2022

Sublinear Algorithm And Lower Bound For Combinatorial Problems

Yu Chen
University of Pennsylvania

Sublinear Algorithm And Lower Bound For Combinatorial Problems

Abstract

As the scale of the problems we want to solve in real life becomes larger, the input sizes of the problems we want to solve could be much larger than the memory of a single computer. In these cases, the classical algorithms may no longer be feasible options, even when they run in linear time and linear space, as the input size is too large.

In this thesis, we study various combinatorial problems in different computation models that process large input sizes using limited resources. In particular, we consider the query model, streaming model, and massively parallel computation model. In addition, we also study the tradeoffs between the adaptivity and performance of algorithms in these models.We first consider two graph problems, vertex coloring problem and metric traveling salesman problem (TSP). The main results are structure results for these problems, which give frameworks for achieving sublinear algorithms of these problems in different models. We also show that the sublinear algorithms for $(\Delta + 1)$-coloring problem are tight. We then consider the graph sparsification problem, which is an important technique for designing sublinear algorithms. We give proof of the existence of a linear size hypergraph cut sparsifier, along with a polynomial algorithm that calculates one. We also consider sublinear algorithms for this problem in the streaming and query models. Finally, we study the round complexity of submodular function minimization (SFM). In particular, we give a polynomial lower bound on the number of rounds we need to compute $s - t$ max flow - a special case of SFM - in the streaming model. We also prove a polynomial lower bound on the number of rounds we need to solve the general SFM problem in polynomial queries.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

Graduate Group

Computer and Information Science

First Advisor

Sampath Kannan

Second Advisor

Sanjeev Khanna

Subject Categories

Computer Sciences

SUBLINEAR ALGORITHM AND LOWER BOUND FOR COMBINATORIAL PROBLEMS

Yu Chen

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2022

Supervisor of Dissertation	Co-Supervisor of Dissertation
Sanjeev Khanna	Sampath Kannan
Henry Salvatori Professor of Computer and Information Science	Henry Salvatori Professor of Computer and Information Science

Graduate Group Chairperson

Mayur Naik, Professor of Computer and Information Science and Graduate Chair

Dissertation Committee

Anindya De (Chair), Assistant Professor of Computer and Information Science
Rajeev Alur, Zisman Family Professor of Computer and Information Science
Aaron Roth, Henry Salvatori Professor of Computer and Information Science
Deeparnab Chakrabarty, Associate Professor (Department of Computer Science, Dartmouth College)

ABSTRACT

SUBLINEAR ALGORITHM AND LOWER BOUND FOR COMBINATORIAL PROBLEMS

Yu Chen

Sanjeev Khanna

Sampath Kannan

As the scale of the problems we want to solve in real life becomes larger, the input sizes of the problems we want to solve could be much larger than the memory of a single computer. In these cases, the classical algorithms may no longer be feasible options, even when they run in linear time and linear space, as the input size is too large.

In this thesis, we study various combinatorial problems in different computation models that process large input sizes using limited resources. In particular, we consider the query model, streaming model, and massively parallel computation model. In addition, we also study the tradeoffs between the adaptivity and performance of algorithms in these models.

We first consider two graph problems, vertex coloring problem and metric traveling salesman problem (TSP). The main results are structure results for these problems, which give frameworks for achieving sublinear algorithms of these problems in different models. We also show that the sublinear algorithms for $(\Delta + 1)$-coloring problem are tight.

We then consider the graph sparsification problem, which is an important technique for designing sublinear algorithms. We give proof of the existence of a linear size hypergraph cut sparsifier, along with a polynomial algorithm that calculates one. We also consider sublinear algorithms for this problem in the streaming and query models.

Finally, we study the round complexity of submodular function minimization (SFM). In particular, we give a polynomial lower bound on the number of rounds we need to compute $s - t$ max flow - a special case of SFM - in the streaming model. We also prove a polynomial lower bound on the number of rounds we need to solve the general SFM problem in polynomial queries.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER 1

Introduction

Nowadays, the total amount of data created, captured, and consumed globally increases rapidly. Despite the storage capacity and computing power of modern computers increasing, their growth rates are far behind the amount of data that occurs every day. The emergence of massive data sets sometimes makes the traditional computing model unrealistic. For example, sometimes, the input size is much larger than the storage capacity of a single computer. In this scenario, we cannot assume that the computer has random access to the entire input, which is usually assumed by traditional algorithms. As a result, we need to consider *sublinear* computing models that use limited resources compared to the problem size. These resource includes *time*, *space* and *communication*.

In this thesis, we study several fundamental combinatorial problems in sublinear computing models. Combinatorial problems appear in various domains like computer science, data science, sociology, etc. Many combinatorial problems have many applications in different areas. For example, the submodular function optimization problem, along with its various important special cases such as minimum/maximum cut, maximum coverage, and matroid intersection, has found applications in areas such as information retrieval, image segmentation, and speech analysis. With the emergence of massive data sets, some of these problems considered tractable in the standard computing model (in other words, polynomial-time solvable) may no longer be efficiently solvable with limited resources. Thus, it is critical to study these problems in sublinear computation models. Can we solve these problems in limited time, limited space, or limited communication? Or can we prove that there are no such algorithms in these models?

Depending on the specific task and scenario, we need to consider different computational models. For example, when space is the primary resource, we consider the *streaming model*, where the goal is to design algorithms that use a small memory while processing a much

larger input. When time is the primary resource, we consider the *query model*, where the goal is to design algorithms that read a small fraction of the input and give an output by the limiting information we get. When the input is distributed in multiple machines, we consider different communication models such as the *Alice-Bob model* and the *massively parallel computation model* (MPC), where the goal is to design algorithms such that the machines only do a sublinear size of communication. So given a combinatorial problem, the question is: can we solve the problem using limited time/space/communication? Moreover, can we prove a matching lower bound on the time/space/communication needed to solve it?

In the rest of this chapter, we will first formally define the computational models in Section 1.1. Then in Section 1.2, we will describe the combinatorial problems we consider in this thesis, as well as some relevant previous work and our contributions. Finally, in Section 1.3, we give the organization of the rest of the thesis.

1.1. Computational Models

In this section, we introduce the three models we consider in this thesis. They are streaming model, query model, and massively parallel computation model. We introduce them and their background one by one in the following subsections. In Section 1.1.4, we discuss the tradeoff between the number of rounds an algorithm uses and the resources the algorithm need in each model.

1.1.1. Streaming Model

In the case when space is the primary resource, we consider the streaming model. In streaming model, the input is a sequence of items $\langle a_1, a_2, \ldots, a_m \rangle$ arriving one by one. Depending on the specific problem we are solving, the items can be different. For example, in graph problems, the items are usually edges in the graph. In statistics problems, the items could be the entries of a large vector or matrix. In the streaming model, the algorithm's memory size is much smaller than the total size of all items in the stream, so the algorithm needs to decide which information it wants to memorize on the fly. At the end of the stream, the algorithm needs to output the answer.

The streaming model is usually used in scenarios where the data are generated on the fly or are read in order. Two examples for the first scenario are the log files auto-generated during a task and the data transmitted by a satellite. For the second scenario, one example is when the input is stored in external memory.

