S_j^i$ for some $j \in \ell$. Since $|S_j^i| = |S_{j'}^{i-1}|/2$ for all i, j, j', at iteration $i^\star = \log_2 n$, either $v \in \hat{e}$ or $v \in S_j^{i^\star}$ for some j and $|S_j^{i^\star}| = 1$, in which case v is then added to e by the algorithm. Next, if $v \notin e$, since $e \cap S_j^i \neq \emptyset$ for all i, j by Lemma 3.4.3, v is never added to e. Thus, FINDEDGEADAPTIVE(S, s) returns the edge e

Assume now that S does not contain any edges. In this case, every query is answered by a zero, and FINDEDGEADAPTIVE(S, s) returns None.

Finally, assume that S contains at least two edges. If FINDEDGEADAPTIVE(S, s) does not return None, then Step 14 ensure that the returned edge $\hat{e}$ has size at most s. Step 14 also ensures that $\hat{e}$ contains at least one edge. If $\hat{e}$ strictly contains an edge, then there is a vertex $v \in \hat{e}$ such that $Q_M(\hat{e}\backslash\{v\}) = 1$, in which case FINDEDGEADAPTIVE(S, s) would have returned None. Therefore, $\hat{e}$ is an edge of M of size less than s.

We now show that FINDEDGEADAPTIVE(S, s) runs in $\log_2 |S|$ rounds and makes at most $2s \log_2 |S| + (s + 1)$ queries. At every iteration of the while loop, FINDEDGEADAPTIVE(S, s) makes less than $2s$ queries (recall that if $|\mathcal{S}| > s$ then the algorithm returns None). We show that the number of iterations of the while loop is less than $\log_2 |S|$. At every iteration of the while loop, for every couple (S, T) in $\mathcal{S}$, the size of S is divided by 2. Therefore, after $\log_2 |S|$ iterations, either $\mathcal{S}$ is empty of for every couple (S, T) in $\mathcal{S}$ we have $|S| = 1$. This guarantees that no sets are added to $\mathcal{S}'$. Therefore, the number of iterations of the while loop is less than $\log_2 |S|$. This also shows that the adaptive complexity of FINDEDGEADAPTIVE(S, s) is less than $\log_2 |S|$. After the while loop, we make at most $s+1$ queries and the total number of queries is less than $2s \log |S| + s + 1$. □

Constructing a unique-edge covering family. Next, we give an algorithm called FINDDISJOINTEDGES that, for any hypermatching M, constructs a unique-edge covering family $\mathcal{S}$ of the α-near-uniform hypermatching $M_{s,\alpha,V'}$. $M_{s,\alpha,V'}$ is defined as the subgraph of M over edges of size between s/α and s and over vertices that are not in an edge of size less than s/α. The unique-edge covering family $\mathcal{S}$ is constructed with i.i.d. samples that contain each vertex in V', independently, with probability $n^{-\alpha/s}$. The algorithm then calls a FINDEDGE subroutine, which is either FINDEDGEPARALLEL or FINDEDGEADAPTIVE, on samples $S_i \in \mathcal{S}$ that contain at least one edge.

We show that $n^{\alpha} \log^2 n$ such samples suffices to construct a unique-edge covering family $\mathcal{S}$ of the α-near-uniform hypermatching $M_{s,\alpha,V'}$.

Lemma 3.4.4. *Assume $s/\alpha \geq 2$, $\alpha \geq 1$, that the* FINDEDGE *subroutine uses at most q queries*

Algorithm 9 FINDDISJOINTEDGES(s, α, V', FINDEDGE), returns edges of size between s/α and s

```
Input: edge size $s$, parameter $\alpha \geq 1$, set of vertices $V' \subseteq V$, subroutine FINDEDGE
1: $\hat{M} \leftarrow \emptyset$
2: for $i = 1, \dots, n^{\alpha} \log^2 n$ do (in parallel)
3:     $S_i \leftarrow$ set of independently sampled vertices from $V'$, each with probability $n^{-\alpha/s}$.
4:     if $Q_M(S_i) = 1$ then
5:         $e \leftarrow$ FINDEDGE($S_i, s$)
6:         if $e$ is not None then add $e$ to $\hat{M}$
7: return $\hat{M}$.
```

in at most r rounds, and let $\mathcal{S} = \{S_i\}_i$ be the $n^{\alpha} \log^2 n$ samples, then FINDDISJOINTEDGES *is an $(r+1)$-adaptive algorithm with $n^{\alpha} \log^2 n + 2\alpha q n \log^2 n/s$ queries such that, with probability $1 - O(n^{-\log n})$, $\mathcal{S}$ is a unique-edge covering family of the hypermatching $M_{s,\alpha,V'} = \{e \in M : e \subseteq V', s/\alpha \leq |e| \leq s\}$.*

We present the main ideas behind the proof before presenting it in details. We show that (1) every edge $e \in M$ is contained in between $(\log^2 n)/2$ and $2 \log^2 n$ samples w.h.p. (by the Chernoff bound) and (2) conditioned on $e \subseteq S_i$, the probability that S_i contains another edge is at most α/s (by a union bound over all other edges). This implies that, with probability $1 - e^{-\Omega(\log^2 n)}$, every edge e is contained in at least one sample that does not contain any other edge. Since there are at most $\alpha n/s$ edges in a hypermatching of minimum edge size s/α, the total number of samples S such that $Q_M(S) = 1$ is at most $(2\alpha n \log^2 n)/(s)$ and the query complexity is thus $n^{\alpha} \log^2 n + 2\alpha q n \log^2 n/s$.

Proof. We know that a sample S_i contains at least an edge if and only if $Q_M(S_i) = 1$. We also know from Lemma 3.4.1 and Lemma 3.4.2 that if S_i contains a unique edge e of size less than s, then the subroutine FINDEDGE(S_i, s) will return e. Therefore, to learn all edges of size in $[s/\alpha, s]$, we only need to make sure that each such edge appears in at least one sample that does not contain any other edges, in other words, that $\mathcal{S}$ is a unique-edge covering family for $M_{s,\alpha,V'} = \{e \in M : e \subseteq V',\ s/\alpha \leq |e| \leq s\}$. Below, we show that it is the case with probability $1 - O(n^{-\log n}) = 1 - e^{-\Omega(\log^2 n)}$.

We first show that with high probability, each edge e of size $|e| \in [s/\alpha, s]$ is contained in at least $\log^2 n/2$ sample S_i's. We use X_e to denote the number of samples containing e, then we have

$$\mathbb{E}(X_e) = n^\alpha \log^2 n \cdot n^{-\frac{\alpha|e|}{s}} \geq n^\alpha \log^2 n \cdot n^{-\alpha} = \log^2 n.$$

By Chernoff bound, we have

$$P(X_e \leq \log^2 n/2) \leq n^{-\log n/8}.$$

As there are at most $\alpha n/s$ edges of size between s/α and s, by a union bound, we have

$$P(\exists e \in M \text{ s.t. } |e| \in [s/\alpha, s], X_e \leq \log^2 n/2) \leq \frac{\alpha n}{s} n^{-\log n/8} \leq n^{-(\log n/8-1)}.$$

Subsequently,

$$P(\forall e \in M \text{ s.t. } |e| \in [s/\alpha, s], X_e \geq \log^2 n/2) \geq 1 - n^{-(\log n/8-1)}.$$

From now on we condition on the event that all edges e whose size is between s/α and s are included in at least $\log^2 n/2$ samples. We show below that given $e \subseteq S_i$ for a sample S_i, the probability that there exists another edge of size at least s in S_i is upper bounded by $n^{-1/n}$. Recall that M is the hidden matching we want to learn. We abuse notation and use $e' \in M \cap S_i$ to denote that an edge e' is both in the matching M and included in the sample S_i. We have

$$\begin{aligned}
\mathbb{P}(\exists e' \in M \cap S_i, e' \neq e \mid e \subseteq S_i) &\leq \sum_{e' \in M, e' \neq e} \mathbb{P}(e' \subseteq S_i \mid e \subseteq S_i) && (3.1)\\
&= \sum_{e' \in M, e' \neq e} \mathbb{P}(e' \subseteq S_i) && (3.2)\\
&\leq \frac{\alpha n}{s} \cdot (n^{-\frac{\alpha}{s}})^{\frac{s}{\alpha}} && (3.3)\\
&= \frac{\alpha n}{s} \cdot n^{-\frac{\alpha s}{\alpha s}} = \frac{\alpha}{s}.
\end{aligned}$$

where Eq.(3.1) uses union bound. Eq.(3.2) is due to the fact that M is a matching and thus $e \cap e' = \emptyset$

and vertices are sampled independently. Eq.(3.3) follows because the total number of remaining edges is upper bounded by $\alpha n/s$ and each edge is in S_i with probability at most $(n^{-\frac{\alpha}{s}})^{\frac{s}{\alpha}}$.

As each edge e with size between s/α and s is contained in at least $\log^2 n/2$ samples, we have that

$$\mathbb{P}(\forall S_i \text{ s.t. } e \subseteq S_i, \exists\, e' \in M \cap S_i, e' \neq e) \leq (\frac{s}{\alpha})^{-\log^2 n/2} \leq n^{-\log n/4},$$

where the last inequality follows because $s/\alpha \geq 2$.

By another union bound on all edges of size between s/α and s (at most n of them), we have that

$$\mathbb{P}(\exists e \text{ s.t. } |e| \in [s/\alpha, s]\,, \forall S_i \text{ s.t. } e \subseteq S_i, \exists e' \in M \cap S_i, e' \neq e) \leq n^{-(\log n/2-1)}.$$

We can thus conclude that with probability at least $1-O(n^{-\log n}) = 1-e^{-\Omega(\log^2 n)}$, for all $e \in M$ with size between s/α and s, there exists at least one sample S_i that contains e but no other remaining edges. By Lemma 3.4.1 and Lemma 3.4.2, e is returned by FINDEDGE(S_i, s) and e is added to the matching $\hat{M}$ that is returned by FINDDISJOINTEDGES(s, ρ, V').

Next we show that FINDDISJOINTEDGES is an r-adaptive algorithm with $n^\alpha \log^2 n + 2\alpha q n \log^2 n/s$ queries. We first observe that FINDDISJOINTEDGES makes only parallel calls to FINDEDGES (after verifying that $Q_M(S_i) = 1$). Therefore, since FINDEDGES runs in at most r rounds, we get that FINDDISJOINTEDGES runs in at most $r + 1$ rounds. To prove a bound on the number of queries, we first argue using a Chernoff bound that with probability $1 - e^{-\Omega(\log^2 n)}$, every edge e of size at least s/α is in at most $2\log^2 n$ samples S_i. Therefore there are at most $\frac{2\alpha n}{s} \log^2 n$ samples S_i such that $Q_M(S_i) = 1$. For every one of these samples, we call FINDEDGES(S_i, s), which makes q queries by assumption. Therefore, with high probability $1 - e^{-\Omega(\log^2 n)}$, FINDDISJOINTEDGES makes $O(n^\alpha \log^2 n + \alpha q \frac{2n}{s} \log^2 n)$ queries. □

The main algorithm for hypermatchings. The main algorithm first finds edges of size 1 by brute force with FINDSINGLETONS. It then iteratively learns the edges of size between s/α and s by calling FINDDISJOINTEDGES(s, α, V'), where V' is the set of vertices that are not in edges learned in previous iterations. At the end of each iteration the algorithm increases s.

Algorithm 10 FINDMATCHING$(\alpha,$ FINDEDGE$)$, learns a hypermatching.

```
Input: parameter $\alpha \geq 1$, subroutine FINDEDGE
1: $\hat{M} \leftarrow$ FINDSINGLETONS, $s \leftarrow 2\alpha$
2: while $s < n$ do
3:     $V' \leftarrow V \setminus \{v : v \in e$ for some $e \in \hat{M}\}$
4:     $\hat{M} \leftarrow \hat{M} \cup$ FINDDISJOINTEDGES$(s, \alpha, V',$ FINDEDGE$)$
5:     $s \leftarrow \lfloor \alpha s \rfloor + 1$
6: return $\hat{M}$.
```

Theorem 3.4.1. *Let M be a hypermatching, then* FINDMATCHING *learns M with high probability and*

- *with $O(n \log^5 n)$ queries in $O(\log^3 n)$ rounds, with $\alpha = \frac{1}{1-\frac{1}{2\log n}}$ and* FINDEDGEADAPTIVE*, or*
- *in $O(\log n)$ rounds and with $O(n^2 \log^3 n)$ queries, with $\alpha = 2$ and* FINDEDGEPARALLEL.

The following claim is a technical result we need to prove Theorem 3.4.1. We present its proof in Appendix B.1.

Claim 3.4.1. *For $0 \leq x < 1$ we have $\frac{1}{1-x} \geq e^x$.*

Proof of Theorem 3.4.1. FINDSINGLETONS learns, with probability 1, all the edges of size $s = 1$. For edges of size greater than 2, from Lemma 3.4.4, we know that each call of FINDDISJOINTEDGES with parameters s, α and V' and with the subroutine FINDEDGE fails to find all edges in V' of size between s/α and s with probability at most $e^{-\Omega(\log^2 n)}$. As there are at most n different edge sizes, FINDDISJOINTEDGES is called at most n times (we show below that it is actually called at most $\log n/(\log \alpha)$ times). Thus, the probability that at least one of the calls fails is upper bounded by

$$n \cdot e^{-\Omega(\log^2 n)} = e^{\log n - \Omega(\log^2 n))} = e^{-\Omega(\log^2 n)}.$$

We can thus conclude that the probability that all calls are successful is at least $1 - e^{-\Omega(\log^2 n)}$.

Next we show, that FINDMATCHING makes $O(\frac{\log n}{\log \alpha})$ calls to FINDDISJOINTEDGES. We use s_t to denote the value of s after the t^{th} call of FINDDISJOINTEDGES for $t = 1, 2, \ldots, T$ with T being the number of adaptive rounds of FINDMATCHING and set $s_0 = 2\alpha$. We show that the algorithm needs less than $O(\frac{\log n}{\log \alpha})$ rounds to go over all the possible edge sizes of the matching. In step 5 of FINDMATCHING, we update s_t as follows: $s_t \leftarrow \lfloor \alpha s_{t-1} \rfloor + 1$. Therefore

$$s_t \geq \alpha s_{t-1}, \tag{3.4}$$

and

$$s_t \geq (\alpha)^t s_0, \tag{3.5}$$

Since the maximum s we input to FINDDISJOINTEDGES is n, we have that $n \geq s_T \geq 2(\alpha)^{T+1}$ and subsequently

$$T \leq \frac{\log n - \log 2}{\log \alpha}.$$

Therefore, FINDMATCHING makes $O(\frac{\log n}{\log \alpha})$ calls to FINDDISJOINTEDGES. From Lemma 3.4.4, we know that with high probability $1-e^{-\Omega(\log^2 n)}$, FINDDISJOINTEDGES$(s, \alpha, V', \text{FINDEDGE})$ runs in $r + 1$ rounds and makes $O(n^\alpha \log^2 n + 2\alpha q n \log^2 n/s)$ queries when FINDEDGES(S, s) runs in r rounds and makes q queries. From Lemma 3.4.1 and Lemma 3.4.2 we know that for the subroutines FINDEDGEADAPTIVE and FINDEDGEPARALLEL we have $r = O(\log n), q = O(s \log n)$ and $r = 1, q = O(n)$ respectively. Therefore FINDDISJOINTEDGES runs in $O(\log n)$ rounds and makes $O(n^\rho \log^2 n + 2n \log^3 n)$ queries with FINDEDGEADAPTIVE and in 2 rounds $O(n^\alpha \log^2 n + 2n^2 \log^2 n)$ with FINDEDGEPARALLEL. We conclude that with high probability,

- FINDMATCHING runs in $O(\frac{\log^2 n}{\log \alpha})$ rounds and makes $O(n^\alpha \frac{\log^3 n}{\log \alpha} + 2\alpha n \frac{\log^4 n}{\log \alpha})$ with FIND-EDGEADAPTIVE,

- FINDMATCHING runs in $O(\frac{\log n}{\log \alpha})$ rounds and makes $O(n^\alpha \frac{\log^3 n}{\log \alpha} + 2n^2 \frac{\log^3 n}{\log \alpha})$ queries with

FINDEDGEPARALLEL.

Now we turn our attention to particular values of α.

- Suppose now that $\alpha = 1/(1 - \frac{1}{2\log n})$ and that we use FINDEDGEADAPTIVE. We get then $\alpha = 1 + o(1)$. Furthermore, by Claim 3.4.1, we have

$$\begin{aligned} \log \alpha &= \log \frac{1}{1 - \frac{1}{2\log n}} \\ &\geq \frac{1}{2\log n}. \end{aligned}$$

  Therefore we get that $\frac{1}{\log \alpha} = O(\log n)$. We also observe that for $n \geq 3$

$$\begin{aligned} \alpha &= 1 + \frac{\frac{1}{2\log n}}{1 - \frac{1}{2\log n}} \\ &\leq 1 + \frac{1}{\log n}. \end{aligned}$$

  Therefore $n^{\rho} \leq n \cdot n^{\frac{1}{\log n}} = O(n)$. We finally conclude that when $\rho = 1/(1 - \frac{1}{2\log n})$ and we use the subroutine FINDEDGEADAPTIVE, FINDMATCHING runs in $O(\log^2 n/\log \alpha) = O(\log^3 n)$ adaptive rounds and makes $O(n\log^4 n + n\log^5 n) = O(n\log^5 n)$ queries in total.

- Whit $\alpha = 2$ and FINDEDGEPARALLEL, we have $r = 1$ and we get that FINDMATCHING learns any hypermatching w.h.p. in $O(\log n)$ rounds and with at most $O(n^2 \log^3 n)$ queries.

□

3.5 Hardness of non-adaptive learning for hypermatchings

In this section, we show that no non-adaptive algorithm is capable of learning a hypermatching with polynomially many queries. This result is in sharp contrast to matchings (where edges are of size $d = 2$) for which there exists a non-adaptive learning algorithm with $O(n \log n)$ queries [143]. We present a construction of a family of hypermatchings for which there is no non-adaptive learning algorithm with poly(n) queries. Each hypermatching M_P in this family depends on a partition $P = (P_1, P_2, P_3)$ of the vertex set V into three parts P_1, P_2, P_3 where $|P_1| = n - (\sqrt{n} + 1)$, $|P_2| = 1$, and $|P_3| = \sqrt{n}$. M_P contains $(n - (\sqrt{n} + 1))/2$ edges of size 2 which form a perfect matching over P_1 and one edge of size $\sqrt{n}$ which contains all the vertices in P_3. The only vertex in P_2 is not contained in any edges in M_P.

The main idea of the construction is that, after one round of poly(n) queries, vertices in P_2 and P_3 are indistinguishable to the learning algorithm. However, the algorithm needs to distinguish vertices in P_3 from the vertex in P_2 to learn the edge P_3.

Theorem 3.5.1. *There is no non-adaptive algorithm with* poly(n) *query complexity that can learn every non-uniform hypermatching with probability* $\omega(1/\sqrt{n})$.

To argue that learning hypermatchings non-adaptively requires an exponential number of queries, we fix the number of vertices n, we construct a family of matchings M_P which depend on a partition P of the vertex set $[n]$ into 3 parts P_1, P_2, P_3 such that

- $|P_1| = n - (\sqrt{n} + 1)$.
- P_1 contains a perfect matching M_1 with edges of size 2.
- $|P_2| = 1$.
- P_3 is an edge of the matching s.t. $|P_3| = \sqrt{n}$.
- $M_P = M_1 \cup \{P_3\}$.

We denote all such partitions by $\mathcal{P}_3$. For a partition $P \in \mathcal{P}_3$, multiple matchings are possible, depending on the perfect matching M_1. We use M_P to denote a random matching from all the possible matchings satisfying the properties above. The main idea of the construction is that after one round of queries, elements in P_2 and P_3 are indistinguishable to any algorithm with polynomial queries. However, a learning algorithm needs to distinguish elements in P_3 from the element in P_2 to learn the edge P_3.

Before presenting the proof, we formalize what it means for a set of elements to be indistinguishable. Since the next definition is also used in the next subsection, we consider an arbitrary family $\mathcal{P}_R$ of partitions $P = (P_0, \dots, P_R)$. Given a partition $(P_0, \dots, P_R)$, we denote by $P_{r:}$ the union of parts P_i such that $i \geq r$, i.e. $P_{r:} := \cup_{i=r}^{R} P_i$. Informally, we say that queries $Q_{M_P}(S)$ are independent of the partition of $P_{r:}$ if the values $Q_{M_P}(S)$ of these queries do not contain information about which elements are in P_r, or $P_{r+1}, \dots,$ or P_R.

Definition 3.5.1. *Given a family of partitions $\mathcal{P}_R$ and a partition $P = (P_0, \dots, P_R) \in \mathcal{P}_R$, let P' be a partition chosen uniformly at random from $\{(P'_0, \dots, P'_R) \in \mathcal{P}_3 \ : \ P'_0 = P_0, \dots P'_{r-1} = P_{r-1}\}$. Let $M_{P'}$ be a matching on P' such that M_P and $M_{P'}$ are equal on $P_r := \cup_{i=r}^{R} P_i$. A query $Q_{M_P}(S)$ is independent from $P_r := \cup_{i=r}^{R} P_i$ if $Q_{M_P}(S) = Q_{M_{P'}}(S)$ with high probability over P'.*

For example, in the partition (P_1, P_2, P_3) above , any query $Q_{M_P}(S)$ such that S contains an edge from M_1 is independent of the partition of $P_2 \cup P_3$ since this query will always answer 1.

The main lemma (Lemma 3.5.1) shows that, with high probability, for all queries $S \in \mathcal{S}$, the answer $Q_{M_P}(S)$ is independent of which $\sqrt{n}$ vertices in $P_2 \cup P_3$ constitute the edge P_3. The analysis of this lemma consists of two cases. If the query S is small ($|S| < (n + \sqrt{n} + 1)/2$), then we show that S contains less than $\sqrt{n}$ vertices from $P_2 \cup P_3$, w.h.p. over the randomization of P, by the Chernoff bound. Therefore S does not contain P_3 and $Q_{M_P}(S)$ is independent of the partition of $P_2 \cup P_3$ into P_2 and P_3. If S is large ($|S| \geq (n + \sqrt{n} + 1)/2$), then S contains an edge of size two from P_1. Thus $Q_{M_P}(S) = 1$ and $Q_{M_P}(S)$ is independent of the partition of $P_2 \cup P_3$ into P_2 and P_3.

In Lemma 3.5.2, we argue that if all queries $Q_{M_P}(S)$ of $\mathcal{A}$ are independent of the partition of $P_2 \cup P_3$ into P_2 and P_3, then, with high probability, the matching returned by this algorithm is not equal to M_P. By the probabilistic method, there is a matching M_P with $P \in \mathcal{P}_3$ that $\mathcal{A}$ does not successfully learn with probability $1 - O(1/\sqrt{n})$.

Analysis of the construction. We consider a non-adaptive algorithm $\mathcal{A}$, a uniformly random partition $P = (P_1, P_2, P_3)$ in $\mathcal{P}_3$, and a matching M_P. We argue that with high probability over both the randomization of P and the decisions of A, the matching returned by $\mathcal{A}$ is not equal to

M_P. The analysis consists of two main parts.

The first part argues that, with high probability, for all queries S made by a non-adaptive algorithm, $Q_{M_P}(S)$ is independent of the partition of $P_2 \cup P_3$.

Lemma 3.5.1. *Let P be a uniformly random partition from $\mathcal{P}_3$. For any collection $\mathcal{S}$ of* $\mathrm{poly}(n)$ *non-adaptive queries, with high probability $1 - e^{-\Omega(n)}$, for all $S \in \mathcal{S}$, S is independent of the partition of $P_2 \cup P_3$.*

Proof. Consider any set $S \in \mathcal{S}$ which is independent of the randomization of P. There are two cases depending on the size of S.

1. $|S| \leq \frac{n+\sqrt{n}+1}{2}$. In this case, if S does not contain at least $\sqrt{n}$ vertices from $P_2 \cup P_3$, then for any other partition P' such that $P'_1 = P_1$, we have $Q_{M_P}(S) = Q_{M_{P'}}(S)$. In fact, both $Q_{M_{P'}}(S)$ and $Q_{M_P}(S)$ will be equal 1 if and only if S contains an edge from M_1. Next we show that, with high probability, S will contain fewer than $\sqrt{n}$ vertices from $P_2 \cup P_3$. Since $P_2 \cup P_3$ is a uniformly random set of size $\sqrt{n} + 1$, by the Chernoff bound, $|S \cap (P_2 \cup P_3)| < \sqrt{n} + 1$ with probability $1 - e^{-\Omega(\sqrt{n})}$.

   We therefore get that $Q_{M_P}(S) = Q_{M_{P'}}(S)$ for any partition $P' = (P_1, P'_2, P'_3)$ and query $|S| \leq \frac{n+\sqrt{n}+1}{2}$ with probability $1 - e^{-\Omega(\sqrt{n}))}$.

2. $|S| > \frac{n+\sqrt{n}+1}{2}$. In this case, the set S contains at least one matched edge from P_1. In fact, the number of edges in M_1 plus the size of $P_2 \cup P_3$ is

$$\frac{|P_1|}{2} + |P_2| + |P_3| = \frac{n - (\sqrt{n} + 1)}{2} + \sqrt{n} + 1 = \frac{n + \sqrt{n} + 1}{2} < |S|.$$

   S must therefore contain at least one edge from M_1, and we get that $Q_{M_P}(S) = Q^{P'}(S)$ for any partition $P' = (P_1, P'_2, P'_3)$ and query $|S| > \frac{n+\sqrt{n}+1}{2}$ with probability 1.

By combining the two cases $|S| \leq \frac{n+\sqrt{n}+1}{2}$ and $|S| > \frac{n+\sqrt{n}+1}{2}$, we get that with probability $1 - e^{-\Omega(\sqrt{n}))}$, $Q_{M_P}(S)$ is independent of the partition of $P_2 \cup P_3$, and by a union bound, this holds for any collection of $poly(n)$ sets S. $\square$

The second part of the analysis argues that if all queries $Q_{M_P}(S)$ of an algorithm are independent of the partition of $P_2 \cup P_3$, then, with high probability, the matching returned by this algorithm is not equal to M_P.

Lemma 3.5.2. *Let P be a uniformly random partition in $\mathcal{P}_3$ and M_P a random matching over P. Consider an algorithm $\mathcal{A}$ such that all the queries $Q_{M_P}(S)$ made by $\mathcal{A}$ are independent of the partition of $P_2 \cup P_3$. Then, the (possibly randomized) matching M returned by $\mathcal{A}$ is, with probability $1 - O(1/\sqrt{n})$, not equal to M_P.*

Proof. Consider an algorithm $\mathcal{A}$ such that all queries $Q_{M_P}(S)$ of $\mathcal{A}$ are independent of the partition of $P_2 \cup P_3$. Thus, the matching M returned by $\mathcal{A}$ is conditionally independent of the randomization of the partition P given P_1.

For the algorithm $\mathcal{A}$ to return M_P, it needs to learn the edge P_3. We distinguish two cases.

- $\mathcal{A}$ does not return any edge that is included in $P_2 \cup P_3$. In this case, with probability 1, M is not equal to M_P.
- $\mathcal{A}$ returns an edge that is included in $P_2 \cup P_3$. We know from the previous lemma that with probability $1-e^{-\Omega(\sqrt{n})}$, all queries are independent from $P_2 \cup P_3$. Therefore, the edge returned by $\mathcal{A}$ is also independent of the partition $P_2 \cup P_3$. The probability that this edge is equal to P_3 is less than $1/\sqrt{n+1}$. Therefore, with probability $(1-e^{-\Omega(\sqrt{n})})(1-1/\sqrt{n+1}) = 1-O(1/\sqrt{n})$, the returned matching M is not equal to M_P.

$\square$

By combining Lemma 3.5.1 and Lemma 3.5.2, we get the hardness result for non-adaptive algorithms.

Proof of Theorem 3.5.1. Consider a uniformly random partition $P \in \mathcal{P}_3$, a matching M_P and an algorithm $\mathcal{A}$ which queries M_P . By Lemma 3.5.1, after one round of queries, with probability $1 - e^{-\Omega(\sqrt{n})}$ over both the randomization of P and of the algorithm, all the queries $Q_{M_P}(S)$ made by $\mathcal{A}$ are independent of the partition of $P_2 \cup P_3$. By Lemma 3.5.2, this implies that, with probability

$1 - O(1/\sqrt{n})$, the matching returned by $\mathcal{A}$ is not a equal to M_P. By the probabilistic method, this implies that there exists a partition $P \in \mathcal{P}_3$ for which, with probability $1 - O(1/\sqrt{n})$, $\mathcal{A}$ does not return M_P after one round of queries. $\square$

3.6 Hardness of adaptive learning for hypermatchings

In this section, we show that no adaptive algorithm can learn hypermatchings in $o(\log \log n)$ rounds with polynomially many queries. The main technical challenge is to generalize the construction and the analysis of the hard family of hypergraphs for non-adaptive learning algorithms to a construction and an analysis which holds over $\Omega(\log \log n)$ rounds. In this construction, each hypermatching M_P depends on a partition $P = (P_0, P_1, \ldots, P_R)$ of the vertex set V into $R+1$ parts. For each $i \in \{0, \ldots R\}$, M_P contains $|P_i|/d_i$ edges of size d_i which form a perfect matching over P_i. We present the detailed construction later this section.

The main idea of the construction is that after i rounds of queries, vertices in $\cup_{j=i+1}^{R} P_i$ are indistinguishable to any learning algorithm. However, the algorithm needs to distinguish vertices in P_j from vertices in $P_{j'}$, for all $j \neq j'$, to learn M_P. Informally, since the edges in $P_{j'}$ have larger size than edges in P_j for $j' > j$, an algorithm can learn only one part P_j at each round of queries, which is the part with edges of minimum size among all parts that have not yet been learned in previous rounds.

Theorem 3.6.1. *There is no* $(\log \log n - 3)$*-adaptive algorithm with* $poly(n)$ *query complexity which can learn every non-uniform hypermatching with probability* $e^{-o(\log^2 n)}$.

We present a proof sketch for Theorem 3.6.1 before going into the full analysis. Let $\mathcal{P}_R$ be the set of all partitions $(P_0, \ldots, P_R)$ of our construction, and let P be a uniformly random partition from $\mathcal{P}_R$, and $\mathcal{A}$ be an algorithm with $\text{poly}(n)$ queries. In Lemma 3.6.5, we show that if vertices in $P_i, P_{i+1}, \ldots, P_R$ are indistinguishable to $\mathcal{A}$ at the beginning of round i of queries, then vertices in $P_{i+1}, \ldots, P_R$ are indistinguishable at the end of round i.

The proof of Lemma 3.6.5 consists of two parts. First, Lemma 3.6.4 shows that if S is small ($|S| \leq (1 - 1/d_i) \sum_{j \geq i} |P_j|$) and is independent of partition of $\cup_{j=i}^{R} P_i$ into $P_i, \ldots, P_R$, then w.h.p., for every $j \geq i+1$, S does not contain any edge contained in P_j. Second, Lemma 3.6.3 shows that if a query S has large size ($|S| \geq (1-1/d_i)\sum_{j\geq i}|P_j|$) and is independent of partition of $\cup_{j=i}^{R} P_i$ into $P_i, \dots, P_R$, then w.h.p. S contains at least one edge from the matching on P_i which implies $Q_{M_P}(S) = 1$. Proving Lemma 3.6.3 is quite involved. Because the edges of a matching are not independent from each other, a Chernoff bound analysis is not enough to show that the probability that a query does not contain any edge from P_i small. We therefore provide an alternative method to bound this probability (Lemma 3.6.2). At the end, in both cases, we show that $Q_{M_P}(S)$ is independent of the partition of $\cup_{j=i+1}^{R} P_i$ into $P_{i+1}, \dots, P_R$. Finally, we conclude the proof of Theorem 3.6.1 similarly as for Theorem 3.5.1.

The full analysis. As we highlighted, our idea is to construct a partition $P_0, \dots, P_i, \dots P_R$, such that every set P_i is a perfect matching of edges of size d_i, and therefore has $|P_i|/d_i$ edges. We want to choose the size $|P_i|$ and d_i (d_i increasing) such that with probability $1 - e^{-poly(n)}$, after i rounds of any adaptive algorithm that asks polynomial number of queries, the partition $P_i \cup P_{i+1} \dots P_R$ is indistinguishable. We choose the edge sizes and partition sizes as follows:

- $d_0 = 3$.
- $d_{i+1} = 3\log^2 n \cdot d_i^2$.
- $|P_i| = 3\log^2 n \cdot d_i^2$.
- $|P_{i+1}| = 3\log^2 n \cdot d_{i+1}^2 = 3\log^2 n \cdot (3\log^2 n \cdot d_i^2)^2 = 3\log^2 n \cdot |P_i|^2$.

We recall that $\mathcal{P}_R$ is the set of partitions $P = (P_0, \dots, P_R)$ satisfying the conditions above. For a fixed i, let $k_i = \frac{|P_i|}{d_i}$, and let $e_1^i, \dots, e_j^i, \dots, e_{k_i}^i$ be the random matching on P_i. Finally, let $n_i = \sum\limits_{l\geq i} |P_l|$. We first show in the next claim that the construction has $\Theta(\log\log n)$ partitions.

Claim 3.6.1. *The construction has $R+1 = \Theta(\log\log n)$ partitions.*

Proof. We first show that the construction has $R+1 \geq \log\log n$ partitions. By induction we have that

$$|P_i| = (3\log^2 n)^{2^i-1}|P_0|^{2^i}.$$

Therefore

$$\begin{aligned} n &\leq \sum_{i=0}^{R} |P_i| \\ &\leq (R+1)|P_R| \\ &\leq (R+1)(3\log^2 n)^{2^R-1} |3\log^n d_0^2|^{2^R} \\ &= (R+1)9^{2^R}(3\log^2 n)^{2^{R+1}-1} \\ &= 9^{2^{R+1}}(3\log^2 n)^{2^{R+1}}. \end{aligned}$$

This implies that

$$2^{R+1}\log(27\log^2 n) \geq \log n,$$

and

$$R+1 \geq \frac{\log\log n}{\log 2} - \frac{\log\left(\log(27\log^2 n)\right)}{\log 2} \geq \log\log n.$$

Next we show that $R = O(\log\log n)$. We know that $|P_R| \leq n$, which implies that $(3\log^2 n)^{2^R-1} \leq n$ and

$$2^R - 1 \leq \frac{\log n}{\log\log^6 n}.$$

Therefore,

$$R = O(\log \frac{\log n}{\log\log^6 n}) = O(\log\log n).$$

□

Suppose now that at the beginning of round i, we learned $P_0 \cup \ldots \cup P_{i-1}$ but $P_i \cup P_{i+1} \ldots P_R$ is indistinguishable. Consider a query S of the i-th round. We show that if the query S is big, then with high probability S contains an edge from P_i, and if S is small, then S does not contain any edge from the high partition $P_{i+1}, P_{i+2}, \ldots P_R$. In both cases, querying S gives an answer that is independent to $P_{i+1}, P_{i+2}, \ldots P_R$ with high probability over all the polynomial number of queries.

Therefore, after the i-th round, $P_{i+1}, P_{i+2}, \dots$ is indistinguishable.

The outline of the proof is as follows. Claim 3.6.2 is a technical result we need to bound the binomial coefficients in our proofs. Lemma 3.6.1 gives the expected size of the intersection of a query with a partition and presents a concentration result on the intersection when the queries are large. Lemma 3.6.2 (proof in Appendix B.1) shows that for the purpose of bounding the probability that no edge of a partition is included in S, we can drop the constraint the edges are disjoint and think of them as being randomly and uniformly sampled (with replacement) from the partition. Lemma 3.6.3 shows that $|S| \geq (1 - \frac{1}{d_i})n_i$, then with probability $1 - e^{-\Omega(\log^2 n)}$, S contains at least one edge from the matching on P_i. If $|S| \leq (1 - \frac{1}{d_i})n_i$, then Lemma 3.6.4 shows with probability $1 - e^{-\Omega(\log^2 n)}$, for every $l \geq i+1$, S does not contain any edge from the matching on P_l. Combining the two lemmas (Lemma 3.6.5) shows that if P is a uniformly random partition in P_R, and if all the queries made by an algorithm $\mathcal{A}$ in the previous i rounds are independent of the partition of $P_i, \dots, P_R$, then any collection of $poly(n)$ non-adaptive queries at round $i+1$, independent of $P_{i+1}, \dots, P_R$ with probability $1 - e^{-\Omega(\log^2 n)}$.

The following claim is a technical result we need to proceed with the hardness proof. Its proof appears in Appendix B.1.

Claim 3.6.2. *If $k < \sqrt{n}$, then*

$$\frac{n^k}{4(k!)} \leq \binom{n}{k} \leq \frac{n^k}{k!}$$

Lemma 3.6.1. *Let S be an arbitrary subset of $P_i \cup P_{i+1} \cup \dots P_R$. After i rounds, for $l \geq i$, the expected size of the intersection $|S \cap P_l|$ is*

$$\mathbb{E}[|S \cap P_l|] = s_l = \frac{|S||P_l|}{n_i},$$

where the expectation is over the partitions of $P_i \cup P_{i+1} \cup \dots P_r$ into $P_i, P_{i+1}, \dots, P_r$. Moreover, for any polynomial set of queries $\mathcal{S}$ such that $|S| \geq (1 - \frac{1}{d_i})n_i$ for $S \in \mathcal{S}$, then with probability $1 - e^{-\Omega(\log^2 n)}$, for all $l \geq i$ and for all $S \in \mathcal{S}$, we have $|S \cap P_l| \geq s_l(1 - \frac{1}{d_i})$.

Proof of Lemma 3.6.1. Every node in S will be in P_l with probability $|P_l|/n_i$, therefore by linearity of expectation

$$\mathbb{E}[|S \cap P_l|] = s_l = \frac{|S||P_l|}{n_i}$$

Suppose now that $|S| \geq (1 - \frac{1}{d_i})n_i$. By Chernoff bound, we have

$$\begin{aligned}\mathbb{P}\big(|S \cap P_l| \leq s_l(1 - \frac{1}{d_i})\big) &\leq e^{-\frac{s_l}{2d_i^2}} \\ &= e^{-\frac{|S|}{n_i}\frac{|P_l|}{2d_i^2}} \\ &\leq e^{-(1-1/d_i)\frac{|P_l|}{2d_i^2}} \\ &\leq e^{-\frac{|P_l|}{3d_i^2}} \\ &\leq e^{-\log^2 n} \\ &\leq \frac{1}{n^{\log n}}.\end{aligned}$$

In the fourth inequality, we used $1 - 1/d_i \leq 2/3$. By a union bound the probability that there exists a partition P_l with $l \geq i$ such that $|S \cap P_l| \leq s_l(1 - \frac{1}{d_i})$ is $O(\log\log n / n^{\log n}) = e^{-\Omega(\log^2 n)}$. Another application of the union bound on the polynomially many queries shows that with probability at least $1 - e^{-\Omega(\log^2 n)}$, for all queries of size greater than $(1 - 1/d_i)n$ and for all partitions greater than i, we have $|S \cap P_l| \geq s_l(1 - \frac{1}{d_i})$. □

Lemma 3.6.2. *Fix an iteration i, and a query S. Let $e'_1, \dots, e'_{k_i}$ be edges of size d_i, independently and uniformly sampled from P_i. Let $e^i_1, \dots, e^i_{k_i}$ be a random matching on P_i. We have*

$$\mathbb{P}(\forall j \in \{1, \dots, k_i\},\ e^i_j \nsubseteq S) \leq \mathbb{P}(\forall j \in \{1, \dots, k_i\},\ e'_j \nsubseteq S) = (1 - \frac{\binom{|S \cap P_i|}{d_i}}{\binom{|P_i|}{d_i}})^{k_i}$$

Lemma 3.6.3. *Consider a query S by an algorithm at round i such that all the queries from previous rounds are independent of $P_i, \dots, P_R$. Let S be such that $|S| \geq (1 - \frac{1}{d_i})n_i$, then with probability $1 - e^{-\Omega(\log^2 n)}$, S contains at least one edge from the matching on P_i*

Proof.

$$\mathbb{P}(\exists j \in \{1,\ldots,k_i\},\ e^i_j \subseteq S) = 1 - \mathbb{P}(\forall j \in \{1,\ldots,k_i\},\ e^i_j \not\subseteq S) \tag{3.6}$$

$$\geq 1 - \Big(1 - \frac{\binom{|S\cap P_i|}{d_i}}{\binom{|P_i|}{d_i}}\Big)^{k_i}. \tag{3.7}$$

where the inequality comes from Lemma 3.6.2. Since $|S| \geq (1 - \frac{1}{d_i})n_i$, we have with high probability that $|S \cap P_i| \geq (1 - \frac{1}{d_i})^2|P_i| \geq d_i^2$. Conditioning on this event, and since we already have $d_i \leq \sqrt{|P_i|}$, we get by Claim 3.6.2

$$\begin{aligned}\frac{\binom{|S\cap P_i|}{d_i}}{\binom{|P_i|}{d_i}} &\geq \frac{|S\cap P_i|^{d_i}}{4(d_i)!}\frac{(d_i)!}{|P_i|^{d_i}} \\ &\geq \frac{1}{4}(1-\frac{1}{d_i})^{2d_i} \\ &\geq \frac{1}{4}e^{-\frac{1/d_i}{1-1/d_i}2d_i} \\ &\geq \frac{1}{4}e^{-\frac{2}{1-1/d_i}} \\ &\geq \frac{1}{4}e^{-4}\end{aligned}$$

Since $k_i = \frac{|P_i|}{d_i} = 3\log^2 n \cdot d_i$, we get that

$$\frac{\binom{|S\cap P_i|}{d_i}}{\binom{|P_i|}{d_i}} \geq \frac{1}{4}e^{-4} \geq \frac{1}{3d_i} = \frac{\log^2 n}{k_i}.$$

The probability (3.7) becomes

$$\mathbb{P}(\exists j \in \{1,\dots,k_i\},\ e^i_j \subseteq S) \geq 1 - (1 - \frac{\binom{|S\cap P_i|}{d_i}}{\binom{|P_i|}{d_i}})^{k_i} \tag{3.8}$$

$$\geq 1 - (1 - \frac{\log^2 n}{k_i})^{k_i} \tag{3.9}$$

$$\geq 1 - \frac{1}{n^{\log n}} \tag{3.10}$$

$$= 1 - e^{-\Omega(\log^2 n)}. \tag{3.11}$$

□

Lemma 3.6.4. *Consider a query S by an algorithm at round i such that all the queries from previous rounds are independent of $P_i, \dots, P_R$. Let S be such that $|S| \leq (1 - \frac{1}{d_i})n_i$, then with probability $1 - e^{-\Omega(\log^2 n)}$, for every $l \geq i+1$, S does not contain any edge from the matching on P_l.*

Proof. Let's fix a partition P_l with $l \geq i+1$. Let $e_1, \dots, e_j, \dots e_{k_l}$ be the random matching on P_l. We have for $j = 1, \dots, k_l$,

$$\mathbb{E}\big[|e_j \cap S|\big] = \frac{|S||e_j|}{n_i} \leq (1 - \frac{1}{d_i})d_l = d_l - \frac{d_l}{d_i}.$$

Going forward, we present an upper bound on $\mathbb{P}(e_j \subset S)$. We can assume without loss of generality that $|S| = (1 - \frac{1}{d_i})n_i$. In fact, for any $j = 1, \dots, k_l$, if $S \subseteq S'$ then we have $\mathbb{P}(e_j \subseteq S) \leq \mathbb{P}(e_j \subseteq S')$.

The probability that $e_j \subset S$ can be expressed as

$$\mathbb{P}(e_j \subset S) = \mathbb{P}(|e_j \cap S| \geq d_l)$$
$$\leq \mathbb{P}\Big(|e_j \cap S| \geq (1+\delta)E\big[|e_j \cap S|\big]\Big),$$

where $\delta = \frac{1}{d_i - 1}$. By Chernoff bound we get

$$\begin{aligned}
\mathbb{P}(e_j \subset S) &\leq 2e^{-\frac{1}{3}\delta^2 E\left[|e_j \cap S|\right]} \\
&= 2e^{-\frac{1}{3}\frac{1}{(d_i-1)^2}(1-\frac{1}{d_i})d_l} \\
&= 2e^{-\frac{1}{3}\frac{1}{(d_i-1)d_i}d_l} \\
&\leq 2e^{-\log^2 n} \\
&\leq \frac{2}{n^{\log n}},
\end{aligned}$$

where the second to last inequality is due to $d_l \geq d_{i+1} = 3\log^2 n \cdot d_i^2$. Therefore by a union bound on the edges of the matching in P_l we get that

$$\begin{aligned}
\mathbb{P}(\exists\, j = 1, \ldots k_l,\;\; e_j \subset S) &\leq \frac{2k_l}{n^{\log n}} \\
&\leq \frac{1}{n^{\log n - 1}}.
\end{aligned}$$

Finally, another union bound on all the partition $l \geq i+1$ yields that the probability that there exists a partition P_l such that an edge of P_l is included in S is less than $O(\log\log n / n^{\log n - 1}) = 1 - e^{-\Omega(\log^2 n)}$. □

Lemma 3.6.5. *If vertices in $P_i, P_{i+1}, \ldots, P_R$ are indistinguishable to $\mathcal{A}$ at the beginning of round i of queries, then vertices in $P_{i+1}, \ldots, P_R$ are indistinguishable at the end of round i with probability $1 - e^{-\Omega(\log^2 n)}$.*

Proof. Consider a query S by an algorithm at round i such that all the queries from previous rounds are independent of the parition $P_i, \ldots, P_R$. Lemma 3.6.4 shows that if S is small ($|S| \leq (1-1/d_i)\sum_{j\geq i}|P_j|$) and is independent of partition of $\cup_{j=i}^R P_i$ into $P_i, \ldots, P_R$, then with probability $1 - e^{-\Omega(\log^2 n)}$, for every $j \geq i+1$, S does not contain any edge contained in P_j. On the other hand, Lemma 3.6.3 shows that if a query S has large size ($|S| \geq (1-1/d_i)\sum_{j\geq i}|P_j|$) and is independent of partition of $\cup_{j=i}^R P_i$ into $P_i, \ldots, P_R$, then with probability $1 - e^{-\Omega(\log^2 n)}$, S contains at least one edge from the matching on P_i which implies $Q_{M_P}(S) = 1$. In both cases, $Q_{M_P}(S)$ is independent of the partition of $\cup_{j=i+1}^{R} P_i$ into $P_{i+1}, \ldots, P_R$ with probability $1 - e^{-\Omega(\log^2 n)}$. By a union bound, this holds for $poly(n)$ queries at round i. $\square$

We are now ready to prove the main theorem of this section.

Proof of Theorem 3.6.1. Consider a uniformly random partition $P = (P_0, \ldots, P_i, \ldots P_R)$, a matching M_P and an algorithm $\mathcal{A}$ which queries M_P in $\log\log n - 3$ rounds. By Lemma 3.6.5, after i rounds of queries, with probability $1 - e^{-\Omega(\log^2 n)}$ over both the randomization of P and of the algorithm, all the queries $Q_{M_P}(S)$ made by $\mathcal{A}$ are independent of the partition of $P_i \cup \ldots \cup P_R$. Therefore, and since $R \geq \log\log n - 1$ by Claim 3.6.1, we get that after $\log\log n - 3$ round of queries, with probability $1 - e^{-\Omega(\log^2 n)}$ over both the randomization of P and the algorithm $\mathcal{A}$, all the queries $Q_{M_P}(S)$ made by $\mathcal{A}$ are independent of the partition of $P_{R-1} \cup P_R$. We now distinguish two cases:

- $\mathcal{A}$ does not a return any edge that is included in $P_{R-1} \cup P_R$.
- $\mathcal{A}$ returns a set of edges that is included in $P_{R-1} \cup P_R$. In this case, we know that with probability $1 - e^{-\Omega(\log^2 n)}$ all the queries $Q_{M_P}(S)$ made by $\mathcal{A}$ are independent of the partition of $P_{R-1} \cup P_R$ into P_{R-1} and P_R. The edges that are returned by $\mathcal{A}$ and included in $P_{R-1} \cup P_R$ are therefore also independent from the partition of $P_{R-1} \cup P_R$ into P_{R-1} and P_R with probability $1 - e^{-\Omega(\log^2 n)}$. To fully learn M_P, $\mathcal{A}$ needs to make the distinction between points in P_{R-1} and points in P_R, but there are $\binom{|P_R|+|P_{R-1}|}{|P_{R-1}|}$ ways of partitioning $P_{R-1} \cup P_R$ into P_{R-1} and P_R. Therefore the probability that $\mathcal{A}$ correctly learns M_P is less than

$$(1 - e^{-\Omega(\log^2 n)}) \frac{1}{\binom{|P_R|+|P_{R-1}|}{|P_{R-1}|}}.$$

In the rest of the proof, we show that $1/\binom{|P_R|+|P_{R-1}|}{|P_{R-1}|} = e^{-\Omega(\log^2 n)}$. This implies that the probability that $\mathcal{A}$ learns M_P correctly is less than $(1 - e^{-\Omega(\log^2 n)})e^{-\Omega(\log^2 n)} = e^{-\Omega(\log^2 n)}$.

We know that $n \leq \sum_{i=0}^{R} |P_i| \leq (R + 1)|P_R|$. Therefore $|P_R| \geq n/(R + 1) = \Omega(n/\log\log n)$.

By Claim 3.6.2, we get that

$$\begin{aligned}
\frac{1}{\binom{|P_R|+|P_{R-1}|}{|P_{R-1}|}} &\leq \frac{4(|P_{R-1}|)!}{(|P_R|+|P_{R-1}|)^{|P_{R-1}|}} \\
&\leq \left(\frac{|P_{R-1}|}{|P_R|+|P_{R-1}|}\right)^{|P_{R-1}|} \\
&\leq \left(\frac{1}{3\log^2 n|P_{R-1}|}\right)^{|P_{R-1}|} \\
&\leq \left(\frac{1}{3\log^2 n|P_{R-1}|}\right)^{3\log^2 n} \\
&\leq \left(\frac{1}{3\log^2 n|P_{R-1}|}\right)^{3\log^2 n} \\
&= e^{-\Omega(\log^2 n)}.
\end{aligned}$$

Therefore, with probability $1 - e^{-\Omega(\log^2 n)}$, the matching returned by $\mathcal{A}$ is not equal to M_P. □

3.7 Conclusion

In this chapter, we studied the problem of learning a hypergraph via edge detecting queries. In this problem, a learner queries subsets of vertices of a hidden hypergraph and observes whether these subsets contain an edge or not. We identify hypermatchings as a family of hypergraphs that can be learned without suffering from a query complexity that grows exponentially in the size of the edges. We show that hypermatchings with n vertices are learnable with poly(n) queries: we give an $O(\log^3 n)$-round algorithm with $O(n \log^5 n)$ queries to learn hypermatchings. We complement this upper bound by showing that there are no algorithms with poly(n) queries that learn hypermatchings in $o(\log\log n)$ adaptive rounds. In [19], we also deal with hypergraphs with maximum degree Δ and edge size ratio ρ, we give a non-adaptive algorithm with $O((2n)^{\rho\Delta+1} \log^2 n)$ queries. To the best of our knowledge, these are the first algorithms with poly(n, m) query complexity for learning non-trivial families of hypergraphs that have a super-constant number of edges of super-constant size.

Chapter 4: Distributed Hypergraph Matching

After learning hypegraphs, we now turn and focus on the problem of computing maximum matchings in hypergraphs. This chapter is based on the article [20] written in collaboration with Clifford Stein, that is published in the *International Workshop on Approximation and Online Algorithms*, 2020.

4.1 Introduction

As we touched upon in Chapter 1, the size of inputs in optimization problems is continuously increasing, and massive graphs are becoming more ubiquitous. This increase strengthens the need for scalable parallel and distributed algorithms that solve graph problems. In recent years, we have seen progress in many graph problems (e.g., spanning trees, connectivity, shortest paths [118, 153]) and, most relevant to this thesis, matchings [115, 154]. A natural generalization of matchings in graphs is to matchings in hypergraphs. Hypergraph matching is an important problem with many applications such as capital budgeting, crew scheduling, facility location, scheduling airline flights [140], forming a coalition structure in multi-agent systems [141] and determining the winners in combinatorial auctions [142] (see [155] for a partial survey). Although matching problems in graphs are one of the most well-studied problems in algorithms and optimization, the NP-hard problem of finding a maximum matching in a hypergraph is not as well understood.

In this chapter, we are interested in the problem of finding matchings in very large hypergraphs, large enough that we cannot solve the problem on one computer. We develop, analyze and experimentally evaluate three parallel algorithms for hypergraph matchings in the MPC model. Two of the algorithms are generalizations of parallel algorithms for matchings in graphs. The third algorithm develops new machinery which we call a hyper-edge degree constrained subgraph

(HEDCS), generalizing the notion of an edge-degree constrained subgraph (EDCS). The EDCS has been recently used in parallel and dynamic algorithms for graph matching problems [7, 156, 114]. We will show a range of algorithm tradeoffs between approximation ratio, rounds, memory, and computation, evaluated both as worst-case bounds, and via computational experiments.

Recall that a hypergraph H is a pair $H = (V, E)$ where V is the set of vertices and E is the set of hyperedges. A *hyperedge* $e \in E$ is a nonempty subset of the vertices. The cardinality of a hyperedge is the number of vertices it contains. When every hyperedge has the same cardinality d, the hypergraph is said to be d*-uniform*. A graph is therefore a 2-uniform hypergraph. A hypergraph is *linear* if the intersection of any two hyperedges has at most one vertex. A *hypergraph matching* is a subset of the hyperedges $M \subseteq E$ such that every vertex is covered at most once, i.e., the hyperedges are mutually disjoint. This notion generalizes matchings in graphs. The cardinality of a matching is the number of hyperedges it contains. A matching is called maximum if it has the largest cardinality of all possible matchings, and maximal if it is not contained in any other matching. In the d-Uniform Hypergraph Matching Problem (also referred to as Set Packing or d-Set Packing), a d-uniform hypergraph is given, and one needs to find the maximum cardinality matching.

We adopt the most restrictive MapReduce-like model of modern parallel computation among [117, 157, 110, 118], the Massively Parallel Computation (MPC) model of [110]. This model is widely used to solve different graph problems such as matching, vertex cover [111, 112, 113, 114, 115], independent set [115, 116], as well as many other algorithmic problems. As we detailed in Chapter 1, in this model, we have k machines (processors), each with space s. The computation proceeds in rounds. At the beginning of each round, the data (e.g., vertices and edges) is distributed across the machines. In each round, a machine performs local computation on its data (of size s), and then sends messages to other machines for the next round. Crucially, the total amount of communication sent or received by a machine is bounded by s, its space. Each machine treats the received messages as the input for the next round. Our model limits the number of machines and the memory per machine to be substantially sublinear in the size of the input. On the other hand, no restrictions are placed on the computational power of any individual machine. The main complexity measure is therefore the memory per machine and the number of rounds R required to solve a problem, which we consider to be the "parallel time" of the algorithm. For the rest of the chapter, $G(V, E)$ is a d-uniform hypergraph with n vertices and m hyperedges, and when the context is clear, we will simply refer to G as a graph and to its hyperedges as edges. $MM(G)$ denotes the maximum matching in G, and $\mu(G) = |MM(G)|$ is the size of that matching. We define $d_G(v)$ to be the degree of a vertex v (the number of hyperedges that v belongs to) in G. When the context is clear, we will omit indexing by G and simply denote it $d(v)$.

4.2 Our contribution and results.

We design and implement algorithms for the d-UHM in the MPC model. We will give three different algorithms, demonstrating different trade-offs between the model's parameters. Our algorithms are inspired by methods to find maximum matchings in graphs, but require developing significant new tools to address hypergraphs. We are not aware of previous algorithms for hypergraph matching in the MPC model. First we generalize the randomized coreset algorithm of [113] which finds an 3-rounds $O(1)$-approximation for matching in graphs. Our algorithm partitions the graph into random pieces across the machines, and then simply picks a maximum matching of each machine's subgraph. We show that this natural approach results in a $O(d^2)$-approximation. While the algorithmic generalization is straightforward, the analysis requires several new ideas.