In this thesis, we mainly focus on graph problems in the streaming model. For graph problems, the items in the stream are the edges in the graph. An algorithm is a *semi-streaming* if the algorithm uses $\tilde{O}(n) = n \cdot \mathsf{Polylog(n)}$ space where n is the number of vertices in the graph. The concept of semi-streaming algorithm is introduced by Feigenbaum *et al.* [117]. Since then, many fundamental graph problems have been studied in this setting. Some examples are maximum matching [108, 117, 188, 201], minimum cut [234], shortest path [43], maximal independent set [126] and minimum dominating set [146].

1.1.2. Query Model

In the standard computing model, the algorithm needs linear time to read the input once, so a linear time algorithm is usually considered the best we can do when designing an algorithm. However, when considering massive data sets, even reading the whole input once might take a lot of time and make the algorithm impractical. In this case, we consider the query model. In this model, the algorithm can make queries about the input through an oracle and then output the answer based on the oracle's results. For example, in the submodular function minimization problem with a gound set of size N, to fully describe the function may require $\Omega(2^N)$ bits. However, since we are guaranteed that the function is submodular, we do not need the full picture of the function to find the minimizer. In the query model, we allow the algorithm to query the value of a set. There are, in fact, algorithms that only use $\mathsf{Poly}(N)$ queries to find the minimizer [137, 162]. Other than submodular function minimization problem, query algorithms are also designed for submodular function maximization [25, 27, 196] and matroid intersection [59, 77].

In graph problems, we usually consider *pair query* or *neighbor query*. The former queries whether there is an edge between a pair of vertices, and the latter asks the oracle to return a

neighbor of a vertex. Although we usually allow both queries, pair queries are usually used when the graph is dense, and neighbor queries are usually used when the graph is sparse. The query model is first used to design sublinear algorithms for property testing on graphs [132, 133]. Later on, there are also sublinear algorithms designed for graph optimization problems, such as minimum spanning tree [83] and maximum matching [44, 171, 218, 220, 255].

For the problems defined in a metric space, we also usually use *pair query*. The algorithm can ask for the distance between two points in the metric space in this setting. Indyk [155] designed sublinear algorithms for several metric space problems such as k-median and maximum TSP. Sublinear algorithms have also been designed for metric MST [100] and metric k-nearest neighbor [101].

1.1.3. Massively Parallel Computing Model

As we discussed in the previous sections, the main challenge of massive data sets is that the input size might be much larger than the storage of a single computer. In this case, we store the input in multiple computers. To solve the problem, different computers communicate with each other to figure out the answer. There are different models to address this kind of situation. The differences among these models include how the input is distributed, how the computers communicate, and what amount of information a computer can send/receive in a certain amount of time.

In this thesis, we focus on the *massivly parallel computing (MPC)* model. In this model, each machine has sublinear size memories. The computation proceeds in synchronous rounds. Each machine first does computations on its own data and then sends messages to other computers in each round. These messages will arrive at the start of the next round. The constraint is, in each round, the total size of the message a computer sends or receives cannot larger than its own memory. The main goal is to minimize the number of rounds needed to compute the answer. MPC model is first introduced in [179], and refined by a series of work [12, 42, 135]. It has been studied for many combinatorial problems such as graph connectivity [47, 184, 229], maximum matching [15, 45, 48, 99, 128] and maximal

independent set [45, 126, 187].

1.1.4. Tradeoff between Round Complexity and Computational Resources

One important aspect of sublinear models is the adaptivity of the algorithms. For example, suppose two query algorithms both use N queries. In the first algorithm, each query it asks depends on the previous queries' answer, and in the second algorithm, each query does not depend on the previous answers. If the oracle could simultaneously process a large number of queries simultaneously (or simplily there are a lot of oracles available), the second algorithm would be much faster than the first one since it can ask its queries simultaneously. Therefore, in some scenarios, the algorithms with less adaptivity are preferable even if they make more queries. In this thesis, we study the tradeoff between the adaptivity of the algorithms and the resources they use in the streaming and query models.

In the streaming model, the tradeoff is between the number of passes the algorithm reads the stream and the memory size the algorithm uses. For many problems, allowing for multiple passes over the stream greatly enhances the capability of stream algorithms. A striking example is the (global) minimum cut problem: While $\Omega(n^2)$ space is needed for computing an exact minimum cut in a single pass [256], a recent result of [234] implies that a minimum cut of an undirected unweighted graph can be computed in $\widetilde{O}(n)$ space in only two passes over the stream[1]. Table 1.1 presents several other examples of this phenomenon.

In the query model, the tradeoff is between the number of rounds of the algorithm and the total number of queries the algorithm uses. In an r-round algorithm, the queries being asked in round i only depend on the answers of the queries in the first $i-1$ rounds, and the algorithm outputs the answer at the end of round r. The rounds-of-adaptivity versus query complexity question has seen a lot of work on different problems such as submodular function maximization [71, 89], monotone submodular function minimization with cardinality constraint [25, 27, 86, 87, 111, 112, 114, 196], and convex function minimization [70, 105, 217].

[1] The result of [234] is not stated as a streaming algorithm. However, the algorithm in [234] combined with the known graph streaming algorithms for cut sparsifiers (see, e.g. [202]) immediately imply the claimed result.

Problem	Multi-Pass				Single-Pass		
	Space	**Apx**	**Passes**	**Ref**	**Space**	**Apx**	**Ref**
Unweighted Min-Cut	$\widetilde{O}(n)$	1	2	[234]	$\Omega(n^2)$	1	[256]
Unweighted s-t Min-Cut	$\widetilde{O}(n^{5/3})$	1	2	[234]	$\Omega(n^2)$	1	[256]
Triangle Counting	$\widetilde{O}(\frac{m^{3/2}}{T})$	$1+\varepsilon$	4	[52]	$\Omega(\frac{m^3}{T^2})$	$\Theta(1)$	[190]
Maximum Matching	$\widetilde{O}(n)$	$1+\varepsilon$	$O(1)$	[201]	$n^{1+\Omega(\frac{1}{\log\log n})}$	$\frac{1}{1+\ln 2}$	[167]
Single Source Shortest Path	$\widetilde{O}(n)$	$1+\varepsilon$	$O(1)$	[43]	$\Omega(n^2)$	$\frac{5}{3}$	[118]
Maximal Independent Set	$\widetilde{O}(n)$	–	$O(\log\log n)$	[126]	$\Omega(n^2)$	–	[18]
Minimum Dominating Set	$\widetilde{O}(n)$	$O(\log n)$	$O(\log n)$	[146]	$n^{2-o(1)}$	$n^{o(1)}$	[19]

Table 1.1: A sample of multi-pass graph streaming algorithms and corresponding single-pass lower bounds. All results are for graphs $G(V, E)$ with n vertices and m edges (and T triangles).

1.2. Our Contributions

In this thesis, we consider the following problems in sublinear settings:

- $(\Delta+1)$-coloring problem;
- Traveling salsman problem;
- Hypergraph cut sparisification problem;
- s-t minimum cut problem;
- Lexicographically-first maximal independent set problem
- Submodular function minimization problem;

In this section, we give an overview of the results we have, and breifly discuss the related background.

1.2.1. Sublinear Algorithms for $(\Delta+1)$-Coloring Problem

A proper c-coloring of a graph $G(V,E)$ assigns a color to every vertex from the palette of colors $\{1,\ldots,c\}$ such that no edge is monochromatic, i.e., has the same color on both endpoints. A celebrated case of graph coloring is the $(\Delta+1)$ coloring problem where Δ is the maximum degree of the graph. Not only does every graph admit a $(\Delta+1)$ coloring, remarkably, *any* partial coloring of vertices of a graph can be extended to a proper $(\Delta+1)$ coloring of all vertices: simply pick uncolored vertices in any order and assign a color to a vertex not yet assigned to any of its neighbors; since the max-degree is Δ, such a color always exists. Due to these reasonings, $(\Delta+1)$-coloring problem admits a greedy algorithm that runs in linear time and space - simply color the vertices one by one. In this thesis, we study $(\Delta+1)$-coloring problem in sublinear models. We give sublinear algorithms for all three models we discussed in Section 1.1. At the core of our results is a remarkably simple meta-algorithm for the $(\Delta+1)$ coloring problem that is based on a key sparsification result for this problem that we establish. We prove that, if each vertex in the graph randomly samples $O(\log n)$ colors, the graph still admits a $(\Delta+1)$-coloring scheme with high probability where each vertex is restricted to use one of the colors it sampled. The sublinear algorithms are then obtained by efficiently implementing this meta-algorithm in each model separately. In particular, we design sublinear algorithms that find a $(\Delta+1)$-coloring scheme of a graph:

1. A single pass streaming algorithm that uses $\tilde{O}(n)$ space.
2. A query algorithm that uses $\tilde{O}(n^{1.5})$ queries.
3. An MPC algorithm that runs in two MPC rounds on machines with memory $\tilde{\Omega}(n)$.

Furthermore, our algorithms obtain essentially optimal bounds in each model considered. Indeed, space-complexity of our streaming algorithm and round-complexity of our MPC algorithm in are clearly optimal (to within polylog factors and constant factors, respectively). We also prove that query and time complexity of our sublinear time algorithm is also optimal up to polylog factors.

These results are based on the work I did with Sepehr Assadi and Sanjeev Khanna [18].

1.2.2. Sublinear Algorithms for Metric Traveling Salesman Problem

In the metric traveling salesman problem (TSP), we are given n points in an arbitrary metric space with an $n \times n$ matrix D specifying pairwise distances between them. The goal is to find a simple cycle (a TSP tour) of minimum cost that visits all n points. An equivalent view of the problem is that we are given a complete weighted undirected graph $G(V, E)$ where the weights satisfy triangle inequality, and the goal is to find a Hamiltonian cycle of minimum weight. The study of metric TSP is intimately connected to many algorithmic developments, and the poly-time approximability of metric TSP and its many natural variants are a subject of extensive ongoing research (see, for instance, [11, 124, 178, 183, 204, 210, 238, 239, 242, 243, 246] and references within for some relatively recent developments).

In this thesis, we consider metric TSP in query model and streaming model. A standard approach to estimating the metric TSP cost is to compute the cost of a minimum spanning tree, and output two times this cost as the estimate of the TSP cost (since any spanning tree can be used to create a spanning simple cycle by at most doubling the cost). In the query model, the the cost of the minimum spanning tree can be approximated to within a factor of $(1+\varepsilon)$ in $\tilde{O}(n)$ queries [100], and in streaming model, there is a natural $O(n)$-space algorithm that computes a minimum spanning tree. Therefore, in both model, the cost of metric TSP could be approximated to within a fact of $(2+\varepsilon)$ in near linear queries/space. The problem is, can we break the barrier of 2 in $o(n^2)$ queries/space?

We consider two well-studied special cases of the traveling salesman problem. The first case is graphic TSP, in which the metric is induced by an unweighted graph. The second case is 1-2 TSP, in which the distance between any pair of vertices is 1 or 2. We give a $\tilde{O}(n)$ query algorithm and a $\tilde{O}(n)$ space single-pass streaming algorithm that approximate the optimal TSP to within a factor of $(2 - O(1))$ for each of these two cases. In addition, we also prove a $\tilde{\Omega}(n^2)$ query lower bound for approximating TSP size to within a factor of $(1 + O(1))$ in these two special cases and hence the general case.

These results are based on the work I did with Sampath Kannan and Sanjeev Khanna [90].

1.2.3. Hypergraph Cut Sparsifier

When designing sublinear algorithms for graph problems, one common method is builing a compressed representation that preserves relevant properties of the graph. Cuts in graphs are a fundamental object of study, and play a central role in the study of graph algorithms. Consequently, the problem of *sparsifying* a graph while approximately preserving its cut structure has been extensively studied (see, for instance, [4, 6, 7, 29, 41, 50, 130, 170, 172, 174, 175, 195, 241], and references therein). A cut-preserving sparsifier not only reduces the space requirement for any computation, but it can also reduce the time complexity of solving many fundamental cut, flow, and matching problems as one can now run the algorithms on the sparsifier which may contain far fewer edges. In a seminal work, Benczúr and Karger [50] showed that given any n-vertex undirected weighted graph G and a parameter $\varepsilon \in (0,1)$, there is a near-linear time algorithm that outputs a weighted subgraph G' of G of size $\tilde{O}(n/\varepsilon^2)$ such that the weight of every cut in G is preserved to within a multiplicative $(1 \pm \varepsilon)$-factor in G'. The graph G' is referred to as the $(1 \pm \varepsilon)$-*approximate cut sparsifier* of G.

In this thesis, we consider the problem of cut sparsification for hypergraphs. A hypergraph $H(V, E)$ consists of a vertex set V and a set E of hyperedges where each edge $e \in E$ is a subset of vertices. The rank of a hypergraph is the size of the largest edge in the hypergraph, that is, $\max_{e \in E} |e|$. Hypergraphs are a natural generalization of graphs and many applications require estimating cuts in hypergraphs (see, for instance, [72, 73, 153, 250]). Note that unlike graphs, an n-vertex hypergraph may contain exponentially many (in n) hyperedges.

We prove that any hypergraph has a cut sparsifier with $\tilde{O}(n/\varepsilon^2)$ edges, and also give a polynomial algorithm that constructs one. This result improves the the size of hypergraph cut sparsifier from $\tilde{O}(n^2/\varepsilon^2)$ given by Kogan and Krauthgamer [185] and $\tilde{O}(nr^2/\varepsilon^2)$ given by Chekuri and Xu [88] where r is the maximum size of the hyperedges. Moreover, the space bound $\tilde{O}(n^2)$ (each hyperedge may have size $\Theta(n)$) of the cut sparsifier is also the best

possible to within a logarithmic factor due to a recent work Kapralov *et al.* [169].

On the other hand, since a hyperedge can contain subset of vertices of any size, there might be exponential number of hyperedge in a hypergraph. just reading the whole graph might take exponential time. With the same coauthors, we study sublinear algorithms for hypergraph sparsification using suitable oracle access to the input hypergraph. We consider the problem in the cut query model, where we can query the size of a cut. In this model, we give an algorithm that constructs a hypergraph cut sparsifier whose running time is polynomial in the number of vertices and independent of the number of hyperedges. We also show that the algorithm can be generalized to compute hypergraph spectral sparsifications.

These results are based on the works I did with Sanjeev Khanna and Ansh Nagda [91, 92].