Theorem 4.4.1 (restated). *There exists an MPC algorithm that with high probability computes a $(3d(d-1)+3+\epsilon)$-approximation for the d-UHM problem in 3 MPC rounds on machines of memory $s = \tilde{O}(\sqrt{nm})$.*

Our second result concerns the MPC model with per-machine memory $O(d \cdot n)$. We adapt the sampling technique and post-processing strategy of [111] to construct maximal matchings in hypergraphs, and are able to show that in d-uniform hypergraphs, this technique yields a maximal matching, and thus a d-approximation to the d-UHM problem in $O(\log n)$ rounds.

Theorem 4.5.1 (restated). *There exists an MPC algorithm that given a d-uniform hypergraph $G(V,E)$ with high probability computes a maximal matching in G in $O(\log n)$ MPC rounds on machines of memory $s = \Theta(d \cdot n)$.*

Our third result generalizes the edge degree constrained subgraphs (EDCS), originally introduced by Bernstein and Stein [7] for maintaining large matchings in dynamic graphs, and later used for several other problems including matching and vertex cover in the streaming and MPC model [114, 156]. We call these generalized subgraphs hyper-edge degree constrained subgraphs (HEDCS). We show that they exist for specific parameters, and that they contain a good approximation for the d-UHM problem. We prove that an HEDCS of a hypergraph G, with well chosen parameters, contain a fractional matching with a value at least $\frac{d}{d^2-d+1}\mu(G)$, and that the underlying fractional matching is special in the sense that for each hyperedge, it either assigns a value of 1 or a value less than some chosen ϵ. We call such a fractional matching an ϵ-restricted fractional matching. For Theorem 4.6.2, we compute an HEDCS of the hypergraph in a distributed fashion. This procedure relies on the robustness properties that we prove for the HEDCSs under sampling.

Theorem 4.6.1 (restated informally). *Let G be a d-uniform hypergraph and $0 < \epsilon < 1$. There exists an HEDCS that contains an ϵ-restricted fractional matching M_f^H with total value at least $\mu(G)\left(\frac{d}{d^2-d+1} - \epsilon\right)$.*

Theorem 4.6.2 (restated). *There exists an MPC algorithm that given a d-uniform hypergraph $G(V,E)$, where $|V| = n$ and $|E| = m$, can construct an HEDCS of G in 2 MPC rounds on machines of memory $s = \tilde{O}(n\sqrt{nm})$ in general and $s = \tilde{O}(\sqrt{nm})$ for linear hypergraphs.*

Corollary 4.6.4 (restated). *There exists an MPC algorithm that with high probability achieves a $d(d-1+1/d)^2$-approximation to the d-Uniform Hypergraph Matching in 3 rounds.*

Table 4.1 summarizes our results.

Approximation ratio	Rounds	Memory per machine	Computation per round
$3d(d-1)+3$	3	$\tilde{O}(\sqrt{nm})$	Exponential
d	$O(\log n)$	$O(dn)$	Polynomial
$d(d-1+1/d)^2$	3	$\tilde{O}(n\sqrt{nm})$ in general $\tilde{O}(\sqrt{nm})$ for linear hypergraphs	Polynomial

Table 4.1: Our parallel algorithms for the d-uniform hypergraph matching problem.

Experimental results. We implement our algorithms in a simulated MPC model environment and test them both on random and real-world instances. Our experimental results are consistent with the theoretical bounds on most instances, and show that there is a trade-off between the extent to which the algorithms use the power of parallelism and the quality of the approximations. This trade-off is illustrated by comparing the number of rounds and the performance of the algorithms on machines with the same memory size. See Section 2.7 for more details.

Our techniques. For Theorem 4.4.1 and Theorem 4.6.2, we use the concept of composable coresets, which has been employed in several distributed optimization models such as the streaming and MapReduce models [158, 159, 160, 161, 162, 163]. Roughly speaking, the main idea behind this technique is as follows: first partition the data into smaller parts. Then compute a representative solution, referred to as a coreset, from each part. Finally, obtain a solution by solving the optimization problem over the union of coresets for all parts. We use a randomized variant of composable coresets, first introduced in [120], where the above idea is applied on a random clustering of the data. This randomized variant has been used after for Graph Matching and Vertex Cover [113, 114], as well as Column Subset Selection [164]. Our algorithm for Theorem 4.4.1 is similar to previous works, but the analysis requires new techniques for handling hypergraphs. Theorem 4.5.1 is a relatively straightforward generalization of the corresponding result for matching in graphs.

The majority of our technical innovation is contained in Theorem 4.6.1 and 4.6.2. Our general approach is to construct, in parallel, an HEDCS that will contain a good approximation of the maximum matching in the original graph and that will fit on one machine. Then we can run an approximation algorithm on the resultant HEDCS to come up with a good approximation to the maximum matching in this HEDCS and hence in the original graph. In order to make this approach work, we need to generalize much of the known EDCS machinery [7, 156] to hypergraphs. This endeavor is quite involved, as almost all the proofs do not generalize easily and, as the results show, the resulting bounds are weaker than those for graphs. We first show that HEDCSs exist, and that they contain large fractional matchings. We then show that they are useful as a coreset, which amounts to showing that even though there can be many different HEDCSs of some fixed hypergraph $G(V, E)$, the degree distributions of every HEDCS (for the same parameters) are almost identical. In other words, the degree of any vertex v is almost the same in every HEDCS of G. We show also that HEDCS are robust under edge sampling, in the sense that edge sampling from a HEDCS yields another HEDCS. These properties allow to use HEDCS in our coresets and parallel algorithm in the rest of the chapter.

4.3 Related Work

Hypergraph Matching. The problem of finding a maximum matching in d-uniform hypergraphs is NP-hard for any $d \geq 3$ [21, 165], and APX-hard for $d = 3$ [129]. The most natural approach for an approximation algorithm for the hypergraph matching problem is the greedy algorithm: repeatedly add an edge that doesn't intersect any edges already in the matching. This solution is clearly within a factor of d from optimal: from the edges removed in each iteration, the optimal solution can contain at most d edges (at most one for each element of the chosen edge). It is also easy to construct examples showing that this analysis is tight for the greedy approach. All the best known approximation algorithms for the Hypergraph Matching Problem in d-uniform hypergraphs are based on local search methods [166, 167, 168, 169, 170]. The first such result by Hurkens and Schrijver [170] gave a $(\frac{d}{2} + \epsilon)$-approximation algorithm. Halldorsson [169] presented a quasi-polynomial $(\frac{d+2}{3})$-approximation algorithm for the unweighted d-UHM. Sviridenko and Ward [171] established a polynomial time $(\frac{d+2}{3})$-approximation algorithm that is the first polynomial time improvement over the $(\frac{d}{2} + \epsilon)$ result from [170]. Cygan [172] and Furer and Yu [173] both provide a $(\frac{d+1}{3} + \epsilon)$ polynomial time approximation algorithm, which is the best approximation guarantee known so far. On the other hand, Hazan, Safra and Schwartz [109] proved that it is hard to approximate d-UHM problem within a factor of $\Omega(d/\log d)$.

Matching in parallel computation models. The study of the graph maximum matching problem in parallel computation models can be traced back to PRAM algorithms of 1980s [12, 13, 14]. Since then it has been studied in the LOCAL and MPC models, and $(1+\epsilon)$- approximation can be achieved in $O(\log\log n)$ rounds using a space of $O(n)$ [114, 154, 115]. The question of finding a maximal matching in a small number of rounds has also been considered in [174, 111]. Recently, Behnezhad *et al.* [175] presented a $O(\log\log\Delta)$ round algorithm for maximal matching with $O(n)$ memory per machine. While we are not aware of previous work on hypergraph matching in the MPC model, finding a maximal hypergraph matching has been considered in the LOCAL model. Both Fischer *et al.* [176] and Harris [177] provide a deterministic distributed algorithm that computes $O(d)$-approximation to the d-UHM.

Maximum Independent Set. The d-UHM is strongly related to different variants of the Independent Set problem as well as other combinatorial problems. The maximum independent set (MIS) problem on degree bounded graphs can be mapped to d-UHM when the degree bound is d [178, 179, 180, 181]. The d-UHM problem can also be studied under a more general problem of maximum independent set on $(d+1)$-claw-free graphs [166, 168]. (See [182] of connections between d-UHM and other combinatorial optimization problems).

4.4 A 3-round $O(d^2)$-approximation

In this section, we generalize the randomized composable coreset algorithm of Assadi and Khanna [113]. They used a maximum matching as a coreset and obtained a $O(1)$-approximation. We use a hypergraph maximum matching and we obtain a $O(d^2)$-approximation. We first define a

k-partitioning and then present our greedy approach.

Definition 4.4.1 (Random k-partitioning)**.** *Let E be an edge-set of a hypergraph $G(V, E)$. We say that a collection of edges $E^{(1)}, \dots, E^{(k)}$ is a random k-partition of E if the sets are constructed by assigning each edge $e \in E$ to some $E^{(i)}$ chosen uniformly at random. A random k-partition of E naturally results in partitioning the graph G into k subgraphs $G^{(1)}, \dots, G^{(k)}$ where $G^{(i)} := G(V, E^{(i)})$ for all $i \in [k]$.*

Let $G(V, E)$ be any d-uniform hypergraph and $G^{(1)}, \dots, G^{(k)}$ be a random k-partitioning of G. We describe a simple greedy process for combining the maximum matchings of $G^{(i)}$, and prove that this process results in a $O(d^2)$-approximation of the maximum matching of G.

Algorithm 11 Greedy.

```
Input: Hypergraph $G$
Output: A matching $M$ on $G$
1: Construct a random $k$-partitioning of $G$ across the $k$ machines.
2: Let $M^{(0)} := \emptyset$.
3: for $i = 1$ to $k$
4:     Set $MM(G^{(i)})$ to be an arbitrary hypergraph maximum matching of $G^{(i)}$.
5:     Let $M^{(i)}$ be a maximal matching obtained by adding to $M^{(i-1)}$ the edges of $MM(G^{(i})$ that
       do not violate the matching property.
6: return $M := M^{(k)}$.
```

Theorem 4.4.1. *Greedy computes, with high probability, a $\big(3d(d-1) + 3 + o(1)\big)$-approximation for the d-UHM problem in 3 MPC rounds on machines of memory $s = \tilde{O}(\sqrt{nm})$.*

In the rest of the section, we present the proof of Theorem 4.4.1. Let $c = \frac{1}{3d(d-1)+3}$, we show that $M^{(k)} \geq c \cdot \mu(G)$ w.h.p, where $M^{(k)}$ is the output of Greedy. The randomness stems from the fact that the matchings $M^{(i)}$ (for $i \in \{1, \dots, k\}$) constructed by Greedy are random variables depending on the random k-partitioning. We adapt the general approach in [113] for d-uniform hypergraphs, with $d \geq 3$. Suppose at the beginning of the i-th step of Greedy, the matching $M^{(i-1)}$ is of size $o(\mu(G)/d^2)$. One can see that in this case, there is a matching of size $\Omega(\mu(G))$ in G that is entirely incident on vertices of G that are not matched by $M^{(i-1)}$. We can show that in fact $\Omega(\mu(G)/(d^2 k))$ edges of this matching are appearing in $G^{(i)}$, even when we condition on the assignment of the edges in the first $(i-1)$ graphs. Next we argue that the existence of these edges forces any maximum matching of $G^{(i)}$ to match $\Omega(\mu(G)/(d^2k))$ edges in $G^{(i)}$ between the vertices that are not matched by $M^{(i-1)}$. These edges can always be added to the matching $M^{(i-1)}$ to form $M^{(i)}$. Therefore, while the maximal matching in Greedy is of size $o(\mu(G))$, we can increase its size by $\Omega(\mu(G)/(d^2k))$ edges in each of the first $k/3$ steps, hence obtaining a matching of size $\Omega(\mu(G)/d^2)$ at the end. The following Lemma 4.4.1 formalizes this argument.

Lemma 4.4.1. *For any $i \in [k/3]$, if $M^{(i-1)} \leq c \cdot \mu(G)$, then, w.p. $1 - O(1/n)$,*

$$M^{(i)} \geq M^{(i-1)} + \frac{1 - 3d(d-1)c - o(1)}{k} \cdot \mu(G) \ .$$

Before proving the lemma, we define some notation. Let M^* be an arbitrary maximum matching of G. For any $i \in [k]$, we define $M^{*<i}$ as the part of M^* assigned to the first $i-1$ graphs in the random k-partitioning, i.e., the graphs $G^{(1)}, \ldots G^{(i-1)}$. We have the following concentration result:

Claim 4.4.1. *W.p. $1 - O(1/n)$, for any $i \in [k]$:*

$$M^{*<i} \leq \left(\frac{i - 1 + o(i)}{k}\right) \cdot \mu(G)$$

Proof of claim 4.4.1. Fix an $i \in [k]$; each edge in M^* is assigned to $G^{(1)}, \ldots, G^{(i-1)}$, w.p. $(i-1)/k$, hence in expectation, size of $M^{*<i}$ is $\frac{i-1}{k} \cdot \mu(G)$. For a large n, the ratio $\frac{\mu(G)}{k}$ is large and the claim follows from a standard application of Chernoff bound. □

Proof of Lemma 4.4.1. Fix an $i \in [k/3]$ and the set of edges for $E^{(1)}, \ldots E^{(i-1)}$. This also fixes the matching $M^{(i-1)}$ while the set of edges in $E^{(i)}, \ldots, E^{(k)}$ together with the matching $M^{(i)}$ are still random variables. We further condition on the event that after fixing the edges in $E^{(1)}, \ldots, E^{(i-1)}$, $|M^{*<i}| \leq \frac{i-1+o(i)}{k} \cdot \mu(G)$ which happens w.p. $1 - O(1/n)$ by claim 4.4.1.

Let V_{old} be the set of vertices incident on $M^{(i-1)}$ and V_{new} be the remaining vertices. Let $E^{\geq i}$ be the set of edges in $E \setminus E^{(1)} \cup \ldots \cup E^{(i-1)}$. We partition $E^{\geq i}$ into two parts: (i) E_{old}: the set of edges with *at least one* endpoint in V_{old}, and (ii) E_{new}: the set of edges incident entirely on V_{new}. Our goal is to show that w.h.p. any maximum matching of $G^{(i)}$ matches $\Omega(\mu(G)/k)$ vertices in V_{new} to each other by using the edges in E_{new}. The lemma then follows easily from this.

The edges in the graph $G^{(i)}$ are chosen by independently assigning each edge in $E^{\geq i}$ to $G^{(i)}$ w.p. $1/(k-i+1)$. This independence makes it possible to treat the edges in E_{old} and E_{new} separately. We can fix the set of sampled edges of $G^{(i)}$ in E_{old}, denoted by E^{i}_{old}, without changing the distribution of edges in $G^{(i)}$ chosen from E_{new}. Let $\mu_{old} := MM(G(V, E^{i}_{old}))$, i.e., the maximum number of edges that can be matched in $G^{(i)}$ using only the edges in E^{i}_{old}. In the following, we show that w.h.p., there exists a matching of size $\mu_{old} + \Omega(\mu(G)/k)$ in $G^{(i)}$. By the definition of μ_{old}, this implies that any maximum matching of $G^{(i)}$ has to use at least $\Omega(\mu(G)/k)$ edges in E_{new}.

Let M_{old} be any arbitrary maximum matching of size μ_{old} in $G(V, E^{i}_{old})$. Let $V_{new}(M_{old})$ be the set of vertices in V_{new} that are incident on M_{old}. We show that there is a large matching in $G(V, E_{new})$ that avoids $V_{new}(M_{old})$.

Claim 4.4.2. $|V_{new}(M_{old})| < c \cdot d(d-1) \cdot \mu(G)$.

Proof of Claim 4.4.2. Since any edge in M_{old} has at least one endpoint in V_{old}, we have $|V_{new}(M_{old})| \leq (d-1)|M_{old}| \leq (d-1)|V_{old}|$. By the assertion of the lemma, $|M^{(i-1)}| < c \cdot \mu(G)$, and hence $V_{new}(M_{old})| \leq (d-1) \cdot |V_{old}| \leq d(d-1) \cdot |M^{(i-1)}| < c \cdot d(d-1) \cdot \mu(G)$. □

Claim 4.4.3. *There exists a matching in $G(V, E_{new})$ of size $\left(\frac{k-i+1-o(i)}{k} - 2d(d-1)c\right) \cdot \mu(G)$ that avoids the vertices of $V_{new}(M_{old})$.*

Proof of Claim 4.4.3. By the assumption that $|M^{*<i}| \leq \frac{i-1+o(i)}{k} \cdot \mu(G)$, there is a matching of size $\frac{k-i+1-o(i)}{k} \cdot \mu(G)$ in the graph $G(V, E^{\geq i})$. By removing the edges in M that are either incident on V_{old} or $V_{new}(M_{old})$, at most $2d(d-1)c \cdot \mu(G)$ edges are removed from M. Now the remaining matching is entirely contained in E_{new} and also avoids $V_{new(Mold)}$, hence proving the claim. □

We are now ready to finalize the proof of Lemma 4.4.1. Let M_{new} be the matching guaranteed by Claim 4.4.3. Each edge in this matching is chosen in $G^{(i)}$ w.p. $1/(k-i+1)$ independent of the other edges. Hence, by Chernoff bound, there is a matching of size

$$(1 - o(1)) \cdot \left(\frac{1}{k} - \frac{o(i)}{k(k-i+1)} - \frac{2d(d-1)c}{k-i+1}\right) \cdot \mu(G) \geq \frac{1 - o(1) - 3d(d-1)c}{k} \cdot \mu(G)$$

in the edges of M_{new} that appear in $G^{(i)}$ (for $i \leq k/3$). This matching can be directly added to the matching M_{old}, implying the existence of a matching of size $\mu_{old} + \frac{1-o(1)-3d(d-1)c}{k} \cdot \mu(G)$ in $G^{(i)}$. As we argued before, this ensures that any maximum matching of $G^{(i)}$ contains at least $\frac{1-o(1)-3d(d-1)c}{k} \cdot \mu(G)$ edges in E_{new}. These edges can always be added to $M^{(i-1)}$ to form $M^{(i)}$, hence proving the lemma. □

Proof of Theorem 4.4.1. Recall that $M := M^{(k)}$ is the output matching of Greedy. For the first $k/3$ steps of Greedy, if at any step we got a matching of size at least $c \cdot \mu(G)$, then we are already done. Otherwise, at each step, by Lemma 4.4.1, w.p. $1 - O(1/n)$, we increase the size of the maximal matching by $\frac{1-3d(d-1)c-o(1)}{k} \cdot \mu(G)$ edges; consequently, by taking a union bound on the $k/3$ steps, w.p. $1 - o(1)$, the size of the maximal matching would be $\frac{1-3d(d-1)c-o(1)}{3} \cdot \mu(G)$. Since $c = 1/(3d(d-1)+3)$, we ensure that $\frac{1-3d(d-1)c}{3} = c$ and in either case, the matching computed by Greedy is of size at least $\mu(G)/(3d(d-1)+3) - o(\mu(G))$, and this proves that Greedy is a $O(d^2)$-approximation. All is left is to prove that Greedy can be implemented in 3 rounds with a memory of $\tilde{O}(\sqrt{nm})$ per machine. Let $k = \sqrt{\frac{m}{n}}$ be the number of machines, each with a memory of $\tilde{O}(\sqrt{nm})$. We claim that Greedy can run in three rounds. In the first round, each machine randomly partitions the edges assigned to it across the k machines. This results in a random k-partitioning of the graph across the machines. In the second round, each machine sends a maximum matching of its input to a designated central machine M; as there are $k = \sqrt{\frac{m}{n}}$ machines and each machine is sending $\tilde{O}(n)$ size coreset, the input received by M is of size $\tilde{O}(\sqrt{nm})$ and hence can be stored entirely on that machine. Finally, M computes the answer by combining the matchings. □

We conclude this section by stating that computing a maximum matching on every machine is only required for the analysis, i.e., to show that there exists a large matching in the union of coresets. In practice, we can use an approximation algorithm to obtain a large matching from the coresets. In our experiments, we use a maximal matching instead of maximum.

4.5 A $O(\log n)$-rounds d-approximation algorithm

In this section, we show how to compute a maximal matching in $O(\log n)$ MPC rounds, by generalizing the algorithm in [111] to d-uniform hypergraphs. The algorithm provided by Lattanzi *el al.* [111] computes a maximal matching in graphs in $O(\log n)$ if $s = \Theta(n)$, and in at most$\lfloor c/\epsilon \rfloor$ iterations when $s = \Theta(n^{1+\epsilon})$, where $0 < \epsilon < c$ is a fixed constant. Harvey *et al.* [116] show a similar result.

The algorithm first samples $O(s)$ edges and finds a maximal matching M_1 on the resulting subgraph (we will specify a bound on the memory s later). Given this matching, we can safely remove edges that are in conflict (i.e., those incident on nodes in M_1) from the original graph G. If the resulting filtered graph H is small enough to fit onto a single machine, the algorithm augments M_1 with a matching found on H. Otherwise, we augment M_1 with the matching found by recursing on H. Note that since the size of the graph reduces from round to round, the effective sampling probability increases, resulting in a larger sample of the remaining graph.

Theorem 4.5.1. *Given a d-uniform hypergraph $G(V, E)$,* Iterated-Sampling *omputes a maximal matching in G with high probability in $O(\log n)$ MPC rounds on machines of memory $s = \Theta(d \cdot n)$.*

Algorithm 12 Iterated-Sampling

```
Input: A $d$-uniform hypergraph $G$, spread across the machines
Output: A maximal matching $M$
1: Set $M := \emptyset$ and $\mathcal{S} = E$.
2: Sample every edge $e \in \mathcal{S}$ uniformly with probability $p = \frac{s}{5|\mathcal{S}|d}$ to form $E'$.
3: If $|E'| > s$ the algorithm fails.
4: Else give the graph $G(V, E')$ as input to a single machine and compute a maximal matching
   $M'$ on it. Set $M = M \cup M'$.
5: Let $I$ be the unmatched vertices in $G$ and $G[I]$ the induced subgraph with edges $E[I]$. If
   $E[I] > s$, set $\mathcal{S} := E[I]$ and return to step 2.
6: Compute a maximal matching $M''$ on $G[I]$ and output $M = M \cup M''$.
```

Next we present the proof of Theorem 4.5.1. We first show that after sampling edges, the size of the graph $G[I]$ induced by unmatched vertices decreases exponentially with high probability, and therefore the algorithm terminates in $O(\log n)$ rounds. Next we argue that M is indeed a maximal matching by showing that if after terminating, there is an edge that's still unmatched, this will yield a contradiction. This is formalized in the following lemmas.

Lemma 4.5.1. *Let $E' \subset E$ be a set of edges chosen independently with probability p. Then with probability at least $1 - e^{-n}$, for all $I \subset V$ either $|E[I]| < 2n/p$ or $E[I] \cap E' \neq \emptyset$.*

Proof of Lemma 4.5.1. Fix one such subgraph, $G[I] = (I, E[I])$ with $|E[I]| \geq 2n/p$. The probability that none of the edges in $E[I]$ were chosen to be in E' is $(1-p)^{|E[I]|} \leq (1-p)^{2n/p} \leq e^{-2n}$. Since there are at most 2^n total possible induced subgraphs $G[I]$ (because each vertex is either matched or unmatched), the probability that there exists one that does not have an edge in E' is at most $2^n e^{-2n} \leq e^{-n}$. □

Lemma 4.5.2. *If $s \geq 20d \cdot n$ then* Iterated-Sampling *runs for at most $O(\log n)$ iterations with high probability.*

Proof of Lemma 4.5.2. Fix an iteration i of Iterated-Sampling and let p be the sampling probability for this iteration. Let E_i be the set of edges at the beginning of this iteration, and denote by I be the set of unmatched vertices after this iteration. From Lemma 4.5.1, if $|E[I]| \geq 2n/p$ then an edge of $E[I]$ will be sampled with high probability. Note that no edge in $E[I]$ is incident on any edge in M'. Thus, if an edge from $E[I]$ is sampled then Iterated-Sampling would have chosen this edge to be in the matching. This contradicts the fact that no vertex in I is matched. Hence, $|E[I]| \leq 2n/p$ with high probability.

Now consider the first iteration of the algorithm, let $G_1(V_1, E_1)$ be the induced graph on the unmatched nodes after the first step of the algorithm. The above argument implies that $|E_1| \leq \frac{10d \cdot n|E_0|}{s} \leq \frac{10d \cdot n|E|}{s} \leq \frac{|E|}{2}$. Similarly $|E_2| \leq \frac{10d \cdot n|E_1|}{s} \leq \frac{(10d \cdot n)^2|E_1|}{s^2} \leq \frac{|E|}{2^2}$. After i iterations : $|E_i| \leq \frac{|E|}{2^i}$, and the algorithm will terminate after $O(\log n)$ iterations. □

Lemma 4.5.3. Iterated-Sampling *finds a maximal matching of G with high probability.*

Proof of Lemma 4.5.3. First consider the case that the algorithm does not fail. Suppose there is an edge $e = \{v_1, \ldots, v_d\} \in E$ such that none of the v_i's are matched in the final matching M that the algorithm output. In the last iteration of the algorithm, since $e \in E$ and its endpoints are not matched, then $e \in E[I]$. Since this is the last run of the algorithm, a maximal matching M'' of $G[I]$ is computed on one machine. Since M'' is maximal, at least one of the v_i's must be matched in it. All of the edges of M'' get added to M in the last step. This yields a contradiction.

Next, consider the case that the algorithm fails. This occurs due to the set of edges E' having size larger than the memory in some iteration of the algorithm. Note that $\mathbb{E}[|E'|] = |S| \cdot p = s/5d \leq s/10$ in a given iteration. By the Chernoff Bound it follows that $|E'| \geq s$ with probability smaller than $2^s \geq 2^{-20d \cdot n}$ (since $s \geq 20d \cdot n$). By Lemma 4.5.2 the algorithm completes in at most $O(\log n)$ rounds, thus the total failure probability is bounded by $O(\log n \cdot 2^{-20 \cdot n})$ using the union bound. □

We are now ready to show that Iterated-Sampling can be implemented in $O(\log n)$ MPC rounds.

Proof of Theorem 4.5.1. We show that Iterated-Sampling can be implemented in the MPC model with machines of memory $\Theta(dn)$ and $O(\log n)$ MPC rounds. Combining this with Lemma 4.5.3 on the correctness of Iterated-Sampling , we immediately obtain Theorem 4.5.1 for the case of $s = \Theta(dn)$. Every iteration of Iterated-Sampling can be implemented in $O(1)$ MPC rounds. Suppose the edges are initially distributed over all machines. We can sample every edge with the probability p (in each machine) and then send the sampled edges E' to the first machine. With high probability we will have $|E'| = O(n)$, so we can fit the sampled edges in the first machine. Therefore, the sampling can be done in one MPC round. Computing the subgraph of G induced by I can be done in 2 rounds; one round to send the list of unmatched vertices I from the first machine to all the other machines, and the other round to compute the subgraph $G[I]$, and send it to a single machine if it fits, or start the sampling again. □

4.6 A 3-round $O(d^3)$-approximation using HEDCSs

In graphs, edge degree constrained subgraphs (EDCS) [7, 156] have been used as a local condition for identifying large matchings, and leading to good dynamic and parallel algorithms [7, 156, 114]. These papers showed that an EDCS exists, that it contains a large matching, that it is robust to sampling and composition, and that it can be used as a coreset.

In this section, we present a generalization of EDCS for hypergraphs. We prove that an HEDCS exists, that it contains a large matching, that it is robust to sampling and composition, and that it can be used as a coreset. The proofs and algorithms, however, require significant developments beyond the graph case. We first present definitions of a fractional matching and HEDCS.

Definition 4.6.1 (Fractional matching)**.** *In a hypergraph $G(V, E)$, a fractional matching is a mapping from $y : E \mapsto [0, 1]$ such that for every edge e we have $0 \leq y_e \leq 1$ and for every vertex v : $\sum_{e \in v} y_e \leq 1$. The value of such fractional matching is equal to $\sum_{e \in E} y_e$. For $\epsilon > 0$, a fractional matching is an ϵ-restricted fractional matching if for every edge e: $y_e = 1$ or $y_e \in [0, \epsilon]$.*

Definition 4.6.2. *For any hypergraph $G(V, E)$ and integers $\beta \geq \beta^- \geq 0$ a hyperedge degree constraint subgraph HEDCS(H, β, β^-) is a subgraph $H := (V, E_H)$ of G satisfying:*

- *(P1): For any hyperedge $e \in E_H$: $\sum_{v \in e} d_H(v) \leq \beta$.*
- *(P2): For any hyperedge $e \notin E_H$: $\sum_{v \in e} d_H(v) \geq \beta^-$.*

We show via a constructive proof that a hypergraph contains an HEDCS when the parameters of this HEDCS satisfy the inequality $\beta - \beta^- \geq d - 1$.