1.2.4. A New Tool for Proving Round Complexity in Streaming Model

A vast body of work in graph streaming lower bounds concerns algorithms that make only one or a few passes over the stream. These lower bounds are almost always obtained by considering communication complexity of the problem with *limited number of rounds* of communication which gives a lower bound on the space complexity of streaming algorithms with proportional number of passes to the limits on rounds of communication (see e.g. [10, 138]). The communication lower bounds are then typically proved via reductions from (variants of) the *pointer chasing* problem [74, 219, 223] for multi-pass lower bounds and the *indexing* problem [2, 189] and *boolean hidden (hyper-)matching* problem [125, 244] for single-pass lower bounds.

In the pointer chasing problem, Alice and Bob are given functions $f, g : [n] \to [n]$ and the goal is to compute $f(g(\cdots f(g(0))))$ for k iterations. Computing this function in less than k rounds requires $\widetilde{\Omega}(n/k)$ communication [254] (see also [106, 219, 223, 228]). The reductions from pointer chasing to graph streaming lower bounds are based on using vertices of the graph to encode $[n]$ and each edge to encode a pointer [118, 141]. Directly using pointer chasing does not imply lower bounds stronger than $\Omega(n)$ and hence variants of pointer

chasing with multiple pointers such as multi-valued pointer chasing [118, 160] and set pointer chasing [141], were considered. Using multiple pointers however has the undesired side effect that the lower bound deteriorates exponentially with number of rounds. As such, these lower bounds do not go beyond $O(\log n)$ passes even for algorithms with $O(n)$ space.

There are however a number of results that prove lower bounds for a very large number of passes (even close to n). Examples include lower bounds for approximating clique and independent set [145], approximating dominating set [14], computing girth [118], estimating the number of triangles [31, 52, 96, 164], and finding minimum vertex cover or coloring [1]. These results are all proven by considering the communication complexity of the problem with *no limits on rounds* of communication. Such bounds then imply lower bounds on the product of space and number of passes of streaming algorithms (see, e.g. [10]). The communication lower bounds themselves are proven by reductions from a handful of communication problems, mainly the *set disjointness* problem [24, 30, 166, 230].

This approach suffers from two main drawbacks. Firstly, these lower bounds only exhibit space bounds that scale with the reciprocal of the number of passes and are hence unable to capture more nuanced space/pass trade-offs. More importantly, there is an inherit limitation to this approach since the computational model considered here is much stronger than the streaming model.

In this thesis, we introduce and analyze a new communication problem similar in spirit to standard pointer chasing, which we refer to as the *hidden-pointer chasing* (HPC) problem. What differentiate HPC from previous variants of pointer chasing is that the pointers are "hidden" from players and finding each one of them requires solving another communication problem, namely the *set intersection* problem, in which the goal is to *find* the *unique* element in the intersection of players input. There are four players in HPC paired into groups of size two each. Each pair of players inside a group shares n instances of the set intersection problem on n elements. The intersecting element in each instance of each group "points" to an instance in the other group. The goal is to start from a fixed instance and follow these

pointers for a fixed number of steps.

We prove that any r-round protocol that solves HPC with $(r+1)-th$ iteration requires $\Omega(n^2/r^2)$ communication. Given this result, we are able to prove lower bound on the number of passes we need to solve graph problems with sublinear space. In particular, we prove that for minimum s-t cut problem and lexicographically-first maximal independent set problem, any p-pass streaming algorithm that solves these problems requires $\Omega(n^2/p^5)$ space.

These results are based on the work I did with Sepehr Assadi and Sanjeev Khanna [17].

1.2.5. Round Complexity of Submodular Function Minimization Problem

A function $f : 2^U \to \mathbb{Z}$ defined over subsets of a ground set U of N elements is submodular if for any two sets $A \subseteq B$ and an element $e \notin B$, the *marginal* of e on A, that is, $f(A \cup e) - f(A)$ is at least $f(B \cup e) - f(B)$. The submodular function minimization (SFM) problem is to find a subset S minimizing $f(S)$ given access to an evaluation oracle for the function that returns the function value on any specified subset. SFM is a fundamental discrete optimization problem which generalizes classic problems such as minimizing global and s-t cuts in graphs and hypergraphs, and more recently has found applications in areas such as image segmentation [62, 63, 186] and speech analysis [158, 159].

A remarkable fact is that SFM *can* be solved in polynomial time with polynomially many queries to the evaluation oracle. This was first established by Grötschel, Lovász, and Schrijver [137] using the ellipsoid method. Since then, a lot of work [23, 76, 78, 98, 102, 156, 157, 162, 191, 194, 221, 236] has been done trying to understand the query complexity of SFM. The current best known algorithms are an $O(N^3)$-query polynomial-time and an $O(N^2 \log N)$-query exponential time algorithm by Jiang [162] building on the works [102, 194], an $\tilde{O}(N^2 \log M)$-query and time algorithm by Lee, Sidford, and Wong [194] where $|f(S)| \leq M$ for all $S \subseteq U$, and an $\tilde{O}(NM^2)$ query and time algorithm by Axelrod, Liu, and Sidford [23] improving upon [78].

Any SFM algorithm accesses the evaluation oracle in rounds, where the queries made in a

certain round depend only on the answers to queries made in previous rounds. There is a trade-off between the number of queries (per round) made by the algorithm, and the number of rounds needed to find the answer : there is an obvious 1-round algorithm which makes all 2^N queries. All known efficient algorithms for SFM described above are *highly sequential*; all of them proceed in $\Omega(N)$ rounds. From a practical standpoint, given the applications of SFM to problems involving huge data and the availability of computing infrastructure to perform parallel computation, the question of low-depth parallel SFM algorithms is timely.

In this thesis, we prove that we prove that any SFM algorithm with polynomial calls to the function evaluation oracle needs $\tilde{\Omega}(N^{1/3})$ rounds of adaptivity, which is an exponential improvement of the previous bound $\Omega(\frac{\log N}{\log \log N})$ due to Balkanski and Singer [28]. Furthermore, we prove that if we restrict the value of the submodular function in [-1,1], any polynomial algorithm that obtains an additive ε-approximation to the minimum function value needs $\tilde{\Omega}(1/\varepsilon)$ rounds of adaptivity. The instance we use in the lower bound can also give the same lower bound for the round complexity of the matroid intersection problem, a special case of submodular function minimization.

1.3. Organization

In the rest of the thesis, we will give the high level ideas of the result we mentioned before. We will include the full details in the full thesis. In Chapter 2, we set up the basic notations and list common tools we use throughout the thesis. In Chapter 3, Chapter 4, we discuss the sublinear algorithms of $(\Delta + 1)$-coloring and traveling salseman problem. In Chapter 5 is about the hypergraph sparsifier. In Chapter 6, we prove the communication complexity of the hidden point chasing problem, and give the reduction to maximum flow problem. In Chapter 7, we discuss the round complexity of submodular function minimization. In Chapter 8, we summerize the results of this thesis and discuss some open problems.

CHAPTER 2

Background

In this section, we set up the basic notations and list common tools we use throughout the thesis.

2.1. Graphs and Hypergraphs

For any $t \geq 1$, we define $[t] := \{1, \ldots, t\}$. For a tuple $(X_1, \ldots, X_n)$ and integer $i \in [n]$, $X^{<i} := (X_1, \ldots, X_{i-1})$ and $X_{-i} := (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$.

For a graph $G = (V, E)$, we use $V(G) := V$ to denote the vertices and $E(G) := E$ to denote the edges. For every vertex $v \in V$, $N(v)$ denote the set of neighbors of v and $\deg(v) := |N(v)|$ denotes the degree of v. For a set U of vertices, $G[U]$ denote the induced subgraph of G on U.

We use the following standard graph theory facts.

Fact 2.1.1 (Hall's Theorem)**.** *Suppose $G = (L, R, E)$ is a bipartite graph s.t. $\forall S \subseteq L$, $|N(S)| \geq |S|$. Then there exists a matching in G that matches every vertex in L, i.e., a* **left-saturating** *matching.*

Given any weight function $w : S \to \mathbb{R}_{\geq 0}$, we extend it to also be a function on subsets of S so that $w(S') = \sum_{e \in S'} w(e)$ for $S' \subseteq S$.

For a *weighted graph* $G = (V, E, w)$, $w : E(G) \to \mathbb{R}^+$ is a weight function on edges in G. For a subgraph $H \subseteq G$, we define $w(H) = \sum_{e \in E(H)} w(e)$.

A *hypergraph* is defined as a pair (V, E) of vertices and edges, where each edge in E is a subset of V. In this thesis, we allow parallel edges (that is, E is a multiset). To emphasize this, we often refer to a graph/hypergraph as a multigraph/multihypergraph. Given a weight function w that assigns a nonnegative weight to each edge in E, the triple (V, E, w) is a weighted hypergraph. Notice that an unweighted graph/hypergraph can be thought of as a

weighted graph/hypergraph with all weights equal to 1.

In this thesis, we use "graph" to refer to standard graphs with edges of size 2, and "hypergraph" to refer to graphs where edge sizes are arbitrary. We generally use the symbol G to refer to standard graphs, and H to refer to hypergraphs.

A cut $C = (S, \bar{S})$ of a vertex set V is any disjoint partition of V into two sets such that neither of the sets are empty. Given a graph/hypergraph $G = (V, E, w)$ and a cut $C = (S, \bar{S})$, we denote by $\delta_G(S)$ the set of the edges crossing the cut C in G. By definition, $|\delta(S)|$ is the number of edges crossing C and $w(\delta(S))$ is the weight/size of C. A $(1 \pm \varepsilon)$-approximate cut sparsifier of G is a graph/hypergraph $G' = (V, E', w')$ with $E' \subseteq E$ such that

$$\forall S \subseteq V, \quad \left|w'(\delta_{G'}(S)) - w(\delta_G(S))\right| \leq \varepsilon w(\delta_G(S)).$$

2.2. Inequalities and Concentration Bounds

Proposition 2.2.1. *For any two lists of numbers $a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_1 \geq b_2 \geq \cdots \geq b_n$, $\sum_{i=1}^n a_i b_i \leq \frac{1}{n} \sum_{i=1}^n a_i \cdot \sum_{i=1}^n b_i$.*

Proof. The rearrangement inequality [147] states that for any list of numbers $x_1 \leq \cdots \leq x_n$ and $y_1 \leq \cdots \leq y_n$ and any permutation σ of $[n]$,

$$x_1 \cdot y_n + \cdots + x_n \cdot y_1 \leq x_1 \cdot y_{\sigma(1)} + \cdots + x_n \cdot y_{\sigma(n)} \leq x_1 \cdot y_1 + \cdots + x_n \cdot y_n.$$

By rearrangement inequality, for any $0 \leq j < n$,

$$\sum_{i=1}^{n} a_i b_i \leq \sum_{i=1}^{n} a_i b_{i+j},$$

where, with a slight abuse of notation, we use b_{i+j} for $i + j > n$ to denote b_{i+j-n}. As such,

$$\sum_{i=1}^{n} a_i b_i \leq \frac{1}{n} \sum_{j=0}^{n-1} \sum_{i=1}^{n} a_i b_{i+j} = \frac{1}{n} \sum_{i=1}^{n} (a_i \sum_{j=0}^{n-1} b_{i+j}) = \frac{1}{n} \sum_{i=1}^{n} a_i \cdot \sum_{i=1}^{n} b_i \quad \square$$

□

Proposition 2.2.2 (Chernoff-Hoeffding bound)**.** *Let $X_1, \ldots, X_n$ be n independent random variables where each $X_i \in [0,1]$ and $X := \sum_{i=1}^{n} X_i$. For any $\delta \in (0,1)$,*

$$\Pr\Big(\,|X - \mathbb{E}[X]| > \delta \cdot \mathbb{E}[X]\,\Big) \leq 2 \cdot \exp\left(-\frac{\delta^2 \cdot \mathbb{E}[X]}{3}\right).$$

Lemma 2.2.3 (Theorem 2.2 in [123])**.** *Let $\{x_1, \ldots, x_k\}$ be a set of random variables, such that for $1 \leq i \leq k$, each x_i independently takes value $1/p_i$ with probability p_i and 0 otherwise, for some $p_i \in [0,1]$. Then for all $N \geq k$ and $\varepsilon \in (0,1]$,*

$$\Pr\left(\left|\sum_{i\in[k]} x_i - k\right| \geq \varepsilon N\right) \leq 2e^{-0.38\varepsilon^2 \cdot \min_i p_i \cdot N}$$

A function $f(x_1, \ldots, x_n)$ is called *c-Lipschitz* iff changing any single x_i can affect the value of f by at most c. Additionally, f is called *r-certifiable* iff whenever $f(x_1, \ldots, x_n) \geq s$, there exists at most $r \cdot s$ variables $x_{i_1}, \ldots, x_{i_{r\cdot s}}$ so that knowing the values of these variables certifies $f \geq s$.

Proposition 2.2.4 (Talagrand's inequality; cf. [205])**.** *Let $X_1, \ldots, X_n$ be n independent random variables and $f(X_1, \ldots, X_n)$ be a c-Lipschitz and r-certifiable function. For any $t \geq 1$,*

$$\Pr\Big(|f - \mathbb{E}[f]| > t + 30c\sqrt{r \cdot \mathbb{E}[f]}\Big) \leq 4\exp\left(-\frac{t^2}{8c^2 r\,\mathbb{E}[f]}\right).$$

2.3. Submodular Function Minimization

A function $f : 2^U \to \mathbb{Z}$ defined over subsets of a ground set U of N elements is submodular if for any two sets $A \subseteq B$ and an element $e \notin B$, the *marginal* of e on A, that is, $f(A \cup e) - f(A)$ is at least $f(B \cup e) - f(B)$. The submodular function minimization (SFM) problem is to find

a subset S minimizing $f(S)$ given access to an evaluation oracle for the function that returns the function value on any specified subset. There is an algorithm that for any submodular function f finds a set S that minimizes the value of function f by $\tilde{O}(N^3)$ queries and $\tilde{O}(N^4)$ time.

Theorem 2.1 ([194]). *There is an algorithm for submodular function minimization with $O(n^3 \log^2 n)$ queries and $O(n^4 \log^{O(1)} n)$ time where n is the size of the ground set.*

Given a graph/hypergraph, for any vertex set S, the cut function $f(S)$, defined as the weight of edges crossing cut $(S, \bar{S})$ is easily shown to be submodular.