Lemma 4.6.1. *Any hypergraph G contains an HEDCS(G, β, β^-) for any parameters $\beta - \beta^- \geq d - 1$.*

Proof. Consider the following simple procedure for creating an HEDCS H of a given hypergraph G: start by initializing H to be equal to G. And while H is not an HEDCS(G, β, β^-), find an edge e which violates one of the properties of HEDCS and fix it. Fixing the edge e implies removing it from H if it was violating Property *(P1)* and adding it to H if it was violating Property *(P2)*.

The output of the above procedure is clearly an HEDCS of graph G. However, a-priori it is not clear that this procedure ever terminates as fixing one edge e can result in many edges violating the HEDCS properties, potentially undoing the previous changes. In the following, we use a potential function argument to show that this procedure always terminates after a finite number of steps, hence implying that a $HEDCS(G, \beta, \beta^-)$ always exists.

We define the following potential function Φ:

$$\Phi := (\frac{2}{d}\beta - \frac{d-1}{d}) \cdot \sum_{v \in V} d_H(v) - \sum_{e \in H} \sum_{u \in e} d_H(u)$$

We argue that in any step of the procedure above, the value of Φ increases by at least 1. Since the maximum value of Φ is at most $O(\frac{2}{d}n \cdot \beta^2)$, this immediately implies that this procedure terminates in $O(\frac{2}{d}n \cdot \beta^2)$ iterations.

Define $\Phi_1 = (\frac{2}{d}\beta - \frac{d-1}{d}) \cdot \sum_{v \in V} d_H(v)$ and $\Phi_2 = \sum_{e \in H} \sum_{u \in e} d_H(u)$. Let e be the edge we choose to fix at this step, H_b be the subgraph before fixing the edge e, and H_a be the resulting subgraph. Suppose first that the edge e was violating Property *(P1)* of HEDCS. As the only change is in the degrees of vertices $v \in e$, Φ_1 decreases by $(2\beta - (d-1))$. On the other hand, $\sum_{v \in e} d_{H_b}(v) \geq \beta + 1$ originally (as e was violating Property (P1) of HEDCS), and hence after removing e, Φ_2 increases by $\beta + 1$. Additionally, for each edge e_u incident upon $u \in e$ in H_a , after removing the edge e, $\sum_{v \in e_u} d_{H_a}(v)$ decreases by one. As there are at least $\sum_{u \in e} d_{H_a}(u) = \sum_{u \in e} d_{H_b}(u) - d \geq \beta - (d-1)$ choices for e_u, this means that in total, Φ_2 increases by at least $2\beta + 1 - (d-1)$. As a result, in this case Φ increases by at least 1 after fixing the edge e.

Now suppose that the edge e was violating Property *(P2)* of HEDCS instead. In this case, degree of vertices $u \in e$ all increase by one, hence Φ_1 increases by $2\beta - (d-1)$. Additionally, note that since edge e was violating Property (P2) we have $\sum_{v \in e} d_{H_b}(v) \leq \beta^- - 1$, so the addition of edge e decreases Φ_2 by at most $\sum_{v \in e} d_{H_a}(v) = \sum_{v \in e} d_{H_b}(v) + d \leq \beta^- - 1 + d$. Moreover, for each edge e_u incident upon $u \in e$, after adding the edge e, $\sum\limits_{v \in e_u} d_{H_a}(v)$ increases by one and since there are at most $\sum\limits_{v \in e} d_{H_b}(v) \leq \beta^- - 1$ choices for e_u, Φ_2 decreases in total by at most $2\beta^- - 2 + d$. The total variation in Φ is therefore equal to $2\beta - (d-1) - (2\beta^- - 2 + d) = 3 + 2(\beta - \beta^- - d)$. So if $\beta - \beta^- \geq d - 1$, we have that Φ increases by at least 1 after fixing edge e. □

The main result in this section shows that an HEDCS of a graph contains a large ϵ-restricted fractional matching that approximates the maximum matching by a factor less than d.

Theorem 4.6.1. *Let G be a d-uniform hypergraph and $0 \leq \epsilon < 1$. Let $H := HEDCS(G, \beta, \beta(1-\lambda))$ with $\lambda = \frac{\epsilon}{6}$ and $\beta \geq \frac{8d^2}{d-1} \cdot \lambda^{-3}$. Then H contains an ϵ-restricted fractional matching M_f^H with total value at least $\mu(G)(\frac{d}{d^2-d+1} - \frac{\epsilon}{d-1})$.*

In order to prove Theorem 4.6.1, we will need the following two lemmas. The first lemma we prove is an algebraic result that will help us bound the contribution of vertices in the ϵ-restricted fractional matching. The second lemma identifies additional structure on the HEDCS, that we will use to construct the fractional matching of the theorem. For proofs of both lemmas, see Appendix C.1.2

Lemma 4.6.2. *Let $\phi(x) = \min\{1, \frac{(d-1)x}{d(\beta - x)}\}$. If $a_1, \ldots a_d \geq 0$ and $a_1 + \ldots + a_d \geq \beta(1-\lambda)$ for some $\lambda \geq 0$, then $\sum\limits_{i=1}^{d} \phi(a_i) \geq 1 - 5 \cdot \lambda$.*

Lemma 4.6.3. *Given any $HEDCS(G, \beta, \beta(1-\lambda))$ H, we can find two disjoint sets of vertices X and Y that satisfy the following properties:*

1. *$|X| + |Y| = d \cdot \mu(G)$.*
2. *There is a perfect matching in Y using edges in H.*
3. *Letting $\sigma = \frac{|Y|}{d} + \sum\limits_{x \in X} \phi(d_H(x))$, we have that $\sigma \geq \mu(G)(1 - 5\lambda)$.*
4. *All edges in H with vertices in X have at least one other vertex in Y, and have vertices only in X and Y.*

We are now ready to prove Theorem 4.6.1.

Proof of theorem 4.6.1. Suppose we have two sets X and Y satisfying the properties of Lemma 4.6.3. We construct an ϵ-restricted fractional matching M_f^H using the edges in H such that

$$val(M_f^H) \geq (\frac{d}{d^2 - d + 1} - \frac{\epsilon}{d-1})\mu(G),$$

where $val(M_f^H)$ is the value of the fractional matching M_f^H. Now, by Property 2 of Lemma 4.6.3, $|Y|$ contains a perfect matching M_Y^H using edges in H. Let Y^- be a subset of Y obtained by randomly sampling exactly $1/d$ edges of M_Y^H and adding their endpoints to Y^- Let $Y^* = Y \setminus Y^-$ and observe that $|Y^-| = |Y|/d$ and $|Y^*| = \frac{d-1}{d}|Y|$.

Let H^* be the subgraph of H induced by $X \cup Y^*$ (each edge in H^* has vertices in only X and Y^*). We define a fractional matching $M_f^{H^*}$ on the edges of H^* in which all edges have value at most ϵ. We will then let our final fractional matching M_f^H be the fractional matching $M_f^{H^*}$ joined with the perfect matching in H of Y^- (so M_f^H assigns value 1 to the edges in this perfect matching). M_f^H is, by definition, an ϵ-restricted fractional matching.

We now give the details for the construction of $M_f^{H^*}$. Let $V^* = X \cup Y^*$ be the vertices of H^*, and let E^* be its edges. For any vertex $v \in V^*$, define $d_H^*(v)$ to be the degree of v in H^*. Recall that by Property 4 of Lemma 4.6.3, if $x \in X$ then all the edges of H incident to x go to Y (but some might go to Y^-). Thus, for $x \in X$, we have $E[d_H^*(x)] \geq \frac{d_H(x)(d-1)}{d}$.

We now define $M_f^{H^*}$ as follows. For every $x \in X$, we arbitrarily order the edges of H incident to x, and then we assign/add a value of $\min\left\{\frac{\epsilon}{|X\cap e|}, \frac{1}{|X\cap e|}\frac{1}{\beta - d_H(x)}\right\}$ to these edges one by one, stopping when either $val(x)$ (the sum of values assigned to vertices incident to x) reaches 1 or there are no more edges in H incident to x, whichever comes first. In the case that $val(x)$ reaches 1 the last edge might have added value less than $\min\left\{\frac{\epsilon}{|X\cap e|}, \frac{1}{|X\cap e|}\frac{1}{\beta - d_H(x)}\right\}$, where e is the last edge to be considered.

We now verify that $M_f^{H^*}$ is a valid fractional matching in that all vertices have value at most 1. This is clearly true of vertices $x \in X$ by construction. For a vertex $y \in Y^*$, it suffices to show that each edge incident to y receives a value of at most $1/d_H(y) \leq 1/d^*_H(y)$. To see this, first note that the only edges to which $M_f^{H^*}$ assigns non-zero values have at least two endpoints in $X \times Y^*$. Any such edge e receives value at most $\min\left\{\epsilon, \sum_{x \in X \cap e} \frac{1}{|X \cap e|} \frac{1}{\beta - d_H(x)}\right\}$, but since e is in $M_f^{H^*}$ and so in H, we have by Property *(P1)* of an HEDCS that $d_H(y) \leq \beta - d_H(x)$, and so $\sum_{x \in X \cap e} \frac{1}{|X \cap e|} \frac{1}{\beta - d_H(x)} \leq \frac{1}{|X \cap e|} \frac{|X \cap e|}{d_H(y)} \leq \frac{1}{d_H(y)}$.

By construction, for any $x \in X$, we have that the value $val(x)$ of x in $M_f^{H^*}$ satisfies :

$$\begin{aligned} val(x) &= \min\left\{1, \sum_{e \ni x} \min\left\{\frac{\epsilon}{|X \cap e|}, \frac{1}{|X \cap e|} \frac{1}{\beta - d_H(x)}\right\}\right\} \\ &\geq \min\left\{1, d^*_H(x) \cdot \min\left\{\frac{\epsilon}{d-1}, \frac{1}{d-1} \frac{1}{\beta - d_H(x)}\right\}\right\} . \end{aligned}$$

Furthermore, we can bound the value of the fractional matching M_f^{H*}: as $val(M_f^{H*}) \geq \sum_{x \in X} val(x)$. For convenience, we use $val'(x) = \min\left\{1, d^*_H(x) \cdot \min\left\{\epsilon, \frac{1}{\beta - d_H(x)}\right\}\right\}$ such that

$$val(x) \geq \frac{val'(x)}{d-1} \text{ and} \tag{4.1}$$

$$val(M_f^{H*}) \geq \frac{1}{d-1} \sum_{x \in X} val'(x) , \tag{4.2}$$

Next we present a lemma, proved in Appendix C.1.3, that bounds $val'(x)$ for each vertex.

Lemma 4.6.4. *For any $x \in X$, $E[val'(x)] \geq (1 - \lambda)\phi(d_H(x))$.*

This last lemma, combined with (4.2), allows us to lower bound the value of $M_f^{H^*}$:

$$val(M_f^{H^*}) \geq \frac{1}{d-1} \sum_{x \in X} val'(x) \geq \frac{1-\lambda}{d-1} \sum_{x \in X} \phi(d_H(x)).$$

Note that we have constructed M_f^H by taking the fractional value in $M_f^{H^*}$ and adding the perfect matching on edges from Y^-. The latter matching has size $\frac{|Y^-|}{d} = \frac{|Y|}{d^2}$, and the value of M_f^H is bounded by:

$$\begin{aligned}
val(M_f^H) &\geq \frac{1}{d-1}(1-\lambda)\sum_{x\in X}\phi(d_H(x)) + \frac{|Y|}{d^2} \\
&= \frac{1}{d-1}\big((1-\lambda)\sum_{x\in X}\phi(d_H(x)) + \frac{|Y|}{d}\big) - \frac{|Y|}{d^2(d-1)} \\
&\geq \frac{1}{d-1}(1-\lambda)(1-5\lambda)\mu(G) - \frac{|Y|}{d^2(d-1)} \\
&\geq \frac{1}{d-1}(1-6\lambda)\mu(G) - \frac{|Y|}{d^2(d-1)}\ .
\end{aligned}$$

To complete the proof, recall that Y contains a perfect matching in H of $|Y|/d$ edges, so if $\frac{|Y|}{d} \geq \frac{d}{d(d-1)+1}\mu(G)$ then there already exists a matching in H of size at least $\frac{d}{d(d-1)+1}\mu(G)$, and the theorem is true. We can thus assume that $|Y|/d < \big(\frac{d}{d(d-1)+1}\big)\mu(G)$, in which case the previous equation yields that:

$$\begin{aligned}
val(M_f^H) &\geq \frac{1}{d-1}(1-6\lambda)\mu(G) - \frac{|Y|}{d^2(d-1)} \\
&\geq (\frac{1-6\lambda}{d-1})\mu(G) - \frac{\mu(G)}{(d-1)(d(d-1)+1)} \\
&= (\frac{d}{d(d-1)+1} - \frac{6\lambda}{d-1})\mu(G)\ .
\end{aligned}$$

In both cases we get that

$$val(M_f^H) \geq (\frac{d}{d^2-d+1} - \frac{6\lambda}{d-1})\mu(G).$$

□

4.6.1 Sampling and constructing an HEDCS in the MPC model

Our results in this section are general and applicable to every computation model. We prove structural properties about the HEDCSs that will help us construct them in the MPC model. We show that the degree distributions of every HEDCS (for the same parameters β and λ) are almost identical. In other words, the degree of any vertex v is almost the same in every HEDCS of the same hypergraph G. We show also that HEDCS are robust under edge sampling, i.e., that edge sampling from and HEDCS yields another HEDCS, and that the degree distributions of any two HEDCS for two different edge sampled subgraphs of G is almost the same no matter how the two HEDCS are selected. In the following lemma, we argue that any two HEDCS of a graph G (for the same parameters β, β^-) are "somehow identical" in that their degree distributions are close to each other. In the rest of this section, we fix the parameters β, β^- and two subgraphs A and B that are both HEDCS(G, β, β^-).

Lemma 4.6.5. *(Degree Distribution Lemma). Fix a d-uniform hypergraph $G(V, E)$ and parameters β, $\beta^- = (1 - \lambda) \cdot \beta$ (for λ small enough). For any two subgraphs A and B that are HEDCS(G, β, β^-), and any vertex $v \in V$, then $|d_A(v) - d_B(v)| = O(\sqrt{n})\lambda^{1/2}\beta$.*

Proof. Suppose that we have $d_A(v) = k\lambda\beta$ for some k and that $d_B(v) = 0$. We will show that if the $k = \Omega(\frac{\sqrt{n}}{\lambda^{1/2}})$, then this will lead to a contradiction. Let e be one of the $k\lambda\beta$ edges that are incident to v in A. $e \notin B$ so $\sum\limits_{u \neq v} d_B(u) \geq (1 - \lambda)\beta$. From these $(1 - \lambda)\beta$ edges, at most $(1 - k\lambda)\beta$ can be in A in order to respect *(P1)*, so at least we will have $(k - 1)\lambda\beta$ edges in $B \setminus A$, thus we have now covered $k\lambda\beta + (k - 1)\lambda\beta$ edges in both A and B. Let's keep focusing on the edge e, and especially on one of its $(k - 1)\lambda\beta$ incident edges in $B \setminus A$. Let e_1 be such an edge. $e_1 \in B \setminus A$, therefore $\sum\limits_{v' \in e_1} d_A(v') \geq (1 - \lambda)\beta$. The edges incident to e_1 in A that we have covered so far are at most $(1 - k\lambda)\beta$, therefore we still need at least $(k - 1)\lambda$ new edges in A to respect *(P1)*. Out of these $(k - 1)\lambda$ edges, at most $\lambda\beta$ can be in B (because e_1 has already $(1 - \lambda)\beta$ covered edges incident to it in B). Therefore at least $(k - 2)\lambda\beta$ are in $A \setminus B$. Thus, we have so far covered at least $k\lambda\beta + (k - 1)\lambda\beta + (k - 2)\lambda\beta$. One can see that we can keep doing this until we cover at least $\frac{k(k+1)}{2}\lambda\beta$ edges in both A and B. The number of edges in each of A and B cannot exceed $n \cdot \beta$ (each vertex has degree $\leq \beta$), therefore we will get a contradiction if $\frac{k(k+1)}{2}\lambda\beta > 2n\beta$, which holds if $k > \frac{2\sqrt{n}}{\lambda^{1/2}}$. Therefore

$$k \leq \frac{2\sqrt{n}}{\lambda^{1/2}}, \text{ and } d_A(v) = k\lambda\beta \leq 2\sqrt{n}\lambda^{1/2}\beta.$$

□

The next corollary shows that if the hypergraph is linear (every two hyperedges intersect in at most on vertex), then the degree distribution is closer. The proof is in Appendix C.1.1.

Corollary 4.6.1. *For d-uniform linear hypergraphs, the degree distribution is tighter, and $|d_A(v) - d_B(v)| = O(\log n)\lambda\beta$.*

Next we prove two lemmas regarding the structure of different HEDCSs across sampled subgraphs. The first lemma shows that edge sampling an HEDCS results in another HEDCS for the sampled subgraph. The second lemma shows that the degree distributions of any two HEDCS for two different edge sampled subgraphs of G is almost the same.

Lemma 4.6.6. *Let H be a HEDCS(G, β_H, β_H^-) for parameters $\beta_H := (1-\frac{\lambda}{\alpha})\cdot\frac{\beta}{p}$, $\beta_H^- := \beta_H - (d-1)$ and $\beta \geq 15d(\alpha d)^2 \cdot \lambda^{-2} \cdot \log n$ such that $p < 1 - \frac{2}{\alpha}$. Suppose $G_p := G_p^E(V, E_p)$ is an edge sampled subgraph of G and $H_p := H \cap G_p$; then, with high probability:*

1. *For any vertex $v \in V$: $|d_{H_p}(v) - p \cdot d_H(v)| \leq \frac{\lambda}{\alpha d}\beta$*
2. *H_p is a HEDCS of G_p with parameters $(\beta, (1-\lambda)\cdot\beta)$.*

Proof. Let $\alpha' := \alpha d$. For any vertex $v \in V$, $E[d_{H_p}(v)] = p \cdot d_H(v)$ and $d_H(v) \leq \beta_H$ by Property *(P1)* of HEDCS H. Moreover, since each edge incident upon v in H is sampled in H_p independently, by the Chernoff bound:

$$P\Big(|d_{H_p}(v) - p \cdot d_H(v)| \geq \frac{\lambda}{\alpha'}\beta\Big) \leq 2 \cdot \exp(-\frac{\lambda^2\beta}{3\cdot\alpha'^2}) \leq \frac{2}{n^{5d}} .$$

In the following, we condition on the event that:

$$|d_{H_p}(v) - p \cdot d_H(v)| \leq \frac{\lambda}{\alpha'}\beta .$$

This event happens with probability at least $1 - \frac{2}{n^{5d-1}}$ by above equation and a union bound on $|V| = n$ vertices. This finalizes the proof of the first part of the claim. We are now ready to prove that H_p is indeed am HEDCS$(G_p, \beta, (1-\lambda)\cdot\beta)$ conditioned on this event. Consider any edge $e \in H_p$. Since $H_p \subset H$, $e \in H$ as well. Hence, we have,

$$\sum_{v \in e} d_{H_p}(v) \leq p \cdot \beta_H + \frac{d\lambda}{\alpha'}\beta = (1 - \frac{\lambda}{\alpha} + \frac{d\lambda}{\alpha'})\beta = \beta\ ,$$

because $\frac{\alpha}{\alpha'} = \frac{1}{d}$, where the inequality is by Property *(P1)* of HEDCS H and the equality is by the choice of β_H. As a result, H_p satisfies Property *(P1)* of HEDCS for parameter β. Now consider an edge $e \in G_p \setminus H_p$. Since $H_p = G_p \cap H$, $e \notin H$ as well. Hence,

$$\begin{aligned}
\sum_{v \in e} d_{H_p}(v) \geq p \cdot \beta_H^- - \frac{d\lambda}{\alpha'}\beta &= (1 - \frac{\lambda}{\alpha} - \frac{d\lambda}{\alpha'})\beta - p \cdot (d-1) \\
&= (1 - \frac{2\lambda}{\alpha})\beta - p \cdot (d-1) \\
&> (1-\lambda) \cdot \beta\ .
\end{aligned}$$

□

Lemma 4.6.7. *(HEDCS in Edge Sampled Subgraph). Fix any hypergraph $G(V,E)$ and $p \in (0,1)$. Let G_1 and G_2 be two edge sampled subgraphs of G with probability p (chosen not necessarily independently). Let H_1 and H_2 be arbitrary HEDCSs of G_1 and G_2 with parameters $(\beta, (1-\lambda)\cdot\beta)$. Suppose $\beta \geq 15d(\alpha d)^2 \cdot \lambda^{-2} \cdot \log n$, then, with probability $1 - \frac{4}{n^{5d-1}}$, simultaneously for all $v \in V$: $|d_{H_1}(v) - d_{H_2}(v)| = O(n^{1/2})\lambda^{1/2}\beta$.*

Proof. Let H be an HEDCS(G, β_H, β_H^-) for the parameters β_H and β_H^- as defined in the previous lemma. The existence of H follows since $\beta_H - (d-1) \geq \beta_H^-$. Define $\hat{H}_1 := H \cap G_1$ and $\hat{H}_2 := H \cap G_2$. By Lemma 4.6.6, $\hat{H}_1$ (resp. $\hat{H}_2$) is an HEDCS of G_1 (resp. G_2) with parameters $(\beta, (1-\lambda)\beta)$ with probability $1 - \frac{4}{n^{5d-1}}$. In the following, we condition on this event. By Lemma 4.6.5 (Degree Distribution Lemma), since both H_1 (resp. H_2) and $\hat{H}_1$ (resp. $\hat{H}_2$) are HEDCSs for G_1 (resp. G_2), the degree of vertices in both of them should be "close" to each other. Moreover, since by Lemma 4.6.6 the degree of each vertex in $\hat{H}_1$ and $\hat{H}_2$ is close to p times its degree in H, we can argue that the vertex degrees in H_1 and H_2 are close. Formally, for any $v \in V$, we have

$$\begin{aligned} |d_{H_1}(v) - d_{H_2}(v)| &\leq |d_{H_1}(v) - d_{\hat{H}_1}(v)| + |d_{\hat{H}_1}(v) - d_{\hat{H}_2}(v)| + |d_{\hat{H}_2}(v) - d_{H_2}(v)| \\ &\leq O\big(n^{1/2}\big)\lambda^{1/2}\beta + |d_{\hat{H}_1}(v) - pd_H(v)| + |d_{\hat{H}_2}(v) - pd_H(v)| \\ &\leq O\big(n^{1/2}\big)\lambda^{1/2}\beta + O(1) \cdot \lambda \cdot \beta \; . \end{aligned}$$

□

Corollary 4.6.2. *If G is linear, then $|d_{H_1}(v) - d_{H_2}(v)| = O\big(\log n\big)\lambda\beta$.*

We are now ready to present a parallel algorithm that will use the HEDCS subgraph. We first compute an HEDCS in parallel via edge sampling. Let $G^{(1)}, \dots, G^{(k)}$ be a random k-partition of a graph G. We show that if we compute an arbitrary HEDCS of each graph $G^{(i)}$ (with no coordination across different graphs) and combine them together, we obtain a HEDCS for the original graph G. We then store this HEDCS in one machine and compute a maximal matching on it. We present our algorithm for all range of memory $s = n^{\Omega(1)}$. Lemma 4.6.8 and Corollary 4.6.3 serve as a proof to Theorem 4.6.2.

Algorithm 13 HEDCS-Matching(G, s): a parallel algorithm to compute a $O(d^3)$-approximation matching on a d-uniform hypergraph G with m edges on machines of memory $O(s)$

Input: A d-uniform hypergraph G, memory per machine s
Output: A matching on G

```
1: Define $k := \frac{m}{s \log n}$, $\lambda := \frac{1}{2n \log n}$ and $\beta := 500 \cdot d^3 \cdot n^2 \cdot \log^3 n$.
2: for $i = 1$ to $k$, in parallel
3:     Compute $C^{(i)} = HEDCS(G^{(i)}, \beta, (1 - \lambda) \cdot \beta)$ on machine $i$.
4: Define the multi-graph $C(V, E_C)$ with $E_C := \cup_{i=1}^{k} C^{(i)}$. This hypergraph is edge partitioned
   across the machines.
5: Compute and output a maximal matching on $C$.
```

Lemma 4.6.8. *Suppose $k \leq \sqrt{m}$. Then with high probability*

1. *The subgraph C is an HEDCS(G, β_C, β_C^-) for parameters: $\lambda_C = O\big(n^{1/2}\big)\lambda^{1/2}$, $\beta_C = (1 + d \cdot \lambda_C) \cdot k \cdot \beta$ and $\beta_C^- = (1 - \lambda - d \cdot \lambda_C) \cdot k \cdot \beta$.*

2. *The total number of edges in each subgraph $G^{(i)}$ of G is $\tilde{O}(s)$.*

3. *If $s = \tilde{O}(n\sqrt{nm})$, then the graph C can fit in the memory of one machine.*

Proof of Lemma 4.6.8.

1. Recall that each graph $G^{(i)}$ is an edge sampled subgraph of G with sampling probability $p = \frac{1}{k}$. By Lemma 4.6.7 for graphs $G^{(i)}$ and $G^{(j)}$ (for $i \neq j \in [k]$) and their HEDCSs $C^{(i)}$ and $C^{(j)}$, with probability $1 - \frac{4}{n^{5d-1}}$, for all vertices $v \in V$:

$$|d_{C^{(i)}}(v) - d_{C^{(j)}}(v)| \leq O\big(n^{1/2}\big)\lambda^{1/2}\beta \ .$$

By taking a union bound on all $\binom{k}{2} \leq n^d$ pairs of subgraphs $G^{(i)}$ and $G^{(j)}$ for $i \neq j \in [k]$, the above property holds for all $i, j \in [k]$, with probability at least $1 - \frac{4}{n^{4d-1}}$. In the following, we condition on this event.

We now prove that C is indeed a $HEDCS(G, \beta_C, \beta_C^-)$. First, consider an edge $e \in C$ and let $j \in [k]$ be such that $e \in C^{(j)}$ as well. We have

$$\begin{aligned} \sum_{v \in e} d_C(v) &= \sum_{v\in e}\sum_{i=1}^{k} d_{C^{(i)}}(v) \\ &\leq k \cdot \sum_{v \in e} d_{C^{(j)}}(v) + d \cdot k \cdot \lambda_C \cdot \beta \\ &\leq k \cdot \beta + d \cdot k \cdot \lambda_C \cdot \beta \\ &= \beta_C \ . \end{aligned}$$

Hence, C satisfies Property *(P1)* of HEDCS for parameter β_C. Now consider an edge $e \in G \setminus C$ and let $j \in [k]$ be such that $e \in G^{(j)} \setminus C^{(j)}$ (recall that each edge in G is sent to exactly one graph $G^{(j)}$ in the random k-partition). We have,

$$\begin{aligned}
\sum_{v\in e} d_C(v) &= \sum_{v\in e}\sum_{i=1}^{k} d_{C^{(i)}}(v) \\
&\geq k\cdot \sum_{v\in e} d_{C^{(j)}} - d\cdot k\lambda_C\beta \\
&\geq k\cdot(1-\lambda)\cdot\beta - d\cdot k\lambda_C\beta\ .
\end{aligned}$$

2. Let $E^{(i)}$ be the edges of $G^{(i)}$. By the independent sampling of edges in an edge sampled subgraph, we have that $E\left[|E^{(i)}|\right] = \frac{m}{k} = \tilde{O}(s)$. By Chernoff bound, with probability $1-\frac{1}{k\cdot n^{20}}$, the size of $E^{(i)}$ is $\tilde{O}(s)$. We can then take a union bound on all k machines in $G^{(i)}$ and have that with probability $1-1/n^{20}$, each graph $G^{(i)}$ is of size $\tilde{O}(s)$.

3. The number of edges in C is bounded by $n\cdot\beta_c = O(n\cdot k\cdot\beta) = \tilde{O}(\frac{n^3 m}{s}) = \tilde{O}(s)$.