2.4. Information Theory

For random variables X, Y, $\mathbb{H}(\mathsf{X})$ denotes the Shannon entropy of X and $\mathbb{I}(\mathsf{X}\,;\mathsf{Y})$ denotes the mutual information. For distributions μ, ν, $\mathbb{D}(\mu \mid\mid \nu)$ denotes the KL-divergence, $\|\mu - \nu\|_{tvd}$ denotes the total variation distance, and $\mathrm{h}(\mu, \nu)$ denotes the Hellinger distance.

For a random variable A, we use $\textsc{supp}(\mathsf{A})$ to denote the support of A and $\mathrm{dist}(\mathsf{A})$ to denote its distribution. When it is clear from the context, we may abuse the notation and use A directly instead of $\mathrm{dist}(\mathsf{A})$, for example, write $A \sim \mathsf{A}$ to mean $A \sim \mathrm{dist}(\mathsf{A})$, i.e., A is sampled from the distribution of random variable A. We denote the *Shannon Entropy* of a random variable A by $\mathbb{H}(\mathsf{A})$, which is defined as:

$$\mathbb{H}(\mathsf{A}) := \sum_{A \in \textsc{supp}(\mathsf{A})} \Pr(\mathsf{A} = A) \cdot \log\left(1/\Pr(\mathsf{A} = A)\right) \tag{2.1}$$

The *conditional entropy* of A conditioned on B is denoted by $\mathbb{H}(\mathsf{A} \mid \mathsf{B})$ and defined as:

$$\mathbb{H}(\mathsf{A} \mid \mathsf{B}) := \mathop{\mathbb{E}}_{B \sim \mathsf{B}} \left[\mathbb{H}(\mathsf{A} \mid \mathsf{B} = B)\right], \tag{2.2}$$

where $\mathbb{H}(\mathsf{A} \mid \mathsf{B} = B)$ is defined in a standard way by using the distribution of A conditioned on the event $\mathsf{B} = B$ in Eq (2.1). The *mutual information* of two random variables A and B

is denoted by $\mathbb{I}(\mathsf{A}\,;\mathsf{B})$ and is defined as:

$$\mathbb{I}(\mathsf{A}\,;\mathsf{B}) := \mathbb{H}(\mathsf{A}) - \mathbb{H}(\mathsf{A} \mid \mathsf{B}) = \mathbb{H}(\mathsf{B}) - \mathbb{H}(\mathsf{B} \mid \mathsf{A}) = \mathbb{I}(\mathsf{B}\,;\mathsf{A}). \tag{2.3}$$

The *conditional mutual information* $\mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C})$ is $\mathbb{H}(\mathsf{A} \mid \mathsf{C}) - \mathbb{H}(\mathsf{A} \mid \mathsf{B},\mathsf{C})$ and hence by linearity of expectation:

$$\mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}) = \mathop{\mathbb{E}}_{C\sim\mathsf{C}} \left[\mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C} = C)\right]. \tag{2.4}$$

When it may lead to confusion, we use the subscript dist in $\mathbb{H}_{\text{dist}}$ and $\mathbb{I}_{\text{dist}}$ to mean that the random variables in these terms are distributed according to the distribution dist.

Useful Properties of Entropy and Mutual Information

We shall use the following basic properties of entropy and mutual information throughout.

Fact 2.4.1 (cf. [97]; Chapter 2)**.** *Let* A*,* B*,* C*, and* D *be four (possibly correlated) random variables.*

1. $0 \le \mathbb{H}(\mathsf{A}) \le \log|\textsc{supp}(\mathsf{A})|$*. The right equality holds iff* $\text{dist}(\mathsf{A})$ *is uniform.*
2. $\mathbb{I}(\mathsf{A}\,;\mathsf{B}) \ge 0$*. The equality holds iff* A *and* B *are* independent*.*
3. Conditioning on a random variable can only reduce the entropy*:* $\mathbb{H}(\mathsf{A} \mid \mathsf{B},\mathsf{C}) \le \mathbb{H}(\mathsf{A} \mid \mathsf{B})$*. The equality holds iff* $\mathsf{A} \perp \mathsf{C} \mid \mathsf{B}$*.*
4. Subadditivity of entropy*:* $\mathbb{H}(\mathsf{A},\mathsf{B} \mid \mathsf{C}) \le \mathbb{H}(\mathsf{A} \mid C) + \mathbb{H}(\mathsf{B} \mid \mathsf{C})$*.*
5. Chain rule for entropy*:* $\mathbb{H}(\mathsf{A},\mathsf{B} \mid \mathsf{C}) = \mathbb{H}(\mathsf{A} \mid \mathsf{C}) + \mathbb{H}(\mathsf{B} \mid \mathsf{C},\mathsf{A})$*.*
6. Chain rule for mutual information*:* $\mathbb{I}(\mathsf{A},\mathsf{B}\,;\mathsf{C} \mid \mathsf{D}) = \mathbb{I}(\mathsf{A}\,;\mathsf{C} \mid \mathsf{D}) + \mathbb{I}(\mathsf{B}\,;\mathsf{C} \mid \mathsf{A},\mathsf{D})$*.*

We also use the following two standard propositions regarding the effect of conditioning on mutual information.

Proposition 2.4.2. *For random variables* $\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}$*, if* $\mathsf{A} \perp \mathsf{D} \mid \mathsf{C}$*, then,*

$$\mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}) \leq \mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}, \mathsf{D}).$$

Proof. Since A and D are independent conditioned on C, by Fact 2.4.1-(3), $\mathbb{H}(\mathsf{A} \mid \mathsf{C}) = \mathbb{H}(\mathsf{A} \mid \mathsf{C}, \mathsf{D})$ and $\mathbb{H}(\mathsf{A} \mid \mathsf{C}, \mathsf{B}) \geq \mathbb{H}(\mathsf{A} \mid \mathsf{C}, \mathsf{B}, \mathsf{D})$. We have,

$$\begin{aligned}\mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}) &= \mathbb{H}(\mathsf{A} \mid \mathsf{C}) - \mathbb{H}(\mathsf{A} \mid \mathsf{C}, \mathsf{B}) = \mathbb{H}(\mathsf{A} \mid \mathsf{C}, \mathsf{D}) - \mathbb{H}(\mathsf{A} \mid \mathsf{C}, \mathsf{B}) \\ &\leq \mathbb{H}(\mathsf{A} \mid \mathsf{C}, \mathsf{D}) - \mathbb{H}(\mathsf{A} \mid \mathsf{C}, \mathsf{B}, \mathsf{D}) = \mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}, \mathsf{D}). \quad \square\end{aligned}$$

□

Proposition 2.4.3. *For random variables* $\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}$*, if* $\mathsf{A} \perp \mathsf{D} \mid \mathsf{B}, \mathsf{C}$*, then,*

$$\mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}) \geq \mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}, \mathsf{D}).$$

Proof. Since $\mathsf{A} \perp \mathsf{D} \mid \mathsf{B}, \mathsf{C}$, by Fact 2.4.1-(3), $\mathbb{H}(\mathsf{A} \mid \mathsf{B}, \mathsf{C}) = \mathbb{H}(\mathsf{A} \mid \mathsf{B}, \mathsf{C}, \mathsf{D})$. Moreover, since conditioning can only reduce the entropy (again by Fact 2.4.1-(3)),

$$\begin{aligned}\mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}) &= \mathbb{H}(\mathsf{A} \mid \mathsf{C}) - \mathbb{H}(\mathsf{A} \mid \mathsf{B}, \mathsf{C}) \geq \mathbb{H}(\mathsf{A} \mid \mathsf{D}, \mathsf{C}) - \mathbb{H}(\mathsf{A} \mid \mathsf{B}, \mathsf{C}) \\ &= \mathbb{H}(\mathsf{A} \mid \mathsf{D}, \mathsf{C}) - \mathbb{H}(\mathsf{A} \mid \mathsf{B}, \mathsf{C}, \mathsf{D}) = \mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}, \mathsf{D}). \quad \square\end{aligned}$$

□

Finally, we also use the following simple inequality that states that conditioning on a random variable can only increase the mutual information by the entropy of the conditioned variable.

Proposition 2.4.4. *For random variables* A, B *and* C*,* $\mathbb{I}(\mathsf{A}\,;\mathsf{B} \mid \mathsf{C}) \leq \mathbb{I}(\mathsf{A}\,;\mathsf{B}) + \mathbb{H}(\mathsf{C})$.

Proof. By chain rule for mutual information (Fact 2.4.1-(6)), we can write:

$$\begin{aligned}\mathbb{I}(\mathsf{A}\,;\mathsf{B}\mid\mathsf{C}) &= \mathbb{I}(\mathsf{A}\,;\mathsf{B},\mathsf{C}) - \mathbb{I}(\mathsf{A}\,;\mathsf{C}) = \mathbb{I}(\mathsf{A}\,;\mathsf{B}) + \mathbb{I}(\mathsf{A}\,;\mathsf{C}\mid\mathsf{B}) - \mathbb{I}(\mathsf{A}\,;\mathsf{C})\\ &\leq \mathbb{I}(\mathsf{A}\,;\mathsf{B}) + \mathbb{H}(\mathsf{C}\mid\mathsf{B}) \leq \mathbb{I}(\mathsf{A}\,;\mathsf{B}) + \mathbb{H}(\mathsf{C}),\end{aligned}$$

where the first two equalities are by chain rule (Fact 2.4.1-(6)), the second inequality is by definition of mutual information and its positivity (Fact 2.4.1-(2)), and the last one is because conditioning can only reduce the entropy (Fact 2.4.1-(3)). □

Measures of Distance Between Distributions

We shall make use of several measures of distance (or divergence) between distributions in our proofs. We define these measures here and present their main properties that we use in this thesis.

KL-divergence. For two distributions μ and ν, the *Kullback-Leibler divergence* between μ and ν is denoted by $\mathbb{D}(\mu \mid\mid \nu)$ and defined as:

$$\mathbb{D}(\mu \mid\mid \nu) := \mathop{\mathbb{E}}_{a\sim\mu}\left[\log\frac{\Pr_\mu(a)}{\Pr_\nu(a)}\right]. \tag{2.5}$$

We have the following relation between mutual information and KL-divergence.

Fact 2.4.5. *For random variables* $\mathsf{A},\mathsf{B},\mathsf{C}$*,*

$$\mathbb{I}(\mathsf{A}\,;\mathsf{B}\mid\mathsf{C}) = \mathop{\mathbb{E}}_{(b,c)\sim(\mathsf{B},\mathsf{C})}\Big[\mathbb{D}(\mathrm{dist}(\mathsf{A}\mid\mathsf{C}=c) \mid\mid \mathrm{dist}(\mathsf{A}\mid\mathsf{B}=b,\mathsf{C}=c))\Big].$$

Total variation distance. We denote the total variation distance between two distributions μ and ν on the same support Ω by $\|\mu-\nu\|_{tvd}$, defined as:

$$\|\mu-\nu\|_{tvd} := \max_{\Omega'\subseteq\Omega}\left(\mu(\Omega')-\nu(\Omega')\right) = \frac{1}{2}\cdot\sum_{x\in\Omega}\left|\mu(x)-\nu(x)\right|. \tag{2.6}$$

We use the following basic properties of total variation distance.

Fact 2.4.6. *Suppose μ and ν are two distributions for $\mathcal{E}$, then,* $\Pr_{\mu}(\mathcal{E}) \leq \Pr_{\nu}(\mathcal{E}) + \|\mu - \nu\|_{tvd}$.

The following Pinskers' inequality bounds the total variation distance between two distributions based on their KL-divergence,

Fact 2.4.7 (Pinsker's inequality)**.** *For any distributions μ and ν,* $\|\mu - \nu\|_{tvd} \leq \sqrt{\frac{1}{2} \cdot \mathbb{D}(\mu \mid\mid \nu)}$.

Hellinger distance. For two distributions μ and ν, the *Hellinger distance* between μ and ν is denoted by $\mathrm{h}(\mu, \nu)$ and is defined as:

$$\mathrm{h}(\mu, \nu) := \sqrt{\frac{1}{2} \sum_{x \in \Omega} (\sqrt{\mu(x)} - \sqrt{\nu(x)})^2} = \sqrt{1 - \sum_{x \in \Omega} \sqrt{\mu(x)\nu(x)}}. \tag{2.7}$$

The following inequalities relate Hellinger distance and total variation distance (the proof follows from Cauchy-Schwartz).

Fact 2.4.8. *For any distributions μ and ν,* $\mathrm{h}^2(\mu, \nu) \leq \|\mu - \nu\|_{tvd} \leq \sqrt{2} \cdot \mathrm{h}(\mu, \nu)$.

One can also relate Hellinger distance to the KL-divergence as follows.

Fact 2.4.9 (cf. [198])**.** *For any distributions μ and ν,* $\mathrm{h}^2(\mu, \nu) \leq \frac{1}{2} \cdot \big(\mathbb{D}(\mu \mid\mid \frac{\mu+\nu}{2}) + \mathbb{D}(\nu \mid\mid \frac{\mu+\nu}{2})\big)$.

2.5. Communication Complexity

We consider the standard communication model of Yao [251]. We use π to denote the protocol used by players and use $\mathsf{CC}(\pi)$ to denote the *communication cost* of π defined as the worst-case bit-length of the messages communicated between the players. We further use *internal information cost* [32] for protocols that measures the average amount of information each player learns about the input of the other in the protocol, defined formally as follows. Consider an input distribution dist and a protocol π. Let $(\mathsf{X}, \mathsf{Y}) \sim \text{dist}$ and Π denote the random variables for the inputs and the transcript of the protocol (including the public randomness). The *information cost* of π with respect to dist is $\mathsf{IC}_{\text{dist}}(\pi) := \mathbb{I}_{\text{dist}}(\Pi\,; \mathsf{X} \mid \mathsf{Y}) + \mathbb{I}_{\text{dist}}(\Pi\,; \mathsf{Y} \mid \mathsf{X})$. As one bit of communication can only reveal one bit of information, information cost of a protocol lower bounds its communication cost (see Proposition 2.5.4).