□

Corollary 4.6.3. *If G is linear, then by choosing $\lambda := \frac{1}{2\log^2 n}$ and $\beta := 500\cdot d^3\cdot\log^4 n$ in HEDCS-Matching we have:*

1. *With high probability, the subgraph C is a HEDCS(G,β_C,β_C^-) for parameters: $\lambda_C = O(\log n)\lambda$, $\beta_C = (1+d\cdot\lambda_C)\cdot k\cdot\beta$ and $\beta_C^- = (1-\lambda-d\cdot\lambda_C)\cdot k\cdot\beta$.*

2. *If $s = \tilde{O}(\sqrt{nm})$ then C can fit on the memory of one machine.*

Proof of Corollary 4.6.3.

1. Similarly to Lemma 4.6.8 and by using corollary 4.6.2, we know that for graphs $G^{(i)}$ and $G^{(j)}$ (for $i \neq j \in [k]$) and their HEDCSs $C^{(i)}$ and $C^{(j)}$, with high probability , for all vertices $v \in V$:

$$|d_{C^{(i)}}(v) - d_{C^{(j)}}(v)| \leq O\big(\log n\big)\lambda\beta.$$

By taking a union bound on all $\binom{k}{2} \leq n^d$ pairs of subgraphs $G^{(i)}$ and $G^{(j)}$ for $i \neq j \in [k]$, the above property holds for all $i, j \in [k]$, with probability at least $1 - \frac{4}{n^{4d-1}}$. In the following, we condition on this event. Showing that C is indeed a $HEDCS(G, \beta_C, \beta_C^-)$ follows by the same analysis from the proof of Lemma 4.6.8.

2. The number of edges in C is bounded by $n \cdot \beta_c = O(n \cdot k \cdot \beta) = O(n \cdot k) = \tilde{O}(\frac{nm}{s}) = \tilde{O}(s)$.

□

The previous lemmas allow us to formulate the following theorem.

Theorem 4.6.2. *HEDCS-Matching constructs a HEDCS of G in 3 MPC rounds on machines of memory $s = \tilde{O}(n\sqrt{nm})$ in general and $s = \tilde{O}(\sqrt{nm})$ for linear hypergraphs.*

Corollary 4.6.4. *HEDCS-Matching achieves a $d(d-1+1/d)^2$-approximation to the d-Uniform Hypergraph Matching in 3 rounds with high probability.*

Proof of Corollary 4.6.4. We show that with high probability, C verifies the assumptions of theorem 4.6.1. From Lemma 4.6.8, we get that with high probability, the subgraph C is a $HEDCS(G, \beta_C, \beta_C^-)$ for parameters: $\lambda_C = O(n^{1/2})\lambda^{1/2}$, $\beta_C = (1 + d \cdot \lambda_C) \cdot k \cdot \beta$ and $\beta_C^- = (1 - \lambda - d \cdot \lambda_C) \cdot k \cdot \beta$. We can see that $\beta_c \geq \frac{8d^2}{d-1} \cdot \lambda_c^{-3}$. Therefore by Theorem 4.6.1, C contains a $(d-1+\frac{1}{d})$-approximate ϵ-restricted matching. Since the integrality gap of the d-UHM is at most $d-1+\frac{1}{d}$ (see [182] for details), then C contains a $(d-1+\frac{1}{d})^2$-approximate matching. Taking a maximal matching in C multiplies the approximation factor by at most d. Therefore, any maximal matching in C is a $d(d-1+\frac{1}{d})^2$-approximation. □

4.7 Computational Experiments

To understand the relative performance of the proposed algorithms, we conduct a wide variety of experiments on both random and real-life data [183, 184, 185]. We implement the three algorithms Greedy, Iterated-Sampling and HEDCS-Matching using Python, and more specifically relying on the module *pygraph* and its class *pygraph.hypergraph* to construct and perform operations on hypergraphs. We simulate the MPC model by computing a k-partitioning and splitting it into k different inputs. Parallel computations on different parts of the k-partitioning are handled through the use of the *multiprocessing* library in Python. We compute the optimal matching through an Integer Program for small instances of random uniform hypergraphs ($d \leq 10$) as well as geometric hypergraphs. The experiments were conducted on a 2.6 GHz Intel Core i7 processor and 16 GB RAM workstation. The datasets differ in their number of vertices, hyperedges, vertex degree and hyperedge cardinality. In the following tables, n and m denote the number of vertices and number of hyperedges respectively, and d is the size of hyperedges. For Table 4.4, the graphs might have different number of edges and $\bar{m}$ denotes the average number of edges. k is the number of machines used to distribute the hypergraph initially. We limit the number of edges that a machine can store to $\frac{2m}{k}$. In the columns Gr, IS and HEDCS, we store the average ratio between the size of the matching computed by the algorithms Greedy, Iterated-Sampling and HEDCS-Matching respectively, and the size of a benchmark. This ratio is computed by the percentage $\frac{ALG}{BENCHMARK}$, where ALG is the output of the algorithm, and $BENCHMARK$ denotes the size of the benchmark. The benchmarks include the size of optimal solution when it is possible to compute, or the size of a maximal matching computed via a sequential algorithm. $\#I$ denotes the number of instances of random graphs that we generated for fixed n, m and d. β and β^- are the parameters used to construct the HEDCS subgraphs in HEDCS-Matching. These subgraphs are constructed using the procedure in the proof of Lemma 4.6.1.

4.7.1 Experiments with random hypergraphs

We perform experiments on two classes of d-uniform random hypergraphs. The first contains random uniform hypergraphs, and the second contains random geometric hypergraphs.

Random Uniform Hypergraphs. For a fixed n, m and d, each potential hyperedge is sampled independently and uniformly at random from the set of vertices. In Table 4.2, we use the size of a perfect matching $\frac{n}{d}$ as a benchmark, because a perfect matching in random graphs exists with probability $1 - o(1)$ under some conditions on m, n and d. If $d(n, m) = \frac{m \cdot d}{n}$ is the expected degree of a random uniform hypergraph, Frieze and Janson [186] showed that $\frac{d(n,m)}{n^{1/3}} \to \infty$ is a sufficient condition for the hypergraph to have a perfect matching with high probability. Kim [187] further weakened this condition to $\frac{d(n,m)}{n^{1/(5+2/(d-1))}} \to \infty$. We empirically verify, by solving the IP formulation, that for $d = 3, 5$ and for small instances of $d = 10$, our random graphs contain a perfect matching. In Table 4.2, the benchmark is the size of a perfect matching, while in Table 4.3, it is the size of a greedy maximal matching. In terms of solution quality (Tables 4.2 and 4.3) HEDCS-Matching performs consistently better than Greedy, and Iterated-Sampling performs significantly better than the other two. None of the three algorithms are capable of finding perfect matchings for a significant number of the runs. When compared to the size of a maximal matching, Iterated-Sampling still performs better, followed by HEDCS-Matching. However, the ratio is smaller when compared to a maximal matching, which is explained by the deterioration of the quality of greedy maximal matching as n and d grow. Dufosse *et al.* [188] confirm that the approximation quality of a greedy maximal matching on random graphs that contain a perfect matching degrades as a function of n and d. The performance of the algorithms decreases as d grows, which is theoretically expected since their approximations ratio are both proportional to d. The number of rounds for Iterated-Sampling grows slowly with n, which is consistent with $O(\log n)$ bound. Recall that the number of rounds for the other two algorithms is constant and equal to 3.

Geometric Random Hypergraphs. The second class we experimented on is random geometric hypergraphs. The vertices of a random geometric hypergraph (RGH) are randomly sampled from the uniform distribution of the space $[0, 1)^2$. A set of d different vertices $v_1, \ldots, v_d \in V$ forms a hyperedge if, and only if, the distance between any v_i and v_j is less than a previously specified parameter $r \in (0, 1)$. The parameters r and n fully characterize a RGH. We fix $d = 3$ and generate different geometric hypergraphs by varying n and r. We compare the output of the algorithms to the optimal solution that we compute through the IP formulation. Table 4.4 shows that the performance of our three algorithms is almost similar with Iterated-Sampling outperforming Greedy and HEDCS-Matching as the size of the graphs grows. We also observe that random geometric hypergraphs do not contain perfect matchings, mainly because of the existence of some outlier vertices that do not belong to any edge. The number of rounds of Iterated-Sampling still grows with n, confirming the theoretical bound and the results on random uniform hypergraphs.

n	m	d	k	#I	Gr	IS	HEDCS	β	β^-	Rounds IS
15	200	3	5	500	77.6%	86.6%	82.8%	5	3	3.8
30	400		5		78.9%	88.1%	80.3%	7	4	4.56
100	3200		10		81.7%	93.4%	83.1%	5	2	5.08
300	4000		10		78.8%	88.7%	80.3%	8	6	7.05
50	800	5	6	500	66.0%	76.2%	67.0%	16	11	4.89
100	2,800		10		68.0%	79.6%	69.8%	16	11	4.74
300	4,000		10		62.2%	75.1%	65.5%	10	5	6.62
500	8,000		16		63.3%	76.4%	65.6%	10	5	7.62
500	15,000	10	16	500	44.9%	58.3%	53.9%	20	10	6.69
1,000	50,000		20		47.3%	61.3%	50.5%	20	10	8.25
2,500	100,000		20		45.6%	59.9%	48.2%	20	10	8.11
5,000	200,000		20		45.0%	59.7%	47.8%	20	10	7.89
1,000	50,000	25	25	100	27.5%	34.9%	30.8%	75	50	8.10
2,500	100,000		25		26.9%	34.0%	27.0%	75	50	8.26
5,000	250,000		30		26.7%	33.8%	28.8%	75	50	8.23
10,000	500,000		30		26.6%	34.1%	28.2%	75	50	8.46
5,000	250,000	50	30	100	22.4%	30.9%	27.9%	100	50	10.22
10,000	500,000		30		22.2%	31.0%	26.5%	100	50	10.15
15,000	750,000		30		20.9%	30.8%	26.4%	100	50	10.26
25,000	1,000,000		30		20.9%	30.8%	26.4%	100	50	10.29

Table 4.2: Comparison on random instances with perfect matching benchmark, of size $\frac{n}{d}$.

n	m	d	k	#I	Gr	IS	HEDCS	β	λ
15	200	3	5	500	79.1%	87.5%	82.3%	5	3
30	400		5		82.6%	91.3%	84.4%	7	4
100	3200		10		83.9%	96.2%	88.2%	5	2
300	4000		10		81.1%	92.0%	86.3%	4	2
50	800	5	6	500	76.0%	89.1%	78.5%	16	11
100	2,800		10		77.9%	92.1%	81.9%	16	11
300	4,000		10		77.8%	93.9%	87.1%	10	5
500	8,000		16		79.2%	94.3%	85.9%	10	5
500	15,000	10	16	500	71.8%	90.6%	79.2%	20	10
1,000	50,000		20		73.8%	92.3%	81.5%	20	10
2,500	100,000		20		72.2%	91.5%	80.7%	20	10
5,000	200,000		20		72.5%	90.7%	79.8%	20	10
1,000	5,0000	25	20	100	68.2%	87.5%	75.6%	75	50
2,500	100,000		25		69.0%	87.9%	74.3%	75	50
5,000	250,000		30		67.8%	87.3%	75.1%	75	50
10,000	500,000		30		67.2%	86.9%	73.7%	75	50
5,000	250,000	50	30	100	67.4%	86.6%	74.0%	100	50
10,000	500,000		30		68.1%	87.1%	73.4%	100	50
15,000	750,000		30		66.9%	86.2%	73.2%	100	50
25,000	1,000,000		30		67.3%	86.0%	72.8%	100	50

Table 4.3: Comparison on random uniform instances with maximal matching benchmark.

n	r	$\bar{m}$	d	k	#I	Gr	IS	HEDCS	β	β^-	Rounds IS
100	0.2	930.1 ± 323	3	5	100	88.3%	89.0%	89.6%	3	5	4.1
100	0.25	1329.5 ± 445		10		88.0%	89.0%	89.5%	3	5	5.2
250	0.15	13222± 3541		5		85.0%	88.6%	85.5%	4	7	8.1
300	0.15	14838 ± 4813		10		85.5%	88.0%	85.2%	4	7	11.1
300	0.2	27281 ± 8234		10		85.0%	89.0%	86.3%	4	7	13.5

Table 4.4: Comparison on random geometric hypergraphs with optimal matching benchmark.

4.7.2 Experiments with real data

PubMed and Cora Datasets. We employ two citation network datasets, namely the Cora and Pubmed datasets [183, 184]. These datasets are represented with a graph, with vertices being publications, and edges being a citation links from one article to another. We construct the hypergraphs in two steps 1) each article is a vertex; 2) each article is taken as a centroid and forms a hyperedge to connect those articles which have citation links to it (either citing it or being cited). The Cora hypergraph has an average edge size of 3.0 ± 1.1, while the average in the Pubmed hypergraph is 4.3 ± 5.7. The number of edges in both is significantly smaller than the number of vertices, therefore we allow each machine to store only $\frac{m}{k} + \frac{1}{4}\frac{m}{k}$ edges. We randomly split the edges on each machine, and because the subgraphs are small, we are able to compute the optimal matchings on each machine, as well as on the whole hypergraphs. We perform ten runs of each algorithm with different random k-partitioning and take the maximum cardinality obtained. Table 4.5 shows that none of the algorithms is able to retrieve the optimal matching. This behaviour can be explained by the loss of information that using parallel machines implies. We see that Iterated-Sampling, like in previous experiments, outperforms the other algorithms due to highly sequential design. HEDCS-Matching particularly performs worse than the other algorithms, mainly because it fails to construct sufficiently large HEDCSs.

Social Network Communities. We include two larger real-world datasets, orkut-groups and Live-

Journal, from the Koblenz Network Collection [185]. We use two hypergraphs that were constructed from these datasets by Shun [189]. Vertices represent individual users, and hyperedges represent communities in the network. Because membership in these communities does not require the same commitment as collaborating on academic research, these hypergraphs have different characteristics from co-citation hypergraphs, in terms of size, vertex degree and hyperedge cardinality. We use the size of a maximal matching as a benchmark. Table 4.5 shows that **Iterated-Sampling** still provides the best approximation. **HEDCS-Sampling** performs worse than **Greedy** on Livejournal, mainly because the ratio $\frac{m}{n}$ is not big enough to construct an HEDCS with a large matching.

Name	n	m	k	Gr	IS	HEDCS	Rounds IS
Cora	2,708	1,579	2	75.0%	83.2%	63.9%	6
PubMed	19,717	7,963	3	72.0%	86.6%	62.4%	9
Orkut	2,32 $\times 10^6$	1,53 $\times 10^7$	10	55.6%	66.1%	58.1%	11
Livejournal	3,20 $\times 10^6$	7,49 $\times 10^6$	3	44.0%	55.2%	43.3%	10

Table 4.5: Comparison on co-citation and social network hypergraphs.

4.7.3 Experimental conclusions

In the majority of our experiments, Iterated-Sampling provides the best approximation to the d-UHM problem, which is consistent with its theoretical superiority. On random graphs, HEDCS-Matching performs consistently better than Greedy, even-though Greedy has a better theoretical approximation ratio. We suspect it is because the $O(d^3)$-approximation bound on HEDCS-Matching is loose. We conjecture that rounding an $\epsilon-$restricted matching can be done efficiently, which would improve the approximation ratio. The performance of the three algorithms decreases as d grows. The results on the number of rounds of Iterated-Sampling also are consistent with the theoretical bound. However, due to its sequential design, and by centralizing the computation on one single machine while using the other machines simply to coordinate, Iterated-Sampling not only takes more rounds than the other two algorithms, but is also slower when we account for the absolute runtime as well as the runtime per round. We compared the runtimes of our three algorithms on a set of random uniform hypergraphs. Figure 4.1 and 4.2 show that the absolute and per round run-times of Iterated-Sampling grow considerably faster with the size of the hypergraphs. We can also see that HEDCS-Sampling is slower than Greedy, since the former performs heavier computation on each machine. This confirms the trade-off between the extent to which the algorithms use the power of parallelism and the quality of the approximations.

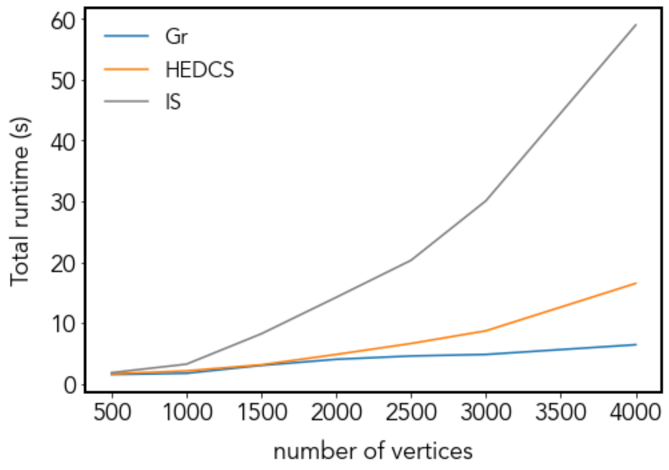

Figure 4.1: Runtime of the three algorithms when $d = 3$ and $m = 20 \cdot d \cdot n$

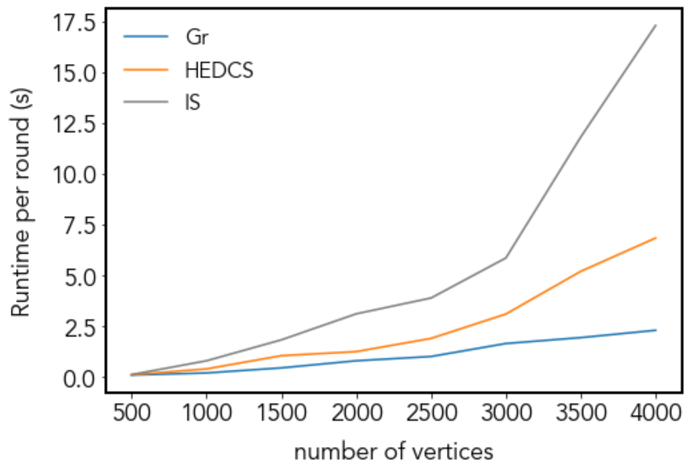

Figure 4.2: Total runtime per round (maximum over all machines in every round) of the three algorithms when $d = 3$ and $m = 20 \cdot d \cdot n$

4.8 Conclusion

In this chapter, we have presented the first algorithms for the d-UHM problem in the MPC model. Our theoretical and experimental results highlight the trade-off between the approximation ratio, the necessary memory per machine, and the number of rounds it takes to run the algorithm. We have also introduced the notion of HEDCS subgraphs, and have shown that an HEDCS contains a good approximation for the maximum matching and that they can be constructed in a few rounds in the MPC model. We believe better approximation algorithms should be possible, especially if we can give better rounding algorithms for a ϵ-restricted fractional hypergraph matching. For future work, it would be interesting to explore whether we can achieve better-than-d approximation in the MPC model in a polylogarithmic number of rounds. Exploring algorithms relying on vertex sampling instead of edge sampling might be a good candidate. In addition, our analysis in this paper is specific to unweighted hypergraphs, and we would like to extend this to weighted hypergraphs.

Conclusion

This thesis focuses on a set of novel questions in the theory of matchings. At a high level, we address three challenging settings in which identifying or computing the optimal matching is either NP-complete or computationally expensive. Our contributions in this thesis are threefold: the first is to establish hardness results or computational lower bound requirements to solve the problems at hand. The second contribution is to design new algorithms that are tractable, scalable and with theoretical guarantees for each setting. Our third type of contribution is to support our theoretical findings with extensive numerical experiments on real-life data.

In Chapter 2, we consider a two-stage robust optimization framework that captures matching problems where one side of the input includes some future demand uncertainty. We propose two models to capture the demand uncertainty: explicit, where all possible second-stage scenarios are listed explicitly, and implicit, where the scenarios are defined by using a budgeted constraint. We present a constant approximation algorithm for the explicit case with any fixed number of scenarios. Our approximation does not depend on the number of first-stage riders or the size of scenarios but scales with the number of scenarios. For the implicit model of uncertainty, where the scenarios can be exponentially many, our analysis depends on the imbalance between supply and demand. We show that under a reasonable assumption on the size of scenarios, there is a tractable constant approximation algorithm.

In Chapters 3 and 4, we focus on the generalization matchings to hypergraphs. In Chapter 3, we consider the problem of learning hidden hypergraph matchings through membership queries. In this problem, the learner is given the vertex set of a hidden hypergraph and an edge-detecting oracle. The oracle answers yes if and only if the queried set of vertices contains an edge, and the goal is to learn the edge set of the hypergraph using a small number of queries. While previous attempts to learn general hypergraphs have been shown to necessitate an exponential number of queries, we identify non-trivial families of hypergraphs that can be learned without suffering from an exponential query complexity. We focus on learning hypermatchings and give a poly-logarithmic -round algorithm with an almost linear number of queries. We complement this upper bound by showing that there are no algorithms with polynomially many queries that learn hypermatchings in a constant number of adaptive rounds. Contrary to previous work that relied on constructing a collection of sets of vertices that aim to identify non-edges, we construct a unique-edge covering family: a collection of sets of vertices such that every edge of the hidden hypergraph is contained in at least one set of the collection. This new concept allows us to bypass the exponential complexity in the case of matchings and low degree hypergraphs.

Finally, in Chapter 4, we study the problem of finding matchings in uniform hypergraphs in the massively parallel computation (MPC) model where the data (e.g., vertices and edges) is distributed across the machines and in each round, a machine performs local computation on its fragment of data, and then sends messages to other machines for the next round. This problem generalizes maximum matchings in graphs. We design and implement the first algorithms for this problem in the MPC model. We give three different algorithms, demonstrating different trade-offs between the model's parameters. Our algorithms are inspired by methods to find maximum matchings in graphs, but require developing significant new tools to address hypergraphs. In particular, for the third algorithm, we introduce the concept of HyperEdge Degree Constrained Subgraph (HEDCS), and we show that an HEDCS contains a large fractional matching and that it can be efficiently computed in parallel. Moreover, we investigate the experimental performance of these algorithms both on random input and real instances. Our results support the theoretical bounds and confirm the trade-offs between the quality of approximation and the speed of the algorithms.

References

[1] F. Schalekamp, D. P. Williamson, and A. van Zuylen, “2-matchings, the traveling salesman problem, and the subtour lp: A proof of the boyd-carr conjecture,” *Mathematics of Operations Research*, vol. 39, no. 2, pp. 403–417, 2014.

[2] A. H. Timmer and J. A. Jess, “Exact scheduling strategies based on bipartite graph matching,” in *Proceedings the European Design and Test Conference. ED&TC 1995*, IEEE, 1995, pp. 42–47.

[3] M. Riedel, “Online matching for scheduling problems,” in *Annual Symposium on Theoretical Aspects of Computer Science*, Springer, 1999, pp. 571–580.

[4] S. Albers, “Online algorithms: A survey,” *Mathematical Programming*, vol. 97, no. 1, pp. 3–26, 2003.

[5] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. cambridge university press, 2005.

[6] S. Lattanzi, S. Mitrović, A. Norouzi-Fard, J. M. Tarnawski, and M. Zadimoghaddam, “Fully dynamic algorithm for constrained submodular optimization,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 923–12 933, 2020.

[7] A. Bernstein and C. Stein, “Fully dynamic matching in bipartite graphs,” in *International Colloquium on Automata, Languages, and Programming*, Springer, 2015, pp. 167–179.

[8] D. Bertsimas, D. B. Brown, and C. Caramanis, “Theory and applications of robust optimization,” *SIAM review*, vol. 53, no. 3, pp. 464–501, 2011.

[9] D. Bertsimas and V. Goyal, “On the power and limitations of affine policies in two-stage adaptive optimization,” *Mathematical programming*, vol. 134, no. 2, pp. 491–531, 2012.

[10] A. Ben-Tal, O. El Housni, and V. Goyal, “A tractable approach for designing piecewise affine policies in two-stage adjustable robust optimization,” *Mathematical Programming*, vol. 182, no. 1, pp. 57–102, 2020.

[11] J. F. JaJa, “Pram (parallel random access machines),” in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Boston, MA: Springer US, 2011, pp. 1608–1615, ISBN: 978-0-387-09766-4.

[12] A. Israeli and A. Itai, “A fast and simple randomized parallel algorithm for maximal matching,” *Information Processing Letters*, vol. 22, no. 2, pp. 77–80, 1986.

[13] N. Alon, L. Babai, and A. Itai, “A fast and simple randomized parallel algorithm for the maximal independent set problem,” *Journal of algorithms*, vol. 7, no. 4, pp. 567–583, 1986.

[14] M. Luby, “A simple parallel algorithm for the maximal independent set problem,” *SIAM journal on computing*, vol. 15, no. 4, pp. 1036–1053, 1986.

[15] D. Peleg, *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

[16] N. Linial, “Distributive graph algorithms global solutions from local data,” in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, IEEE, 1987, pp. 331–335.

[17] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[18] O. E. Housni, V. Goyal, O. Hanguir, and C. Stein, “Matching drivers to riders: A two-stage robust approach,” in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[19] E. Balkanski, O. Hanguir, and S. Wang, “Learning low degree hypergraphs,” *arXiv preprint arXiv:2202.09989*, 2022.

[20] O. Hanguir and C. Stein, “Distributed algorithms for matching in hypergraphs,” in *International Workshop on Approximation and Online Algorithms*, Springer, 2020, pp. 30–46.

[21] R. M. Karp, “Reducibility among combinatorial problems,” in *Complexity of computer computations*, Springer, 1972, pp. 85–103.

[22] F. L. Hitchcock, “The distribution of a product from several sources to numerous localities,” *Journal of mathematics and physics*, vol. 20, no. 1-4, pp. 224–230, 1941.

[23] T. E. Easterfield, “A combinatorial algorithm,” *Journal of the London Mathematical Society*, vol. 1, no. 3, pp. 219–226, 1946.

[24] A. A. Bertossi, P. Carraresi, and G. Gallo, “On some matching problems arising in vehicle scheduling models,” *Networks*, vol. 17, no. 3, pp. 271–281, 1987.

[25] M. Schreieck, H. Safetli, S. A. Siddiqui, C. Pflügler, M. Wiesche, and H. Krcmar, “A matching algorithm for dynamic ridesharing,” *Transportation Research Procedia*, vol. 19, pp. 272–285, 2016.

[26] A. G. Fowler, “Minimum weight perfect matching of fault-tolerant topological quantum error correction in average o(1) parallel time,” *arXiv preprint arXiv:1307.1740*, 2013.

[27] Y. Faenza, S. Gupta, and X. Zhang, "Discovering opportunities in new york city's discovery program: An analysis of affirmative action mechanisms," *arXiv preprint arXiv:2203.00544*, 2022.

[28] Y. Faenza, V. Powers, and X. Zhang, "Two-sided popular matchings in bipartite graphs with forbidden/forced elements and weights," *arXiv preprint arXiv:1803.01478*, 2018.

[29] D. Vukičević, "Applications of perfect matchings in chemistry," in *Structural Analysis of Complex Networks*, Springer, 2011, pp. 463–482.

[30] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2.

[31] W.-Y. Kim and A. C. Kak, "3-d object recognition using bipartite matching embedded in discrete relaxation," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 13, no. 03, pp. 224–251, 1991.

[32] P. E. John, H. Sachs, and M. Zheng, "Kekulé patterns and clar patterns in bipartite plane graphs," *Journal of chemical information and computer sciences*, vol. 35, no. 6, pp. 1019–1021, 1995.

[33] A. Azad and A. Pothen, "Multithreaded algorithms for matching in graphs with application to data analysis in flow cytometry," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, IEEE, 2012, pp. 2494–2497.

[34] J. E. Hopcroft and R. M. Karp, "An n^5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[35] A. V. Karzanov, "An exact estimate of an algorithm for finding a maximum flow, applied to the problem on representatives," *Problems in Cybernetics*, vol. 5, pp. 66–70, 1973.

[36] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[37] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of mathematics*, vol. 17, pp. 449–467, 1965.

[38] Y.-Q. Cheng, V. Wu, R. Collins, A. R. Hanson, and E. M. Riseman, "Maximum-weight bipartite matching technique and its application in image feature matching," in *Visual Communications and Image Processing'96*, International Society for Optics and Photonics, vol. 2727, 1996, pp. 453–462.

[39] S. Ólafsson, "Weighted matching in chess tournaments," *Journal of the Operational Research Society*, vol. 41, no. 1, pp. 17–24, 1990.

[40] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[41] J. Edmonds, “Maximum matching and a polyhedron with 0, 1-vertices,” *Journal of research of the National Bureau of Standards B*, vol. 69, no. 125-130, pp. 55–56, 1965.

[42] H. N. Gabow, “Data structures for weighted matching and nearest common ancestors with linking,” in *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, 1990, pp. 434–443.

[43] H. N. Gabow and R. E. Tarjan, “Faster scaling algorithms for general graph matching problems,” *Journal of the ACM (JACM)*, vol. 38, no. 4, pp. 815–853, 1991.

[44] R. Duan and S. Pettie, “Linear-time approximation for maximum weight matching,” *Journal of the ACM (JACM)*, vol. 61, no. 1, pp. 1–23, 2014.

[45] M. Cygan, H. N. Gabow, and P. Sankowski, “Algorithmic applications of baur-strassen’s theorem: Shortest cycles, diameter, and matchings,” *Journal of the ACM (JACM)*, vol. 62, no. 4, pp. 1–30, 2015.

[46] I. Shames, A. Dostovalova, J. Kim, and H. Hmam, “Task allocation and motion control for threat-seduction decoys,” in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, IEEE, 2017, pp. 4509–4514.

[47] R. S. Garfinkel, “An improved algorithm for the bottleneck assignment problem,” *Operations Research*, vol. 19, no. 7, pp. 1747–1751, 1971.

[48] U. Derigs and U. Zimmermann, “An augmenting path method for solving linear bottleneck assignment problems,” *Computing*, vol. 19, no. 4, pp. 285–295, 1978.

[49] H. N. Gabow and R. E. Tarjan, “Algorithms for two bottleneck optimization problems,” *Journal of Algorithms*, vol. 9, no. 3, pp. 411–417, 1988.

[50] A. P. Punnen and K. Nair, “Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem,” *Discrete Applied Mathematics*, vol. 55, no. 1, pp. 91–93, 1994.

[51] M. Khoo, T. A. Wood, C. Manzie, and I. Shames, “A distributed augmenting path approach for the bottleneck assignment problem,” *arXiv preprint arXiv:2011.09606*, 2020.

[52] V. Kaibel and M. Peinhardt, “On the bottleneck shortest path problem,” 2006.

[53] P. Bose, A. Maheshwari, G. Narasimhan, M. Smid, and N. Zeh, “Approximating geometric bottleneck shortest paths,” *Computational Geometry*, vol. 29, no. 3, pp. 233–249, 2004.

[54] R. S. Garfinkel and K. Gilbert, "The bottleneck traveling salesman problem: Algorithms and probabilistic analysis," *Journal of the ACM (JACM)*, vol. 25, no. 3, pp. 435–448, 1978.

[55] D. S. Hochbaum and D. B. Shmoys, "A unified approach to approximation algorithms for bottleneck problems," *Journal of the ACM (JACM)*, vol. 33, no. 3, pp. 533–550, 1986.

[56] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, 1994.

[57] R. Dochow, *Online algorithms for the portfolio selection problem*. Springer, 2016.

[58] A. Mehta and D. Panigrahi, "Online matching with stochastic rewards," in *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, IEEE, 2012, pp. 728–737.

[59] I. Ashlagi, M. Burq, C. Dutta, P. Jaillet, A. Saberi, and C. Sholley, "Maximum weight online matching with deadlines," *arXiv preprint arXiv:1808.03526*, 2018.

[60] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, "An optimal algorithm for on-line bipartite matching," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, 1990, pp. 352–358.

[61] B. Birnbaum and C. Mathieu, "On-line bipartite matching made simple," *Acm Sigact News*, vol. 39, no. 1, pp. 80–87, 2008.

[62] A. Mehta, "Online matching and ad allocation," 2013.

[63] N. Buchbinder, K. Jain, and J. S. Naor, "Online primal-dual algorithms for maximizing ad-auctions revenue," in *European Symposium on Algorithms*, Springer, 2007, pp. 253–264.

[64] N. R. Devanur and T. P. Hayes, "The adwords problem: Online keyword matching with budgeted bidders under random permutations," in *Proceedings of the 10th ACM conference on Electronic commerce*, 2009, pp. 71–78.

[65] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani, "Adwords and generalized online matching," *Journal of the ACM (JACM)*, vol. 54, no. 5, 22–es, 2007.

[66] G. Aggarwal, G. Goel, C. Karande, and A. Mehta, "Online vertex-weighted bipartite matching and single-bid budgeted allocations," in *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, SIAM, 2011, pp. 1253–1264.

[67] N. R. Devanur, K. Jain, and R. D. Kleinberg, "Randomized primal-dual analysis of ranking for online bipartite matching," in *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, SIAM, 2013, pp. 101–107.

[68] B. Haeupler, V. S. Mirrokni, and M. Zadimoghaddam, "Online stochastic weighted matching: Improved approximation algorithms," in *International workshop on internet and network economics*, Springer, 2011, pp. 170–181.

[69] N. Korula and M. Pál, "Algorithms for secretary problems on graphs and hypergraphs," in *International Colloquium on Automata, Languages, and Programming*, Springer, 2009, pp. 508–520.

[70] J. Feldman, A. Mehta, V. Mirrokni, and S. Muthukrishnan, "Online stochastic matching: Beating 1-1/e," in *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, IEEE, 2009, pp. 117–126.

[71] V. H. Manshadi, S. O. Gharan, and A. Saberi, "Online stochastic matching: Online actions based on offline statistics," *Mathematics of Operations Research*, vol. 37, no. 4, pp. 559–573, 2012.

[72] A. Mehta, B. Waggoner, and M. Zadimoghaddam, "Online stochastic matching with unequal probabilities," in *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, SIAM, 2014, pp. 1388–1404.

[73] M. Feldman, O. Svensson, and R. Zenklusen, "Online contention resolution schemes," in *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, SIAM, 2016, pp. 1014–1033.

[74] G. Goel and A. Mehta, "Online budgeted matching in random input models with applications to adwords.," in *SODA*, vol. 8, 2008, pp. 982–991.

[75] C. Karande, A. Mehta, and P. Tripathi, "Online bipartite matching with unknown distributions," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 2011, pp. 587–596.

[76] M. Mahdian and Q. Yan, "Online bipartite matching with random arrivals: An approach based on strongly factor-revealing lps," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 2011, pp. 597–606.

[77] P. Jaillet and X. Lu, "Online stochastic matching: New algorithms with better bounds," *Mathematics of Operations Research*, vol. 39, no. 3, pp. 624–646, 2014.

[78] E. Lee and S. Singla, "Maximum matching in the online batch-arrival model," in *International Conference on Integer Programming and Combinatorial Optimization*, Springer, 2017, pp. 355–367.

[79] Y. Feng and R. Niazadeh, "Batching and optimal multi-stage bipartite allocations," *Chicago Booth Research Paper*, no. 20-29, 2020.

[80] L. Zhang *et al.*, “A taxi order dispatch model based on combinatorial optimization,” in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 2151–2159.

[81] S. Khuller, S. G. Mitchell, and V. V. Vazirani, “On-line algorithms for weighted bipartite matching and stable marriages,” *Theoretical Computer Science*, vol. 127, no. 2, pp. 255–267, 1994.

[82] B. Kalyanasundaram and K. Pruhs, “Online weighted matching,” *Journal of Algorithms*, vol. 14, no. 3, pp. 478–488, 1993.

[83] A. Meyerson, A. Nanavati, and L. Poplawski, “Randomized online algorithms for minimum metric bipartite matching,” in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, Society for Industrial and Applied Mathematics, 2006, pp. 954–959.

[84] N. Bansal, N. Buchbinder, A. Gupta, and J. S. Naor, “An o(log k2)-competitive algorithm for metric bipartite matching,” in *European Symposium on Algorithms*, Springer, 2007, pp. 522–533.

[85] S. Raghvendra, “A robust and optimal online algorithm for minimum metric bipartite matching,” in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[86] N. Aslani, O. Kuzgunkaya, N. Vidyarthi, and D. Terekhov, “A robust optimization model for tactical capacity planning in an outpatient setting,” *Health Care Management Science*, vol. 24, no. 1, pp. 26–40, 2021.

[87] K. Dhamdhere, V. Goyal, R Ravi, and M. Singh, “How to pay, come what may: Approximation algorithms for demand-robust covering problems,” in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS’05)*, IEEE, 2005, pp. 367–376.

[88] U. Feige, K. Jain, M. Mahdian, and V. Mirrokni, “Robust combinatorial optimization with exponential scenarios,” in *International Conference on Integer Programming and Combinatorial Optimization*, Springer, 2007, pp. 439–453.

[89] O. El Housni and V. Goyal, “On the optimality of affine policies for budgeted uncertainty sets,” *Mathematics of Operations Research*, vol. 46, no. 2, pp. 674–711, 2021.

[90] O. E. Housni, V. Goyal, and D. Shmoys, “On the power of static assignment policies for robust facility location problems,” in *International Conference on Integer Programming and Combinatorial Optimization*, Springer, 2021, pp. 252–267.

[91] J. R. Birge and F. Louveaux, *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[92] O. El Housni and V. Goyal, "Beyond worst-case: A probabilistic analysis of affine policies in dynamic optimization," in *Advances in neural information processing systems*, 2017, pp. 4756–4764.

[93] A. Gupta, V. Nagarajan, and R. Ravi, "Thresholded covering algorithms for robust and max-min optimization," in *International Colloquium on Automata, Languages, and Programming*, Springer, 2010, pp. 262–274.

[94] O. Baron, J. Milner, and H. Naseraldin, "Facility location: A robust optimization approach," *Production and Operations Management*, vol. 20, no. 5, pp. 772–785, 2011.

[95] A. Atamtürk and M. Zhang, "Two-stage robust network flow and design under demand uncertainty," *Operations Research*, vol. 55, no. 4, pp. 662–673, 2007.

[96] I. Katriel, C. Kenyon-Mathieu, and E. Upfal, "Commitment under uncertainty: Two-stage stochastic matching problems," *Theoretical Computer Science*, vol. 408, no. 2-3, pp. 213–223, 2008.

[97] B. Escoffier, L. Gourvès, J. Monnot, and O. Spanjaard, "Two-stage stochastic matching and spanning tree problems: Polynomial instances and approximation," *European Journal of Operational Research*, vol. 205, no. 1, pp. 19–30, 2010.

[98] J. Matuschke, U. Schmidt-Kraepelin, and J. Verschae, "Maintaining perfect matchings at low cost," *arXiv preprint arXiv:1811.10580*, 2018.

[99] B. Heintz and A. Chandra, "Enabling scalable social group analytics via hypergraph analysis systems," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[100] M. Karbstein, *Line planning and connectivity*. 2013.

[101] R. Borndörfer, M. Reuther, T. Schlechte, and S. Weider, "Vehicle rotation planning for intercity railways," 2012.

[102] D. Gunopulos, H. Mannila, R. Khardon, and H. Toivonen, "Data mining, hypergraph transversals, and machine learning," in *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1997, pp. 209–216.

[103] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3558–3565.

[104] S. Bai, F. Zhang, and P. H. Torr, "Hypergraph convolution and hypergraph attention," *Pattern Recognition*, vol. 110, p. 107 637, 2021.

[105] C. Flamm, B. M. Stadler, and P. F. Stadler, "Generalized topologies: Hypergraphs, chemical reactions, and biological evolution," in *Advances in Mathematical Chemistry and Applications*, Elsevier, 2015, pp. 300–328.

[106] C. Özturan, "On finding hypercycles in chemical reaction networks," *Applied Mathematics Letters*, vol. 21, no. 9, pp. 881–884, 2008.

[107] S. Feng *et al.*, "Hypergraph models of biological networks to identify genes critical to pathogenic viral response," *BMC bioinformatics*, vol. 22, no. 1, pp. 1–21, 2021.

[108] D. Di *et al.*, "Hypergraph learning for identification of covid-19 with ct imaging," *Medical Image Analysis*, vol. 68, p. 101 910, 2021.

[109] E. Hazan, S. Safra, and O. Schwartz, "On the complexity of approximating k-set packing," *computational complexity*, vol. 15, no. 1, pp. 20–39, 2006.

[110] P. Beame, P. Koutris, and D. Suciu, "Communication steps for parallel query processing," in *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, ACM, 2013, pp. 273–284.

[111] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii, "Filtering: A method for solving graph problems in mapreduce," in *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, ACM, 2011, pp. 85–94.

[112] K. J. Ahn and S. Guha, "Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints," *ACM Transactions on Parallel Computing (TOPC)*, vol. 4, no. 4, p. 17, 2018.

[113] S. Assadi and S. Khanna, "Randomized composable coresets for matching and vertex cover," in *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, 2017, pp. 3–12.

[114] S. Assadi, M. Bateni, A. Bernstein, V. Mirrokni, and C. Stein, "Coresets meet edcs: Algorithms for matching and vertex cover on massive graphs," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2019, pp. 1616–1635.

[115] M. Ghaffari, T. Gouleakis, C. Konrad, S. Mitrović, and R. Rubinfeld, "Improved massively parallel computation algorithms for mis, matching, and vertex cover," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, 2018, pp. 129–138.

[116] N. J. Harvey, C. Liaw, and P. Liu, "Greedy and local ratio algorithms in the mapreduce model," in *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, 2018, pp. 43–52.

[117] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for mapreduce," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, SIAM, 2010, pp. 938–948.

[118] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev, "Parallel algorithms for geometric graph problems," in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, ACM, 2014, pp. 574–583.

[119] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani, "Fast greedy algorithms in mapreduce and streaming," *ACM Transactions on Parallel Computing (TOPC)*, vol. 2, no. 3, pp. 1–22, 2015.

[120] V. Mirrokni and M. Zadimoghaddam, "Randomized composable core-sets for distributed submodular maximization," in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, ACM, 2015, pp. 153–162.

[121] Uber, *Uber marketplace and matching*, `https://marketplace.uber.com/matching`, 2020.

[122] Lyft, *Matchmaking in lyft line - part 1*, `https://eng.lyft.com/matchmaking-in-lyft-line-9c2635fe62c4`, 2016.

[123] D. Bertsimas and M. Sim, "The price of robustness," *Operations research*, vol. 52, no. 1, pp. 35–53, 2004.

[124] O. E. Housni and V. Goyal, "On the optimality of affine policies for budgeted uncertainty sets," *arXiv preprint arXiv:1807.00163*, 2018.

[125] B. Cheng *et al.*, "Stl: Online detection of taxi trajectory anomaly based on spatial-temporal laws," in *International Conference on Database Systems for Advanced Applications*, Springer, 2019, pp. 764–779.

[126] N. Kong and A. J. Schaefer, "A factor 12 approximation algorithm for two-stage stochastic matching problems," *European Journal of Operational Research*, vol. 172, no. 3, pp. 740–746, 2006.

[127] Y. Feng, R. Niazadeh, and A. Saberi, "Two-stage stochastic matching with application to ride hailing," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SIAM, 2021, pp. 2862–2877.

[128] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[129] V. Kann, "Maximum bounded 3-dimensional matching is max snp-complete," *Information Processing Letters*, vol. 37, no. 1, pp. 27–35, 1991.

[130] V. Nagarajan, B. Schieber, and H. Shachnai, "The euclidean k-supplier problem," in *International Conference on Integer Programming and Combinatorial Optimization*, Springer, 2013, pp. 290–301.

[131] S. Khuller, R. Pless, and Y. J. Sussmann, "Fault tolerant k-center problems," *Theoretical Computer Science*, vol. 242, no. 1-2, pp. 237–245, 2000.

[132] D. C. Torney, "Sets pooling designs," *Annals of Combinatorics*, vol. 3, no. 1, pp. 95–101, 1999.

[133] D. Angluin and J. Chen, "Learning a hidden hypergraph," *Journal of Machine Learning Ressearch*, vol. 7, 2215–2236, Dec. 2006.

[134] ——, "Learning a hidden graph using o(log n) queries per edge," in *International Conference on Computational Learning Theory*, Springer, 2004, pp. 210–223.

[135] H. Abasi, N. H. Bshouty, and H. Mazzawi, "On exact learning monotone dnf from membership queries," in *International Conference on Algorithmic Learning Theory*, Springer, 2014, pp. 111–124.

[136] H. Abasi, N. H. Bshouty, and H. Mazzawi, "Non-adaptive learning of a hidden hypergraph," *Theoretical Computer Science*, vol. 716, pp. 15 –27, 2018, Special Issue on ALT 2015.

[137] H. Abasi, "Error-tolerant non-adaptive learning of a hidden hypergraph," in *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[138] M. P. Menden *et al.*, "Community assessment to advance computational prediction of cancer drug combinations in a pharmacogenomic screen," *Nature communications*, vol. 10, no. 1, pp. 1–17, 2019.

[139] Å. Flobak, B. Niederdorfer, V. T. Nakstad, L. Thommesen, G. Klinkenberg, and A. Lægreid, "A high-throughput drug combination screen of targeted small molecule inhibitors in cancer cell lines," *Scientific data*, vol. 6, no. 1, pp. 1–10, 2019.

[140] S. S. Skiena, *The algorithm design manual: Text*. Springer Science & Business Media, 1998, vol. 1.

[141] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, “Coalition structure generation with worst case guarantees,” *Artificial Intelligence*, vol. 111, no. 1-2, pp. 209–238, 1999.

[142] T. Sandholm, “Algorithm for optimal winner determination in combinatorial auctions,” *Artificial intelligence*, vol. 135, no. 1-2, pp. 1–54, 2002.

[143] N. Alon, R. Beigel, S. Kasif, S. Rudich, and B. Sudakov, “Learning a hidden matching,” *SIAM Journal on Computing*, vol. 33, no. 2, pp. 487–501, 2004.

[144] R. Beigel, N. Alon, S. Kasif, M. S. Apaydin, and L. Fortnow, “An optimal procedure for gap closing in whole genome shotgun sequencing,” in *Proceedings of the fifth annual international conference on Computational biology*, 2001, pp. 22–30.

[145] H. Gao, F. K. Hwang, M. T. Thai, W. Wu, and T. Znati, “Construction of d (h)-disjunct matrix for group testing in hypergraphs,” *Journal of Combinatorial Optimization*, vol. 12, no. 3, pp. 297–301, 2006.

[146] F. K.-m. Hwang and D.-z. Du, *Pooling designs and nonadaptive group testing: important tools for DNA sequencing*. World Scientific, 2006, vol. 18.

[147] D. Angluin, “Queries and concept learning,” *Machine learning*, vol. 2, no. 4, pp. 319–342, 1988.

[148] N. H. Bshouty, “Exact learning from an honest teacher that answers membership queries,” *Theoretical Computer Science*, vol. 733, pp. 4–43, 2018.

[149] J. Chodoriwsky and L. Moura, “An adaptive algorithm for group testing for complexes,” *Theoretical Computer Science*, vol. 592, pp. 1 –8, 2015.

[150] A. G. D’yachkov, I. V. Vorobyev, N. A. Polyanskii, and V. Y. Shchukin, “On multistage learning a hidden hypergraph,” in *2016 IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 1178–1182.

[151] F. Y. Chin, H. C. Leung, and S.-M. Yiu, “Non-adaptive complex group testing with multiple positive sets,” *Theoretical Computer Science*, vol. 505, pp. 11–18, 2013.

[152] H.-B. Chen, H.-L. Fu, and F. K. Hwang, “An upper bound of the number of tests in pooling designs for the error-tolerant complex model,” *Optimization Letters*, vol. 2, no. 3, pp. 425–431, 2008.

[153] A. Andoni, Z. Song, C. Stein, Z. Wang, and P. Zhong, “Parallel graph connectivity in log diameter rounds,” in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, 2018, pp. 674–685.

[154] A. Czumaj, J. Lacki, A. Madry, S. Mitrovic, K. Onak, and P. Sankowski, "Round compression for parallel matching algorithms," *SIAM Journal on Computing*, no. 0, STOC18–1, 2019.

[155] R. Vemuganti, "Applications of set covering, set packing and set partitioning models: A survey," in *Handbook of combinatorial optimization*, Springer, 1998, pp. 573–746.

[156] A. Bernstein and C. Stein, "Faster fully dynamic matchings with small approximation ratios," in *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2016, pp. 692–711.

[157] M. T. Goodrich, N. Sitchinava, and Q. Zhang, "Sorting, searching, and simulation in the mapreduce framework," in *International Symposium on Algorithms and Computation*, Springer, 2011, pp. 374–383.

[158] S. Abbar, S. Amer-Yahia, P. Indyk, S. Mahabadi, and K. R. Varadarajan, "Diverse near neighbor problem," in *Proceedings of the twenty-ninth annual symposium on Computational geometry*, ACM, 2013, pp. 207–214.

[159] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause, "Distributed submodular maximization: Identifying representative elements in massive data," in *Advances in Neural Information Processing Systems*, 2013, pp. 2049–2057.

[160] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause, "Streaming submodular maximization: Massive data summarization on the fly," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2014, pp. 671–680.

[161] M.-F. F. Balcan, S. Ehrlich, and Y. Liang, "Distributed k-means and k-median clustering on general topologies," in *Advances in Neural Information Processing Systems*, 2013, pp. 1995–2003.

[162] M Bateni, A. Bhashkara, S. Lattanzi, and V. Mirrokni, "Mapping core-sets for balanced clustering," *Advances in Neural Information Processing Systems*, vol. 26, pp. 5–8, 2014.

[163] P. Indyk, S. Mahabadi, M. Mahdian, and V. S. Mirrokni, "Composable core-sets for diversity and coverage maximization," in *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM, 2014, pp. 100–108.

[164] A. Bhaskara, A. Rostamizadeh, J. Altschuler, M. Zadimoghaddam, T. Fu, and V. Mirrokni, "Greedy column subset selection: New bounds and distributed algorithms," 2016.

[165] C. H. Papadimitriou, *Computational complexity*. John Wiley and Sons Ltd., 2003.

[166] P. Berman, “A d/2 approximation for maximum weight independent set in d-claw free graphs,” in *Scandinavian Workshop on Algorithm Theory*, Springer, 2000, pp. 214–219.

[167] P. Berman and P. Krysta, “Optimizing misdirection,” in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2003, pp. 192–201.

[168] B. Chandra and M. M. Halldórsson, “Greedy local improvement and weighted set packing approximation,” *Journal of Algorithms*, vol. 39, no. 2, pp. 223–240, 2001.

[169] M. M. Halldórsson, “Approximating discrete collections via local improvements.,” in *SODA*, vol. 95, 1995, pp. 160–169.

[170] C. A. J. Hurkens and A. Schrijver, “On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems,” *SIAM Journal on Discrete Mathematics*, vol. 2, no. 1, pp. 68–72, 1989.

[171] M. Sviridenko and J. Ward, “Large neighborhood local search for the maximum set packing problem,” in *International Colloquium on Automata, Languages, and Programming*, Springer, 2013, pp. 792–803.

[172] M. Cygan, “Improved approximation for 3-dimensional matching via bounded pathwidth local search,” in *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, IEEE, 2013, pp. 509–518.

[173] M. Furer and H. Yu, “Approximate the k-set packing problem by local improvements,” *arXiv preprint arXiv:1307.2262*, 2013.

[174] M. Ghaffari and J. Uitto, “Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation,” in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2019, pp. 1636–1653.

[175] S. Behnezhad, M. Hajiaghayi, and D. G. Harris, “Exponentially faster massively parallel maximal matching,” in *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, 2019, pp. 1637–1649.

[176] M. Fischer, M. Ghaffari, and F. Kuhn, “Deterministic distributed edge-coloring via hypergraph maximal matching,” in *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, IEEE, 2017, pp. 180–191.

[177] D. G. Harris, “Distributed approximation algorithms for maximum matching in graphs and hypergraphs,” *arXiv preprint arXiv:1807.07645*, 2018.

[178] M. M. Halldórsson, “Approximations of weighted independent set and hereditary subset problems,” in *Graph Algorithms And Applications 2*, World Scientific, 2004, pp. 3–18.

[179] P. Berman and M. Fürer, "Approximating maximum independent set in bounded degree graphs.," in *Soda*, vol. 94, 1994, pp. 365–371.

[180] M. M. Halldórsson and J. Radhakrishnan, "Greed is good: Approximating independent sets in sparse and bounded-degree graphs," *Algorithmica*, vol. 18, no. 1, pp. 145–163, 1997.

[181] L. Trevisan, "Non-approximability results for optimization problems on bounded degree instances," in *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, 2001, pp. 453–461.

[182] Y. H. Chan and L. C. Lau, "On linear and semidefinite programming relaxations for hypergraph matching," *Mathematical programming*, vol. 135, no. 1-2, pp. 123–148, 2012.

[183] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[184] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015.

[185] J. Kunegis, "Konect: The koblenz network collection," in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 1343–1350.

[186] A. Frieze and S. Janson, "Perfect matchings in random s-uniform hypergraphs," *Random Structures & Algorithms*, vol. 7, no. 1, pp. 41–57, 1995.

[187] J. H. Kim, "Perfect matchings in random uniform hypergraphs," *Random Structures & Algorithms*, vol. 23, no. 2, pp. 111–132, 2003.

[188] F. Dufossé, K. Kaya, I. Panagiotas, and B. Uçar, "Effective heuristics for matchings in hypergraphs," in *International Symposium on Experimental Algorithms*, Springer, 2019, pp. 248–264.

[189] J. Shun, "Practical parallel hypergraph algorithms," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2020, pp. 232–249.

[190] P. Berman, B. DasGupta, and E. Sontag, "Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks," *Discrete Applied Mathematics*, vol. 155, no. 6-7, pp. 733–749, 2007.

[191] V. V. Vazirani, *Approximation algorithms*. Springer Science & Business Media, 2013.

Appendix A: Two Stage Robust Matching

A.1 NP-Hardness proofs for TSRMB

We start by presenting the 3-Dimensional Matching and Set Cover problems, that we use in our reductions to show Theorem 2.3.1 and Theorem 2.3.2. Both problems are known to be strongly NP-hard [128, 129].

3-Dimensional Matching (3-DM): Given three sets U, V, and W of equal cardinality n, and a subset T of $U \times V \times W$, is there a subset M of T with $|M| = n$ such that whenever (u, v, w) and (u', v', w') are distinct triples in M, $u \neq u'$, $v \neq v'$, and $w \neq w'$?

Set Cover Problem: Given a set of elements $\mathcal{U} = \{1, 2, ..., n\}$ (called the universe), a collection $S_1, \ldots, S_m$ of m sets whose union equals the universe and an integer p.
Question: Is there a set $C \subset \{1, \ldots, m\}$ such that $|C| \leq p$ and $\bigcup_{i \in C} S_i = \mathcal{U}$?

Theorem 2.3.2. *In the implicit model of uncertainty, even when $k = 1$, there is no $(2 - \epsilon)$-approximation algorithm for TSRMB for any fixed $\epsilon > 0$, unless $P = NP$.*

Proof of Theorem 2.3.2. We prove the theorem for $k = 1$. We start from an instance of the Set Cover problem and construct an instance of the TSRMB problem. Consider an instance of the decision problem of set cover. We can use it to construct the following TSRMB instance:

- Create m drivers $D = \{1, \ldots, m\}$. For each $j \in \{1, \ldots, m\}$, driver j corresponds to set S_j.
- Create $m - p$ first stage riders, $R_1 = \{1, \ldots, m - p\}$.
- Create n second stage riders, $R_2 = \{1, \ldots, n\}$.

- Set $\mathcal{S} = \{\{1\}, \ldots, \{n\}\}$. Every scenario is of size 1.

As for the distances between riders and drivers, we define them as follows:

- For $(i, j) \in R_1 \times D$, $d(i, j) = 1$.
- For $(i, j) \in R_2 \times D$, $d(i, j) = \begin{cases} 1 & \text{if } i \in S_j \\ 3 & \text{otherwise.} \end{cases}$
- For $i, i' \in R_1 \cup R_2$, $d(i, i') = \min_{j \in D} d(i, j) + d(i', j)$.
- For $j, j' \in D$, $d(j, j') = \min_{i \in R_1 \cup R_2} d(i, j) + d(i, j')$.

This choice of distances induces a metric graph. Moreover, every feasible solution to this TSRMB instance has a first stage cost of exactly 1. We show that a set cover of size $\leq p$ exists if and only if there is a TSRMB solution with total cost equal to 2. Suppose without loss of generality that $S_1, \ldots, S_p$ is a set cover. Then by using the drivers $\{1, \ldots, p\}$ in the second stage, we ensure that every scenario is matched with a cost of 1. This implies the existence of a solution with total cost equal to 2. Now suppose there is a solution to the TSRMB problem with cost equal to 2. Let D_2 be the set of second stage drivers of this solution, then we have $|D_2| = p$. We claim that the sets corresponding to drivers in D_2 form a set cover. In fact, since the total cost of the TSRMB solution is equal to 2, the second stage cost is equal to 1. This means that for every scenario $i \in \{1, \ldots, n\}$, there is a driver $j \in D_2$ within a distance 1 from i. Therefore $i \in S_j$ and $\{S_j : j \in D_2\}$ is a set cover.

Next we show that if $(2 - \epsilon)$-approximation (for some $\epsilon > 0$) to the TSRMB exists then Set Cover is decidable. We know that the TSRMB problem has a solution of cost 2 if and only if there is a set cover of size less than p. Furthermore, if there is no such set cover, the cost of the optimal solution must be 4. Therefore, if the algorithm guarantees a ratio of $(2 - \epsilon)$ and there is a set cover of size less than p, the algorithm delivers a solution with a total cost of 2. If there is no set cover, then clearly the solution produced by the algorithm has a cost of 4. □

Remark A.1.1. *For $k \geq 2$, we can use a generalization of Set Cover to show that the problem is hard for any k. We use a reduction from the Set MultiCover Problem ([190, 191]) defined below.*

Set MultiCover Problem: Given a set of elements $\mathcal{U} = \{1, 2, ..., n\}$ (called the universe) and a collection $S_1, \ldots, S_m$ of m sets whose union equals the universe. A "coverage factor" (positive integer) k and an integer p. Is there a set $C \subset \{1, \ldots, m\}$ such that $|C| \leq p$ and for each element $x \in \mathcal{U}$, $|j \in C \ : \ x \in S_j| \geq k$?

We can create an instance of TSRMB from a Set MultiCover instance similarly to Set Cover with the exception that $\mathcal{S} = \{S \subset R_2 \ s.t. \ |S| = k\}$. The hardness result follows similarly.

A.2 Proofs of Section 2.6.2

Proof of Claim 2.6.4.

1. In the optimal solution of the original problem, R_1 is matched to a subset D_1^* of drivers. The scenario S_1 is matched to a set of drivers D_{S_1} where $D_1^* \cap D_{S_1} = \emptyset$. Let D_o be the set of drivers that are matched to $o_1, \ldots, o_j^*$ in a scenario that contains $o_1, \ldots, o_j^*$. It is clear that $D_1^* \cap D_o = \emptyset$. We claim that $D_o \cap D_{S_1} = \emptyset$. In fact, suppose there is a driver $\rho \in D_o \cap D_{S_1}$. This implies the existence of some o_j with $j \leq j^*$ and some rider $r \in S_1$ such that $d(\rho, o_j) \leq OPT_2$ and $d(\rho, r) \leq OPT_2$. But then $d(\delta, o_j) \leq d(\delta, r) + d(\rho, r) + d(\rho, o_j) \leq 3OPT_2$ which contradicts the fact the o_j is an outlier. Therefore $D_o \cap D_{S_1} = \emptyset$. We show that D_1^* is a feasible first stage solution to the single scenario problem of $S_1 \cup \{o_1, \ldots o_j^*\}$ with a cost less than $OPT_1 + OPT_2$. In fact, D_1^* can be matched to R_1 with a cost less than OPT_1, D_{S_1} to S_1 and D_o to $\{o_1, \ldots, o_j^*\}$ with a cost less than OPT_2. Therefore $\Omega_{j^*} + \Delta_{j^*} \leq OPT_1 + OPT_2$.

2. Recall that $cost_1\big(D_1(j^*), R_1\big) = \Omega_{j^*}$. Consider a scenario S and a rider $r \in S$. Let $\mathcal{B}'$ be the set of the $n - \ell$ closest second stage riders to δ. Let $D_{S_1}(j^*)$ be set of second stage drivers matched to S_1 in the single scenario problem for scenario $S_1 \cup \{o_1, \ldots, o_{j^*}\}$. Let $D_o(j^*)$ be the set of second stage drivers matched to $\{o_1, \ldots, o_{j^*}\}$ in the single scenario problem for scenario $S_1 \cup \{o_1, \ldots, o_{j^*}\}$. Recall that the second stage cost for this single scenario problem is Δ_{j^*}. We distinguish three cases:

(a) If $r \in \mathscr{B}'$, then by Lemma 2.6.2, r is connected to every driver in $D_{S_1}(j^*)$ within a distance less than $\Delta_{j^*} + 4OPT_2$.

(b) If $r \in \{o_{j^*+1}, \ldots, o_\ell\}$, then r is connected to every driver in $D_{S_1}(j^*)$ within a distance less than $3OPT_2 + OPT_2 + \Delta_j^*$.

(c) If $r \in \{o_1, \ldots, o_{j^*}\}$ (i.e., r an outlier), then r can be matched to a different driver in $D_o(j^*)$ within a distance less than OPT_2.

This means that in every case, we can match r to a driver in $D \setminus D_1(j^*)$ with a cost less than $4OPT_2 + \Delta_{j^*}$. This implies that

$$\max_{S \in \mathcal{S}} cost_2\big(D \setminus D_1(j^*), S\big) \leq 4OPT_2 + \Delta_{j^*}$$

and therefore

$$\Omega_{j^*} + \max_{S \in \mathcal{S}} cost_2\big(D \setminus D_1(j^*), S\big) \leq \Omega_{j^*} + \Delta_{j^*} + 4OPT_2 \leq OPT_1 + 5OPT_2.$$

□

Proof of Claim 2.6.5. Let α_j be the second stage cost of $D_1(j)$ on the TSRBM instance with scenarios S_1 and S_2. Formally, $\alpha_j = \max_{S \in \{S_1, S_2\}} cost_2\big(D \setminus D_1(j), S\big)$. Therefore $\beta_j = \Omega_j + \alpha_j$. Let's consider the two sets

$$O_1 = \{r \in \{o_1, \ldots, o_\ell\} \mid d(r, \delta) > 2\alpha_j + OPT_2\}.$$
$$O_2 = \{o_1, \ldots, o_\ell\} \setminus O_1.$$

Consider $D_1(j)$ as a first stage decision to TSRMB with scenarios S_1 and S_2. Let $\tilde{D}_1 \subset D \setminus D_1(j)$ be the set of drivers that are matched to O_1 when the scenario $S_2 = \{o_1, \ldots, o_\ell\}$ is realized. Similarly, let $\tilde{D}_2 \subset D \setminus D_1(j)$ be the drivers matched to scenario S_1. We claim that $\tilde{D}_1 \cap \tilde{D}_2 = \emptyset$. Suppose that there exists some driver $\rho \in \tilde{D}_1 \cap \tilde{D}_2$, this implies the existence of some $o \in O_1$ and $r \in S_1$ such that $d(\rho, o) \leq \alpha_j$ and $d(\rho, r) \leq \alpha_j$. And since $d(r, \delta) \leq OPT_2$ by definition of δ we would have

$$d(o, \delta) \leq d(\rho, o) + d(\rho, r) + d(r, \delta) \leq 2\alpha_j + OPT_2,$$

which contradicts the definition of O_1. Therefore $\tilde{D}_1 \cap \tilde{D}_2 = \emptyset$.

Now consider a scenario $S \in \mathcal{S}$. The riders of $S \cap O_1$ can be matched to $\tilde{D}_1$ with a bottleneck cost less than α_j. Recall that by Lemma 2.6.2, any rider in $R_2 \setminus \{o_1, \ldots, o_\ell\}$ is within a distance less than $4OPT_2$ from any rider in S_1. The riders $r \in S \setminus \{o_1, \ldots, o_\ell\}$ can therefore be matched to any driver $\rho \in \tilde{D}_2$ within a distance less than

$$d(r, \rho) \leq d(r, S_1) + d(S_1, \rho) \leq 4OPT_2 + \alpha_j.$$

As for riders $r \in S \cap O_2$, they can also be matched to any driver ρ of $\tilde{D}_2$ within a distance less than

$$d(r, \rho) \leq d(r, \delta) + d(\delta, S_1) + d(S_1, \rho) \leq 2\alpha_j + OPT_2 + OPT_2 + \alpha_j = 3\alpha_j + 2OPT_2.$$

Therefore we can bound the second stage cost

$$\max_{S \in \mathcal{S}} cost_2\big(D \setminus D_1(j), S\big) \leq \max\{\alpha_j + 4OPT_2, 3\alpha_j + 2OPT_2\}$$

and we get that

$$cost_1\big(D_1(j), R_1\big) + \max_{S \in \mathcal{S}} cost_2\big(D \setminus D_1(j), S\big) \leq \max\{\beta_j + 4OPT_2,\ 3\beta_j + 2OPT_2\}$$

The other inequality $\beta_j \leq cost_1\big(D_1(j), R_1\big) + \max_{S \in \mathcal{S}} cost_2\big(D \setminus D_1(j)\big)$ is trivial. $\square$

A.3 Two-Stage Stochastic Bottleneck Matching Problem (TSSMB)

A.3.1 Problem formulation

In this section, we consider a variant of the TSRMB problem with an expected second stage cost over scenarios instead of a worst-case cost. In particular, we consider a set R_1 of first stage riders which is given first, and must immediately and irrevocably be matched to a subset of drivers D_1 ($D_1 \subset D$). Once R_1 is matched, a scenario $S_i \subset R_2$ is revealed from a list $\mathcal{S} = \{S_1, \ldots, S_q\}$ with probability p_i and need to be matched using the remaining drivers. The expected second stage cost is $\sum_{i=1}^{q} p_i \cdot cost_2(D \setminus D_1, S_i)$. The objective function is given by

$$\min_{D_1 \subset D} \left\{ cost_1(D_1, R_1) + \sum_{i=1}^{q} p_i \cdot cost_2(D \setminus D_1, S_i) \right\},$$

where $cost_1(D_1, R_1)$ and $cost_2(D \setminus D_1, S_i)$ are defined similarly to the TSRMB problem. For brevity of notation, we set $f(D_1) = cost_1(D_1, R_1) + \sum_{i=1}^{q} p_i \cdot cost_2(D \setminus D_1, S_i)$. Given an optimal first-stage solution D_1^*, we denote $OPT_1 = cost_1(D_1^*, R_1)$, $OPT_2 = \sum_{i=1}^{q} p_i \cdot cost_2(D \setminus D_1^*, S_i)$ and $OPT = OPT_1 + OPT_2$.

A.3.2 NP-hardness

Corollary A.3.1. *TSSMB is NP-hard to approximate within a factor better than $\frac{4}{3}$.*

Proof. Similar to the proof of Theorem 2.3.1, with $S_1 = U$, $S_2 = V$, $S_3 = W$, and equal probabilities $p_1 = p_2 = p_3 = \frac{1}{3}$. If there is a valid 3-DM then the total cost is equal to 2, and if there is no 3-DM then the total cost is at least $1 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 3 = \frac{8}{3}$. Therefore any algorithm with an approximation ratio strictly less than $\frac{\frac{8}{3}}{2} = \frac{4}{3}$ implies that 3-Dimensional Matching is decidable. □

A.3.3 No surplus

Consider the case where there is no surplus of drivers, i.e., the total number of drivers is equal to $|R_1|$ plus the size of the maximum scenario. We assume for the the sake of simplicity that all scenarios have the same size. The proof follows as well if the sizes are different. We show in this case that we can have a 3-approximation by considering every scenario independently to get different first stage decisions, and then picking the best first stage decision among them. For every scenario S_i, we solve the following problem

$$\min_{D_1 \subset D} \left\{ cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_i) \right\}$$

When there is no surplus, we know that in the optimal solution, the same set of drivers is matched to every scenario. Therefore if we have a solution for one single scenario, we can use the triangular inequality to bound the cost of this solution for any scenario.

Algorithm 14

```
1: for $i \in \{1, \dots, q\}$ do
2:     $D_i^1 :=$ TSRMB-1-Scenario$(R_1, S_i, D)$.
3: return $D_1 = \arg\min_{D_i^1} f(D_i^1)$
```

Theorem A.3.1. *Algorithm 14 yields a solution with total cost less than $OPT_1 + 3OPT_2$ for TSSMB with no surplus.*

Proof. Let OPT_1 and OPT_2 be the first and second stage cost of the optimal solution, and $b_1, \dots, b_q$ be the bottleneck edge weights in the optimal second stage matchings for $S_1, \dots, S_q$. Therefore $OPT_2 = \sum_{i=1}^{q} p_i \cdot b_i$. We claim that $\min_i f(D_i^1) \leq OPT_1 + 3OPT_2$. For every i, let α_i and β_i be the first and second stage cost of D_i^1 when we consider only scenario S_i, that is

$$cost_1(D_i^1, R_1) + cost_2(D \setminus D_i^1, S_i) = \alpha_i + \beta_i.$$

It is clear that $\alpha_i + \beta_i \leq OPT_1 + b_i$. Furthermore, when a scenario S_j $(j \neq i)$ is realized, we can bound the cost of matching S_j to $D \setminus D_i^1$ by using the triangular inequality,

$$cost_2(D \setminus D_i^1, S_j) \leq \beta_i + b_i + b_j.$$

Therefore we get that,

$$\begin{aligned} f(D_i^1) &\le \alpha_i + p_i \cdot \beta_i + \sum_{j\neq i} p_j(\beta_i + b_j + b_i) \\ &= \alpha_i + \beta_i + \sum_{j\neq i} p_j(b_j + b_i) \\ &\le OPT_1 + p_i \cdot b_i + \sum_{j\neq i} p_j(b_j + 2b_i) \\ &\le OPT_1 + OPT_2 + 2(1-p_i)b_i. \end{aligned}$$

Next we show that $\min_{1\le i\le q}(1-p_i)\cdot b_i \le OPT_2$. Suppose, in the contrary, that for all $i \in \{1,\ldots,q\}$, we have $(1-p_i)\cdot b_i > OPT_2$, then $p_i \cdot b_i > \frac{p_i}{1-p_i}OPT_2$ and by summing we get that

$$OPT_2 > OPT_2 \sum_{i=1}^{q} \frac{p_i}{1-p_i}.$$

We can assume without loss of generality that $OPT_2 > 0$ and therefore we get that

$$1 > \sum_{i=1}^{q} \frac{p_i}{1-p_i} = \sum_{i=1}^{q} \frac{p_i+1-1}{1-p_i} = \sum_{i=1}^{q} \frac{1}{1-p_i} - q,$$

which implies that

$$q \ge \sum_{i=1}^{q} \frac{1}{1-p_i} \tag{A.1}$$

By the Cauchy-Schwarz inequality, we know that

$$(q-1)\cdot \sum_{i=1}^{q} \frac{1}{1-p_i} = \sum_{i=1}^{q}(1-p_i)\cdot \sum_{i=1}^{q} \frac{1}{1-p_i} \ge q^2.$$

Therefore,

$$\sum_{i=1}^{q} \frac{1}{1-p_i} \ge \frac{q^2}{q-1} > q,$$

which contradicts (A.1). Hence $\min_{1\le i\le q}(1-p_i)\cdot b_i \le OPT_2$ and $f(D_1) = \min_i f(D_i^1) \le OPT_1 + 3OPT_2$. □

A.4 Two-Stage Robust Matching Problem (TSRM)

In this section, we consider a variant of the TSRMB problem with the total weight instead of the bottleneck as a second stage cost. Similarly, we consider a set R_1 of first stage riders which is given first, and must immediately and irrevocably be matched to a subset of drivers D_1 ($D_1 \subset D$). Once R_1 is matched, a scenario $S_i \subset R_2$ is revealed from a list $\mathcal{S} = \{S_1, \dots, S_q\}$. The first stage cost, $cost_1(D_1, R_1)$, is the total weight of the matching between D_1 and R_1 the second stage cost $cost_2(D \setminus D_1, S)$ is the cost of the minimum weight matching between the scenario S and the available drivers ($D \setminus D_1$). Formally, let M_1 be the minimum weight perfect matching between R_1 and D_1, and given a scenario S, let M_2^S be the minimum weight perfect matching between the scenario S and the available drivers $D \setminus D_1$, then the cost functions for the TSRM are:

$$cost_1(D_1, R_1) = \sum_{(i,j)\in M_1} d(i,j), \quad \text{and} \quad cost_2(D \setminus D_1, S) = \sum_{(i,j)\in M_2^S} d(i,j).$$

Given an optimal first-stage solution D_1^*, we denote $OPT_1 = cost_1(D_1^*, R_1)$, $OPT_2 = \max\{cost_2(D\setminus D_1^*, S) \mid S \in \mathcal{S}\}$ and $OPT = OPT_1 + OPT_2$. In this variant we consider an explicit model of scenarios and optimize over the worst case scenario. We show that the problem is NP-hard even with two scenarios. We restate a result from the literature that gives a 3-approximation using a greedy approach. We further improve over this approximation in the specific case of no surplus. We assume for the sake of simplicity that all the scenarios have the same size k.

Theorem A.4.1. *TSRM is NP-hard even when the number of scenarios is equal to 2.*

Proof. We construct (in polynomial time) a reduction from the 2-partition problem. Let $\big(I, (s_i)_{i\in I}\big)$ be an instance of the 2-partition problem.

The 2-partition problem:

Instance: Finite set I and number $s_i \in \mathbb{Z}^+$ for $i \in I$.

Question: Is there a subset $I' \subset I$ such that $\sum_{i\in I'} s_i = \sum_{i\in I\setminus I'} s_i$?

It is well known that the 2-partition problem is weakly NP-hard even when $|I'| = |I|/2$.

Without loss of generality, suppose that $I = \{1, \ldots, n\}$ for some integer n. We construct the following instance of TSRM with two scenarios. Let $R_1 = \{r_1, \ldots, r_n\}$, $S_1 = \{r_{n+1}, \ldots, r_{2n}\}$ and $S_2 = \{r_{2n+1}, \ldots, r_{3n}\}$. Note that every scenario is of size n. Let $D = \{\delta_1, \ldots, \delta_{2n}\}$. Let P be a sufficiently big constant such that $P \geq \sum_{i \in I} s_i$. We define the distances between drivers and riders as follows:

- For $j \in \{1, \ldots, n\}$: $d(\delta_j, r_j) = P$, $d(\delta_{n+j}, r_j) = P$ and and $d(\delta_i, r_j) = \infty$ otherwise.
- For $j \in \{n+1, \ldots, 2n\}$, $d(\delta_{j-n}, r_j) = P$, $d(\delta_j, r_j) = s_{j-n}$ and $d(\delta_i, r_j) = \infty$ otherwise.
- For $j \in \{2n+1, \ldots, 3n\}$, $d(\delta_{j-2n}, r_j) = s_{j-2n}$, $d(\delta_{j-n}, r_j) = P$ and $d(\delta_i, r_j) = \infty$ otherwise.

This choice of distances induces a metric bipartite graph on R_1 and $S_1 \cup S_2$. A feasible solution to this TSRM instance with bounded cost has two possibilities to match rider $r_j \in R_1$ ($j \leq n$): either to driver δ_j or driver δ_{n+j}. Consider a feasible bounded cost first stage solution D_1, and let I' be the set of indices $j \leq n$ such that the first stage rider r_j is matched to driver δ_j in the first stage. Then $I \setminus I'$ is the set of elements $j \leq n$ such that the first stage rider r_j is matched to driver δ_{j+n}. In both cases, the cost of matching $r_j \in R_1$ is equal to P. When the scenario S_1 is realized, the driver $r_{n+j} \in S_1$ ($j \leq n$) needs to be matched to δ_{j+n} if $j \in I'$, with a cost P and to δ_j if $j \in I \setminus I'$, with a cost s_j. Similarly, when the scenario S_2 is realized, the driver $r_{2n+j} \in S_2$ ($j \leq n$) needs to be matched to δ_{j+n} if $j \in I'$, with a cost s_j and to δ_j if $j \in I \setminus I'$, with a cost P. The first and second stage costs are therefore:

$$cost_1(D_1, R_1) = P|I|,$$

$$cost_2(D \setminus D_1, S_1) = P|I'| + \sum_{j \in I \setminus I'} s_j,$$

$$cost_2(D \setminus D_1, S_2) = P|I \setminus I'| + \sum_{j \in I'} s_j.$$

We claim that there exists a 2-partition I' such that $|I'| = |I \setminus I'|$ if and only if there is a solution with total cost equal to $\frac{1}{2}\big(3P|I| + \sum\limits_{j\in I} s_j\big)$.

Suppose there exist a 2-partition I' with $|I'| = |I \setminus I'|$. This implies that

$$\sum_{j\in I'} s_j + P|I \setminus I'| = \sum_{j\in I\setminus I'} s_j + P|I'| = \frac{1}{2}\big(P|I| + \sum_{j\in I} s_j\big) \tag{A.2}$$

Let D_1 be the first stage decision that for every $j \leq n$, matches r_j to δ_j if $j \in I'$, and r_j to δ_{n+j} otherwise. The costs of this first stage decision on scenarios S_1 and S_2 are:

$$cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_1) = P|I| + P|I'| + \sum_{j\in I\setminus I'} s_j = \frac{1}{2}\big(3P|I| + \sum_{j\in I} s_j\big),$$
$$cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_2) = P|I| + P|I \setminus I'| + \sum_{j\in I'} s_j = \frac{1}{2}\big(3P|I| + \sum_{j\in I} s_j\big).$$

Therefore the total cost of D_1 is equal to

$$cost_1(D_1, R_1) + \max_{S\in\{S_1,S_2\}} cost_2(D \setminus D_1, S) = \frac{1}{2}\big(3P|I| + \sum_{j\in I} s_j\big).$$

Suppose now that there is a first stage decision D_1 with bounded total cost equal to $\frac{1}{2}\big(3P|I| + \sum\limits_{j\in I} s_j\big)$. Let I' be the set of indices $j \leq n$ such that, in the first stage matching of D_1, r_j is matched to driver δ_j for $j \leq n$. We know that

$$cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_1) = P|I| + P|I'| + \sum_{j\in I\setminus I'} s_j \leq \frac{1}{2}\big(3P|I| + \sum_{j\in I} s_j\big)$$
$$cost_1(D_1, R_1) + cost_2(D \setminus D_1, S_2) = P|I| + P|I \setminus I'| + \sum_{j\in I'} s_j \leq \frac{1}{2}\big(3P|I| + \sum_{j\in I} s_j\big)$$

This implies the following inequalities

$$P|I'| + \sum_{j \in I \setminus I'} s_j \leq \frac{1}{2}\big(P|I| + \sum_{j \in I} s_j\big) \tag{A.3}$$

$$P|I \setminus I'| + \sum_{j \in I'} s_j \leq \frac{1}{2}\big(P|I| + \sum_{j \in I} s_j\big) \tag{A.4}$$

The only way (A.3) and (A.4) can hold is if we have

$$P|I'| + \sum_{j \in I \setminus I'} s_j = P|I \setminus I'| + \sum_{j \in I'} s_j = \frac{1}{2}\big(P|I| + \sum_{j \in I} s_j\big) \tag{A.5}$$

Now suppose that $|I \setminus I'| > |I'|$, since we can make P as big as needed, then equation (A.5) cannot hold. Therefore $|I \setminus I'| \leq |I'|$. Similarly, we get that $|I'| \leq |I \setminus I|$, Therefore, $|I'| = |I \setminus I'|$ and equation (A.5) becomes

$$\sum_{j \in I \setminus I'} s_j = \sum_{j \in I'} s_j. \tag{A.6}$$

This shows that I' is a 2-Partition with $|I'| = |I \setminus I'|$. □

Lemma A.4.1. *The greedy algorithm that minimizes only the first stage cost yields a solution with total cost less than $3OPT_1 + OPT_2$ to the TSRM.*

Proof. Special case of Theorem 2.4 in [82]. □

Lemma A.4.2. *If the surplus $\ell = |D| - |R_1| - k$ is equal to zero, Algorithm 15 yields a solution with a total cost less than $OPT_1 + 5OPT_2$ to the TSRM.*

Algorithm 15

```
1: Pick a scenario $S \in \mathcal{S}$.
2: Compute a minimum weight perfect matching between $S$ and a subset $D_2$ of drivers.
3: Compute a minimum weight perfect matching between $D \setminus D_2$ and $R_1$. Let $D_1$ be the drivers
   used in this matching.
4: return $D_1$.
```

Proof. Consider Algorithm 15. In the remaining of the proof, we will refer to the total cost of the solution given by Algorithm 15 as ALG, and to its first (resp. second) stage cost as ALG_1 (resp. ALG_2). The proof of the lemma follows immediately by combining the following two claims.

Claim A.4.1. $ALG_2 \leq 3OPT_2$.

Proof. We use the notation $M(A, B)$ to refer to the total weight of the minimum weight perfect matching between a set of drivers A and a set of riders B. If the scenario S that was picked by the algorithm is realized, then in this case we know that its second stage cost is less than OPT_2. Now, suppose a different scenario $S' \neq S$ is realized. Let D_2^* be the set of k drivers that the optimal solution saves for the second stage. We use the triangular inequality to establish that :

$$M(D_2, S') \leq M(D_2, S) + M(D_2^*, S) + M(D_2^*, S') \tag{A.7}$$

Let's bound the right hand side terms of equation (A.7). $M(D_2, S) \leq OPT_2$ because by definition, the matching between D_2 and S is the best possible between S and any subset of drivers. Now since $|D| = |R_1| + k$, this means that the optimal solution saves exactly k drivers to be matched with any scenario realization. This implies that $M(D_2^*, S) \leq OPT_2$ and $M(D_2^*, S') \leq OPT_2$. The claim follows immediately. □

Claim A.4.2. $ALG_1 \leq OPT_1 + 2OPT_2$.

Proof of claim. We construct a matching that between $D \setminus D_2$ and R_1 with a total weight less than $OPT_1 + 2OPT_2$. Let $r_1 \in R_1$, and $\delta_1(r_1)$ be the driver matched to r_1 in the optimal solution. If $\delta_1(r_1) \notin D_2$, then just match $\delta_1(r_1)$ with r_1. Therefore we can assume without loss of generality that all the drivers $\delta_1(r_1)$ are used in D_2. This means that eactly $|R_1|$ drivers of $D \setminus D_2$ are used in second stage of the optimal solution. We can match R_1 with $D \setminus D_2$ and bound the cost of this matching as follows:

$$M(D \setminus D_2, R_1) \leq M(D \setminus D_2, S) + M(D_2, S) + M(D_2, R_1) \tag{A.8}$$

$M(D \setminus D_2, S) \leq OPT_2$ because exactly $|R_1|$ drivers from $D \setminus D_2$ are used in the second stage of the optimal solution and $|R_1| = |D \setminus D_2|$. $M(D_2, S) \leq OPT_2$ by definition of D_2. Finally, $M(D_2, R_1) \leq OPT_1$ because D_2 includes all the drivers that were used in the first stage of the optimal matching. Therefore, we get

$$ALG_1 = M(D \setminus D_2, R_1) \leq OPT_1 + 2OPT_2.$$

□

□

Theorem A.4.2. *If the surplus $\ell = |D| - |R_1| - k$ is equal to zero, there exists a polynomial time algorithm with a $\frac{7}{3}$-approximation to the TSRM problem.*

Proof. We show the theorem by balancing between the results of Lemma A.4.1 and Lemma A.4.2. Let $Greedy$ denote the total cost of the greedy algorithm. From Lemma A.4.1 and Lemma A.4.2, we have that $Greedy \leq 3OPT_1 + OPT_2$ and $ALG \leq OPT_1 + 5OPT_2$. By taking the minimum of the two algorithms we get:

$$\begin{aligned}\min\{Greedy,\ ALG\} &= \min\{(3OPT_1 + OPT_2,\ OPT_1 + 5OPT_2\} \\ &= OPT \cdot \min\{\tfrac{3OPT_1+OPT_2}{OPT},\ \tfrac{OPT_1+5OPT_2}{OPT}\}\end{aligned}$$

Therefore, and since $OPT = OPT_1 + OPT_2$, we get that:

$$\min\{Greedy,\ ALG\} \leq OPT \cdot \max_{x\in[0,1]} \min\left\{3x + (1-x),\ x + 5(1-x)\right\} \leq \frac{7}{3} \cdot OPT.$$

□

Appendix B: Proofs from Chapter 3

B.1 Missing analysis for learning hypermatchings

B.1.1 Missing analysis for algorithm for learning hypermatchings (Section 3.4)

Proof of Claim 3.4.1. Consider the function $f(x) = \frac{1}{1-x} - e^x$ for $x < 1$. The derivative of f is

$$f'(x) = \frac{1}{(1-x)^2} - e^x = \frac{1}{1-x} \cdot \left(\frac{1}{1-x} - (1-x)e^x \right).$$

Because $1 - x \leq e^{-x}$, we have $(1-x)e^x \leq 1$. Therefore,

$$f'(x) \geq \frac{1}{1-x} \cdot \left(\frac{1}{1-x} - 1 \right) \geq 0,$$

where the last inequality follows from the assumption that $0 \leq x < 1$. □

B.1.2 Missing analysis for hardness of learning hypermatchings (Section 3.5)

Proof of Claim 3.6.2. We have that

$$\begin{aligned} \binom{n}{k} &= \frac{n(n-1)\ldots(n-(k-1)}{k!} \\ &= \frac{n^k}{k!}(1-\frac{1}{n})\ldots(1-\frac{k-1}{n}) \end{aligned}$$

The upper bound $\binom{n}{k} \leq \frac{n^k}{k!}$ follows immediately. To show the lower bound, we observe that

$$\begin{aligned}\binom{n}{k} &= \frac{n^k}{k!}(1-\frac{1}{n})\dots(1-\frac{k-1}{n}) \\ &\geq \frac{n^k}{k!}(1-\frac{k-1}{n})^{k-1}\end{aligned}$$

Now consider the function $f_n(x,y) = (1-\frac{x}{n})^y$ for $y \geq 1$ and $x \in [1,n]$. For $x \in [1,n]$ and fixed $y \geq 1$, the function f_n decreases as x increases. Similarly, for x fixed, f_n decreases as y increases. Therefore

$$(1-\frac{k-1}{n})^{k-1} \geq (1-\frac{1}{\sqrt{n}})^{\sqrt{n}}.$$

Furthermore, we know that for $x \geq 2$, $(1-1/x)^x \geq 1/4$. Therefore

$$\begin{aligned}\binom{n}{k} &\geq \frac{n^k}{k!}(1-\frac{k-1}{n})^{k-1} \\ &\geq \frac{n^k}{k!}(1-\frac{1}{\sqrt{n}})^{\sqrt{n}} \\ &\geq \frac{n^k}{4(k!)}.\end{aligned}$$

□

Proof of Lemma 3.6.2. Since the partition i is fixed, we drop indexing by i and use e_j to denote e^i_j, k to denote k_i, and d to denote d_i. We also let $n = |P_i|$ and assume without loss of generality that $S \subseteq |P_i|$.

The intuition is that, since the edges e_j must be disjoint but the edges e'_j do not necessarily have to, then the edges e_j will cover a "bigger fraction" of S than e'_j. We prove this by induction on the number of edges. When considering only one edge e_1, we know that

$$\mathbb{P}(e_1 \not\subseteq S) = \mathbb{P}(e_j \not\subseteq S) = \mathbb{P}(e'_1 \not\subseteq S).$$

Now assume that the result is true for j edges, i.e. $\mathbb{P}(e_1 \not\subseteq S, \dots, e_j \not\subseteq S) \leq \mathbb{P}(e'_1 \not\subseteq S)^j$. We want to show that it holds for $j + 1$ edges, i.e.,

$$\mathbb{P}(e_1 \nsubseteq S, \ldots, e_j \nsubseteq S, e_{j+1} \nsubseteq S) \leq \mathbb{P}(e'_1 \nsubseteq S)^{j+1}.$$

By the inductive hypothesis, it is sufficient to show that

$$P(e_{j+1} \nsubseteq S | \, \forall \, l \leq j, \; e_l \nsubseteq S) \leq \mathbb{P}(e_{j+1} \nsubseteq S)$$

This is equivalent to showing that $P(e_{j+1} \subseteq S| \, e_1 \nsubseteq S, \ldots, e_j \nsubseteq S) \geq \mathbb{P}(e_{j+1} \subseteq S)$. Furthermore, we have

$$\begin{aligned}\mathbb{P}(e_{j+1} \subseteq S) = \mathbb{P}(e_{j+1} \subseteq S| \, \forall \, l \leq j, \; e_l \nsubseteq S)\mathbb{P}(\forall \, l \leq j, \; e_l \nsubseteq S) \\ + \mathbb{P}(e_{j+1} \subseteq S| \, \exists \, l \leq j, \; e_l \subseteq S)\mathbb{P}(\exists \, l \leq j, \; e_l \subseteq S)\end{aligned}$$

Therefore, $P(e_{j+1} \nsubseteq S| \, e_1 \nsubseteq S, \ldots, e_j \nsubseteq S) \leq \mathbb{P}(e_{j+1} \nsubseteq S)$ becomes equivalent to

$$P(e_{j+1} \subseteq S| \, \forall \, l \leq j, \; e_l \nsubseteq S)\mathbb{P}(\exists \, l \leq j, \; e_l \subseteq S) \geq \mathbb{P}(e_{j+1} \subseteq S| \, \exists \, l \leq j, \; e_l \subseteq S)\mathbb{P}(\exists \, l \leq j, \; e_l \subseteq S) \tag{B.1}$$

Since, $\mathbb{P}(\exists \, l \leq j, \; e_l \subseteq S) > 0$, the inequality (B.1) is equivalent to

$$P(e_{j+1} \subseteq S| \, \forall \, l \leq j, \; e_l \nsubseteq S) \geq \mathbb{P}(e_{j+1} \subseteq S| \, \exists \, l \leq j, \; e_l \subseteq S) \tag{B.2}$$

To prove equation (B.2), we observe that

$$\mathbb{P}(e_{j+1} \subseteq S) = pP(e_{j+1} \subseteq S| \, \forall \, l \leq j, \; e_l \nsubseteq S) + (1 - p)\mathbb{P}(e_{j+1} \subseteq S| \, \exists \, l \leq j, \; e_l \subseteq S),$$

where $p = \mathbb{P}(\forall \, l \leq j, \; e_l \nsubseteq S)$. Therefore, if we show that $\mathbb{P}(e_{j+1} \subseteq S) \geq \mathbb{P}(e_{j+1} \subseteq S| \, \exists \, l \leq j, \; e_l \subseteq S)$, then we must have $P(e_{j+1} \subseteq S| \, \forall \, l \leq j, \; e_l \nsubseteq S) \geq \mathbb{P}(e_{j+1} \subseteq S) \geq \mathbb{P}(e_{j+1} \subseteq S| \, \exists \, l \leq$

$j,\ e_l \subseteq S)$. To see that $\mathbb{P}(e_{j+1} \subseteq S) \geq \mathbb{P}(e_{j+1} \subseteq S |\ \exists\, l \leq j,\ e_l \subseteq S)$, we first observe that

$$\begin{aligned}\mathbb{P}(e_{j+1} \subseteq S |\ \exists\, l \leq j,\ e_l \subseteq S) &= \mathbb{P}(e_{j+1} \subseteq S |\ e_1 \subseteq S)\\ &= \frac{\binom{|S|-d}{d}}{\binom{n-d}{d}},\end{aligned}$$

while

$$\mathbb{P}(e_{j+1} \subseteq S) = \frac{\binom{|S|}{d}}{\binom{n}{d}}.$$

Therefore,

$$\begin{aligned}\frac{\mathbb{P}(e_{j+1} \subseteq S)}{\mathbb{P}(e_{j+1} \subseteq S |\ e_1 \subseteq S)} &= \frac{\binom{|S|}{d}}{\binom{n}{d}} \frac{\binom{n-d}{d}}{\binom{|S|-d}{d}}\\ &= \frac{(|S|)!}{(|S|-d)!} \frac{(|S|-2d)!}{(|S|-d)!} \frac{(n-d)!}{n!} \frac{(n-d)!}{(n-2d)!}\\ &= \frac{\frac{|S| \quad \ldots(|S|-d+1)}{(|S|-d)\ldots|S|-2d+1)}}{\frac{n \quad \ldots(n-d+1)}{(n-d)\ldots n-2d+1)}}\end{aligned}$$

It is easy to verify that

$$\frac{|S|-d+i}{|S|-2d+i} \geq \frac{n-d+i}{n-2d+i}$$

for every $i \in \{1, \ldots, d\}$, because $|S| \leq n$. Therefore we get that $\mathbb{P}(e_{j+1} \subseteq S) \geq \mathbb{P}(e_{j+1} \subseteq S |\ e_1 \subseteq S)$, and hence $\mathbb{P}(e_{j+1} \subseteq S) \geq \mathbb{P}(e_{j+1} \subseteq S |\ \exists\, l \leq j,\ e_l \subseteq S)$. This proves (B.2) and concludes the induction proof. $\square$

Appendix C: Distributed Hypergraph Matching

C.1 Omitted proofs

C.1.1 Proof of Corollary 4.6.1

Corollary 4.6.1. *For d-uniform linear hypergraphs, the degree distribution is tighter, and $|d_A(v) - d_B(v)| = O(\log n)\lambda\beta$.*

Proof. For linear hypergraphs, assume that $d_A(v) = k\lambda\beta$ with $k = \Omega(\log n)$ and $d_B(v) = 0$. The difference in the analysis is that, for every edge e belonging to the $k\lambda\beta$ edges that are incident to v in A, we can find a new set of at least $(k - (d+1))\lambda\beta$ edges in $B \setminus A$. In fact, for such an edge e, every one of the $(1-\lambda)\beta$ edges that verify $\sum\limits_{u \neq v} d_B(u) \geq (1-\lambda)\beta$ intersect e in exactly on vertex that is not v. The same goes for the subset of at least $(k-1)\lambda\beta$ that are in $B\setminus$ A, these edges already intersect e, and can at most have one intersection in between them. At most $d(d-1)$ of these edges can be considered simultaneously for different $e_1, \ldots, e_d$ from the $k\lambda\beta$ edges incident to v. Therefore, for every edge e, we can find at least a set of $(k-1)\lambda\beta d(d-1) \geq (k-(d+1))\lambda\beta$ new edges that in $B \setminus A$. This means that at this point we have already covered $k\lambda\beta(k-(d+1))\lambda\beta$ edges in both A and B. One can see that we can continue covering new edges just like in the previous lemma, such that at iteration l, the number of covered edges is at least

$$k(k-(d+1))(k-2(d+1))\ldots(k-l(d+1))(\lambda\beta)^l,$$

for $l \leq \frac{k-1}{d+1}$. It is easy to see that for $l = \frac{k-1}{d+1}$, we will have $k(k-(d+1))(k-2(d+1))\ldots(k - l(d+1))(\lambda\beta)^l > 2n\beta$ if $k = \Omega(\log n)$, which will be a contradiction. □

C.1.2 Proof of Lemma 4.6.2 and Lemma 4.6.3

Lemma 4.6.2. *Let* $\phi(x) = \min\{1, \frac{(d-1)x}{d(\beta-x)}\}$*. If* $a_1, \ldots a_d \geq 0$ *and* $a_1 + \ldots + a_d \geq \beta(1-\lambda)$ *for some* $\lambda \geq 0$*, then* $\sum_{i=1}^{d} \phi(a_i) \geq 1 - 5 \cdot \lambda$*.*

Proof. We will provide a proof for $d = 3$ that is easy to generalize. We first show that if $a+b+c \geq \beta$, then $\phi(a) + \phi(b) + \phi(c) \geq 1$. The claim is true if $\phi(a) \geq 1$ or $\phi(b) \geq 1$ or $\phi(c) \geq 1$. Suppose that $\phi(a) < 1$ and $\phi(b) < 1$ and $\phi(c) < 1$. Then :

$$\phi(a) + \phi(b) + \phi(c) = \frac{d-1}{d}\Big(\frac{a}{\beta-a} + \frac{b}{\beta-b} + \frac{c}{\beta-c}\Big) \geq \frac{d-1}{d}\Big(\frac{a}{b+c} + \frac{b}{a+c} + \frac{c}{a+b}\Big) .$$

By Nesbitt's Inequality we know that

$$\frac{a}{b+c} + \frac{b}{a+c} + \frac{c}{a+b} \geq \frac{d}{d-1} = \frac{3}{2},$$

and therefore $\phi(a) + \phi(b) + \phi(c) \geq 1$.

By the general Nesbitt's Inequality (See appendix C.3), we know that for $d > 3$

$$\sum_{i=1}^{d} \frac{a_i}{\sum_{j \neq i} a_j} \geq \frac{d}{d-1} .$$

So if $\sum_{i=1}^{d} a_i \geq \beta$, then $\sum_{i=1}^{d} \phi(a_i) \geq 1$. Now, let $\phi'(x) = \frac{d}{dx}\phi(x)$. To complete the proof, it is sufficient to show that we always have $\phi'(x) \leq \frac{5}{\beta}$. To prove this inequality, note that if $x \geq \frac{d}{2d-1}\beta$ then $\phi(x) = 1$ and thus $\phi'(x) = 0$. Now, if $x \leq \frac{d}{2d-1}\beta$ then:

$$\phi'(x) = \frac{d-1}{d}\frac{d}{dx}\frac{x}{\beta-x} = \frac{d-1}{d}\frac{\beta}{(\beta-x)^2} ,$$

which is increasing in x and maximized at $x = \frac{d}{2d-1}\beta$, in which case $\phi'(x) = \frac{(2d-1)^2}{d(d-1)}\frac{1}{\beta} \leq \frac{5}{\beta}$. In the end we get:

$$\sum_{i=1}^{d} \phi(a_i) \geq 1 - 5 \cdot \lambda \ .$$

□

Lemma 4.6.3. *Given any $HEDCS(G, \beta, \beta(1-\lambda))$ H, we can find two disjoint sets of vertices X and Y that satisfy the following properties:*

1. *$|X| + |Y| = d \cdot \mu(G)$.*
2. *There is a perfect matching in Y using edges in H.*
3. *Letting $\sigma = \frac{|Y|}{d} + \sum_{x \in X} \phi(d_H(x))$,we have that $\sigma \geq \mu(G)(1-5\lambda)$.*
4. *All edges in H with vertices in X have at least one other vertex in Y, and have vertices only in X and Y.*

Proof. Let M^G be some maximum integral matching in G. Some of the edges in M^G are in H, while others are in $G \setminus H$. Let X_0 contain all vertices incident to edges in $M_G \cap (G \setminus H)$, and let Y_0 contain all vertices incident to edges in $M_G \cap H$. We now show that X_0 and Y_0 satisfy the first three properties of the lemma. Property 1 is satisfied because $X_0 \cup Y_0$ consists of all matched vertices in M_G. Property 2 is satisfied by definition of Y_0. To see that Property 3 is satisfied, remark that the vertices of Y_0 each contribute exactly $\frac{1}{d}$. Now, X_0 consists of $|X_0|/d$ disjoint edge in $G \setminus H$, and by Property P2 of a HEDCS, for each such edge $e : \sum_{x \in e} d_H(x) \geq \beta(1-\lambda)$ and by Lemma 4.6.2, we have $\sum_{x \in e} \phi(d_H(x)) \geq (1-5\lambda)$ and each one of these vertices contributes in average at least $\frac{1-5\lambda}{d}$ to σ, just as desired. Loosely speaking, $\phi(d_H(x))$ will end up corresponding to the profit gained by vertex x in the fractional matching M_f^H.

Consider Y_0 and X_0 from above. These sets might not satisfy the Property 4 (that all edges in H with an endpoint in X have at least one other endpoint in Y). Can we transform these into sets X_1 and Y_1, such that the first three properties still hold and there are no hyperedges with endpoints in X_1 and $V \setminus (X_1 \cup Y_1)$; at this stage, however, there will be possibly edges in H with different endpoints in X_1. To construct X_1,Y_1, we start with $X = X_0$ and $Y = Y_0$, and present a transformation that terminates with $X = X_1$ and $Y = Y_1$. Recall that X_0 has a perfect matching using edges in $G \setminus H$. The set X will maintain this property throughout the transformation, and each vertex $x \in X$ has always a unique *mate* e'. The construction does the following : as long as there exists an edge e in H containing x and only endpoints in X and $V \setminus (X \cup Y)$, let e' be the mate of x, we then remove the endpoints of e' from X and add the endpoints of e to Y. Property 1 is maintained because we have removed d vertices from X and added d to Y. Property 2 is maintained because the vertices we added to Y were connected by an edge in H. Property 3 is maintained because X clearly still has a perfect matching in $G \setminus H$, and for the vertices $\{x_1, x_2, \ldots, x_d\} = e'$, the average contribution is still at least $\frac{1-5\lambda}{d}$, as above. We continue this process while there is an edge with endpoints in X and $V \setminus (X \cup Y)$. The process terminates because each time we are removing d vertices from X and adding d vertices to Y. We end up with two sets X_1 and Y_1 such that the first three properties of the lemma are satisfied and there are no edges with endpoints in X_1 and $V \setminus (X_1 \cup Y_1)$. This means that for any edges in H incident to X, this edge is either incident to Y as well or incident to only points in X.

We now set $X = X_1$ and $Y = Y_1$ and show how to transform X and Y into two sets that satisfy all four properties of the lemma. Recall that X_1 still contains a perfect matching using edges in $G \setminus H$; denote this matching M_X^G. Our final set, however, will not guarantee such a perfect matching. Let M_X^H be a maximal matching in X using edges in H (with edges not incident to Y, because they already satisfy Property 4). Consider the edge set $E_X^* = M_X^G \cup M_X^H$.

We now perform the following simple transformation, we remove the endpoints of the edges in M_X^H from X and add them directly to Y. Property 1 is preserved because we are deleting and adding the same number of vertices from X and to Y respectively. Property 2 is preserved because the endpoints we add to Y are matched in M_X^H and thus in H. We will see later for Property 3.

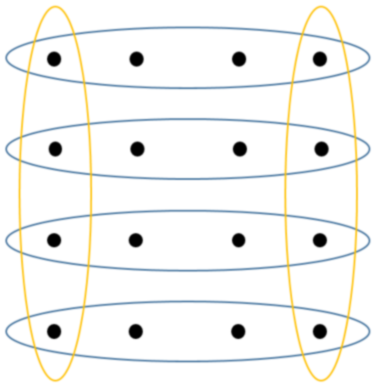

Figure C.1: Example with $d = 4$. In blue the edges of M_X^G and in yellow the edges of M_X^H

Let's check if Property 4 is preserved. To see this, note that we took M_H^X to be maximal among edges of H that contain only endpoints in X, and moved all the matched vertices in M_X^H to Y. Thus all vertices that remain in X are free in M_H^X, and so there are no edges between only endpoints in X after the transformation. There are also no edges in H containing endpoints from X and $V \setminus (X \cup Y)$, because originally they don't exist for $X = X_1$

Let's check if the Property 3 is preserved. As before, this involves showing that after the transformation, the average contribution of a vertex in $X \cup Y$ to σ is at least $\frac{1-5\lambda}{d}$. (Because every vertex in X is incident to an edge in E_X^*, each vertex is accounted for in the transformation.) Now, all vertices that were in Y_1 remain in Y , so their average contribution remains at $1/d$. We thus need to show that the average contribution to σ among vertices in X_1 remains at least $\frac{1-5\lambda}{d}$ after the transformation.

Let $n = |M_X^G|$ and $k = |M_X^H|$. Since M_X^G is a perfect matching on X_1, we always have $k \leq n$. If $n = k$, then all vertices of X_1 are transferred to Y and clearly their average contribution to σ is $\frac{1}{d} \geq \frac{1-5\lambda}{d}$. Now consider when $k \leq n - 1$. Let the edges of M_X^H be $\{e_1, \dots e_k\}$ and these of M_X^G be $\{e'_1, \dots e'_n\}$. Let X' be the set of vertices that remain in X_1. Because the edges of M_X^G are not H, the by two properties of HEDCS :

$$\begin{aligned}\sum_{1\le i\le n,\ x\in e'_i} d_H(x) &\ \ge\ n\beta(1-\lambda)\ , \text{ and}\\ \sum_{1\le i\le k,\ x\in e_i} d_H(x) &\ \le\ k\beta\ .\end{aligned}$$

The sum of the degrees of vertices in X' can be written as the following difference:

$$\begin{aligned}\sum_{x\in X'} d_H(x) &= \sum_{1\le i\le n,\ x\in e'_i} d_H(x) - \sum_{1\le i\le k,\ x\in e_i} d_H(x) \qquad \text{(C.1)}\\ &\ge n\beta(1-\lambda) - k\beta\\ &= (n-k)\cdot\beta - n\beta\lambda\ .\end{aligned}$$

Now we prove that the contribution of vertices from X' on average at least $\frac{1-5\lambda}{d}$. To do so, we need the following claim.

Claim C.1.1. $\sum_{x\in X'} \phi(d_H(x)) \ge (n-k) - 5n\cdot\lambda$.

By claim C.1.1, the average contribution among the vertices that are considered is

$$\frac{(n-k) - 5n\cdot\lambda + k}{nd} = \frac{1-5\lambda}{d}\ ,$$

which proves Property 3 and completes the proof of the lemma. □

Proof of claim C.1.1. Let's denote $m := n-k$. Recall that the number of vertices in X' is equal to md and $\phi(x) = \frac{d-1}{d}\frac{x}{\beta-x}$. Here we will prove it for $\lambda = 0$. This means that we will prove that

$$\sum_{x\in X'} d_H(x) \ge m\cdot\beta \Rightarrow \sum_{x\in X'} \phi(d_H(x)) \ge m\ .$$

If $m = n-k = 1$, then the result clearly holds by Lemma 4.6.2. Let's suppose $k < n-1$ and thus $m \geq 2$. By Lemma 4.6.2, we know that:

$$\frac{md-1}{md}\sum_{x\in X'}\frac{d_H(x)}{m\cdot\beta-d_H(x)} \geq 1\ . \tag{C.2}$$

We also know that :

$$\frac{m\beta-a}{\beta-a} = m+\frac{(m-1)a}{\beta-a}\ , \text{ and} \tag{C.3}$$

$$\frac{a}{\beta-a} = m\cdot\frac{a}{m\beta-a}+\frac{(m-1)a^2}{(m\beta-a)(\beta-a)}\ . \tag{C.4}$$

Combining (C.2) and (C.4), we get:

$$\sum_{x\in X'}\frac{d_H(x)}{\beta-d_H(x)} \geq \frac{m^2d}{md-1}+\sum_{x\in X'}\frac{(m-1)d_H(x)^2}{(m\beta-d_H(x))(\beta-d_H(x))}\ .$$

which leads to:

$$\begin{aligned}\sum_{x\in X'}\phi(d_H(x)) &\geq \frac{d-1}{d}\sum_{x\in X'}\frac{d_H(x)}{\beta-d_H(x)}\\ &\geq m\cdot\frac{m(d-1)}{md-1}+\frac{d-1}{d}\sum_{x\in X'}\frac{(m-1)d_H(x)^2}{(m\beta-d_H(x))(\beta-d_H(x))}\\ &\geq m+\frac{d-1}{d}\sum_{x\in X'}\frac{(m-1)d_H(x)^2}{(m\beta-d_H(x))(\beta-d_H(x))}-\frac{m-1}{md-1}\ .\end{aligned} \tag{C.5}$$

By convexity of the function $x \mapsto \frac{x^2}{(m\beta-x)(\beta-x)}$:

$$\begin{aligned}\sum_{x\in X'}\frac{d_H(x)^2}{(m\beta-d_H(x))(\beta-d_H(x))} &\geq md\frac{(\frac{\sum d_H(x))}{md})^2}{(m\beta-\frac{\sum d_H(x))}{md}(\beta-\frac{\sum d_H(x))}{md})}\\ &\geq \frac{m\beta}{(md-1)(d-1)}\ .\end{aligned} \tag{C.6}$$

Where the last inequality is due to $\sum_{x\in X'} d_H(x) \geq m\cdot\beta$

Therefore, when $\beta \geq \frac{d}{2}$ the right hand side of (C.5) becomes:

$$\begin{aligned} m + \frac{d-1}{d} \sum_{x \in X'} \frac{d_H(x)^2}{(m\beta - d_H(x))(\beta - d_H(x))} - \frac{1}{md-1} &\geq m + \frac{1}{md-1}\left(\frac{m\beta}{d} - 1\right) \\ &\geq m. \end{aligned}$$

□

C.1.3 Proof of Lemma 4.6.4

Lemma 4.6.4. *For any $x \in X$, $E[val'(x)] \geq (1-\lambda)\phi(d_H(x))$.*

Proof. We distinguish three cases:

- $d_H(x) \leq \frac{\beta}{d}$: in this case $\frac{1}{\beta - d_H(x)} \leq \frac{d}{(d-1)\beta} < \epsilon$ and $d_{H^*}(x) \leq d_H(x) \leq \beta - d_H(x)$. This implies that $val'(x) = \frac{d_{H^*}(x)}{\beta - d_H(x)}$, so that :

  $$E[val'(x)] \geq \frac{d-1}{d} \cdot \frac{d_H(x)}{\beta - d_H(x)} = \phi(d_H(x)) \ .$$

  Now consider the case in which $d_H(x) > \frac{\beta}{d}$. Then

  $$E[d_{H^*}(x)] \geq \frac{(d-1)d_H(x)}{d} > \frac{(d-1)}{d^2} \cdot \beta \geq 8 \cdot \lambda^{-3} >> \epsilon^{-1} \ ,$$

  because $\beta \geq \frac{8d^2}{d-1} \cdot \lambda^{-3}$, and by a Chernoff Bound :

  $$\begin{aligned} P\Big[d_{H^*}(x) < (1 - \frac{\lambda}{2})\frac{(d-1)d_H(x)}{d}\Big] &\leq \exp\left(-E[d_{H^*}(x)](\frac{\lambda}{2})^2\frac{1}{2}\right) \\ &\leq \exp\left(-\lambda^{-1}\right) \qquad \text{(C.7)} \\ &\leq \lambda/2 \ . \end{aligned}$$

- Let us now consider the case $d_H(x) > \frac{\beta}{d}$ and $\min\left\{\epsilon, \frac{1}{\beta - d_H(x)}\right\} = \epsilon$. With probability at least

$(1 - \frac{\lambda}{2})$, we have that:

$$d_{H^*}(x) \geq (\beta - \frac{1}{\epsilon})(1 - \frac{\lambda}{2})\frac{d-1}{d} >> \epsilon^{-1}.$$

Thus with probability at least $(1 - \frac{\lambda}{2})$, we have that $d_{H^*}(x)\epsilon > 1$ and:

$$E[val'(x)] \geq (1 - \frac{\lambda}{2}) \geq (1 - \frac{\lambda}{2})\phi(x).$$

- The only case that we need to check is $d_H(x) \geq \frac{\beta}{d}$ and $\min\left\{\epsilon, \frac{1}{\beta - d_H(x)}\right\} = \frac{1}{\beta - d_H(x)}$, so that $val'(x) = \min\left\{1, \frac{d_{H^*}(x)}{\beta - d_H(x)}\right\}$. Again we have that with probability at least $(1 - \frac{\lambda}{2})$:

$$\frac{d_{H^*}(x)}{\beta - d_H(x)} \geq \frac{d-1}{d}\frac{d_H(x)}{\beta - d_H(x)}(1 - \frac{\lambda}{2}) \geq (1 - \frac{\lambda}{2})\phi(d_H(x)).$$

In other words, with probability at least $(1 - \frac{\lambda}{2})$, we have $val(x) \geq (1 - \frac{\lambda}{2})\phi(d_H(x))$, so that $E[val(x)] \geq (1 - \frac{\lambda}{2})^2\phi(d_H(x)) > (1 - \lambda)\phi(d_H(x))$. We just showed that in all cases $E[val'(x)] \geq (1 - \lambda)\phi(d_H(x))$ □

C.2 Chernoff Bound

Let $X_1, \ldots, X_n$ be independent random variables taking value in $[0, 1]$ and $X := \sum_{i=1}^{n} X_i$. Then, for any $\delta \in (0, 1)$

$$Pr\Big(|X - \mathbb{E}[X]| \geq \delta\mathbb{E}[X]\Big) \geq 2 \cdot \exp\Big(-\frac{\delta^2\mathbb{E}[X]}{3}\Big).$$

C.3 Nesbitt's Inequality

Nesbitt's inequality states that for positive real numbers a, b and c,

$$\frac{a}{b+c} + \frac{b}{a+c} + \frac{c}{a+b} \geq \frac{3}{2},$$

with equality when all the variables are equal. And generally, if $a_1, \ldots, a_n$ are positive real numbers and $s = \sum_{i=1}^{n}$, then:

$$\sum_{i=1}^{n} \frac{a_i}{s - a_i} \geq \frac{n}{n-1},$$

with equality when all the a_i are equal.