Let $P : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ be a relation. Alice receives an input $X \in \mathcal{X}$ and Bob receives $Y \in \mathcal{Y}$, where (X, Y) are chosen from a joint distribution dist over $\mathcal{X} \times \mathcal{Y}$. We allow players to have access to both public and private randomness. They communicate with each other by exchanging messages such that each message depends only on the private input and random bits of the player sending the message, and the already communicated messages plus the public randomness. At the end, one of the players need to output an answer Z such that $Z \in P(X, Y)$.

We use π to denote a protocol used by the players. We always assume that the protocol π can be randomized (using both public and private randomness), *even against a prior distribution* dist *of inputs*. For any $0 < \delta < 1$, we say π is a δ-error protocol for P over a distribution dist, if the probability that for an input (X, Y), π outputs some Z where $Z \notin P(X, Y)$ is at most δ (the probability is taken over the randomness of *both* the distribution and the protocol).

Definition 2.5.1 (Communication cost)**.** *The* communication cost *of a protocol π on an input distribution* dist*, denoted by $\mathsf{CC}_{\text{dist}}(\pi)$, is the* worst-case *bit-length of the transcript communicated between Alice and Bob in the protocol π, when the inputs are chosen from* dist*.*

Communication complexity of a problem P is defined as the minimum communication cost of a protocol π that solves P on every distribution dist with probability at least $2/3$.

Information complexity. There are several possible definitions of information cost of a communication prtocol that have been considered depending on the application (see, e.g., [30, 32, 64, 67, 75]). We use the notion of *internal information cost* [32] that measures the average amount of information each player learns about the input of the other player by observing the transcript of the protocol.

Definition 2.5.2 (Information cost)**.** *Consider an input distribution* dist *and a protocol π. Let $(\mathsf{X}, \mathsf{Y}) \sim$ dist denote the random variables for the input of Alice and Bob and Π be the the*

random variable for the transcript of the protocol concatenated *with the public randomness* R *used by* π. *The* (internal) information cost *of* π *with respect to* dist *is* $\mathsf{IC}_{\mathrm{dist}}(\pi) := \mathbb{I}_{\mathrm{dist}}(\Pi\,;\mathsf{X} \mid \mathsf{Y}) + \mathbb{I}_{\mathrm{dist}}(\Pi\,;\mathsf{Y} \mid \mathsf{X})$.

One can also define information complexity of a problem P similar to communication complexity with respect to the information cost. However, we avoid presenting this definition formally due to some subtle technical issues that need to be addressed which lead to multiple different but similar-in-spirit definitions. As such, we state our results directly in terms of information cost.

Note that any public coin protocol is a distribution over private coins protocols, run by first using public randomness to sample a random string $\mathsf{R} = R$ and then running the corresponding private coin protocol π^R. We also use Π^R to denote the transcript of the protocol π^R. We have the following standard proposition.

Proposition 2.5.3. *For any distribution* dist *and any protocol* π *with public randomness* $\boldsymbol{R}$*,*

$$\mathsf{IC}_{\mathrm{dist}}(\pi) = \mathbb{I}_{\mathrm{dist}}(\Pi\,;\mathsf{X} \mid \mathsf{Y},\mathsf{R}) + \mathbb{I}_{\mathrm{dist}}(\Pi\,;\mathsf{Y} \mid \mathsf{X},\mathsf{R}) = \mathop{\mathbb{E}}_{R\sim\mathsf{R}}\left[\mathsf{IC}_{\mathrm{dist}}(\pi^R)\right].$$

Proof. By definition of internal information cost,

$$\begin{aligned}
\mathsf{IC}_{\mathrm{dist}}(\pi) &= \mathbb{I}_{\mathrm{dist}}(\Pi\,;\mathsf{X} \mid \mathsf{Y}) + \mathbb{I}_{\mathrm{dist}}(\Pi\,;\mathsf{Y} \mid \mathsf{X}) = \mathbb{I}(\Pi,\mathsf{R}\,;\mathsf{X} \mid \mathsf{Y}) + \mathbb{I}(\Pi,\mathsf{R}\,;\mathsf{Y} \mid \mathsf{X}) \\
&\qquad\qquad \text{(Π denotes the transcript and the public randomness)} \\
&= \mathbb{I}(\mathsf{R}\,;\mathsf{X} \mid \mathsf{Y}) + \mathbb{I}(\Pi\,;\mathsf{X} \mid \mathsf{Y},\mathsf{R}) + \mathbb{I}(\mathsf{R}\,;\mathsf{Y} \mid \mathsf{X}) + \mathbb{I}(\Pi\,;\mathsf{Y} \mid \mathsf{X},\mathsf{R}) \\
&\qquad\qquad \text{(chain rule of mutual information, Fact 2.4.1-(6))} \\
&= \mathbb{I}(\Pi\,;\mathsf{X} \mid \mathsf{Y},\mathsf{R}) + \mathbb{I}(\Pi\,;\mathsf{Y} \mid \mathsf{X},\mathsf{R}) \\
&\qquad\qquad \text{($\mathbb{I}(\mathsf{R}\,;\mathsf{X} \mid \mathsf{Y}) = \mathbb{I}(\mathsf{R}\,;\mathsf{Y} \mid \mathsf{X}) = 0$ since $\mathsf{R} \perp \mathsf{X},\mathsf{Y}$ and Fact 2.4.1-(2))} \\
&= \mathop{\mathbb{E}}_{R\sim\mathsf{R}}\left[\mathbb{I}(\Pi\,;\mathsf{X} \mid \mathsf{Y},\mathsf{R}=R) + \mathbb{I}(\Pi\,;\mathsf{Y} \mid \mathsf{X},\mathsf{R}=R)\right] = \mathop{\mathbb{E}}_{R\sim\mathsf{R}}\left[\mathsf{IC}_{\mathrm{dist}}(\pi^R)\right],
\end{aligned}$$

concluding the proof. □

The following well-known proposition relates communication cost and information cost.

Proposition 2.5.4 (cf. [67])**.** *For any distribution* dist *and any protocol* π*:* $\mathsf{IC}_{\text{dist}}(\pi) \leq \mathsf{CC}_{\text{dist}}(\pi)$.

Proof. Let us assume first that π only uses private randomness and thus Π only contain the transcript. For any $b \in [\mathsf{CC}_{\text{dist}}(\pi)]$, we define Π_b to be the b-th bit of the transcript. We have,

$$\begin{aligned}
\mathsf{IC}_{\text{dist}}(\pi) &= \mathbb{I}(\Pi\,;\mathsf{X} \mid \mathsf{Y}) + \mathbb{I}(\Pi\,;\mathsf{Y} \mid \mathsf{X}) \\
&= \sum_{b=1}^{\mathsf{CC}_{\text{dist}}(\pi)} \mathbb{I}(\Pi_b\,;\mathsf{X} \mid \Pi^{<b}, \mathsf{Y}) + \mathbb{I}(\Pi_b\,;\mathsf{Y} \mid \Pi^{<b}, \mathsf{X}) \\
&\qquad\qquad \text{(by chain rule of mutual information in Fact 2.4.1-(6))} \\
&= \sum_{b=1}^{\mathsf{CC}_{\text{dist}}(\pi)} \mathop{\mathbb{E}}_{\Pi^{<b}} \left[\mathbb{I}(\Pi_b\,;\mathsf{X} \mid \Pi^{<b} = \Pi^{<b}, \mathsf{Y}) + \mathbb{I}(\Pi_b\,;\mathsf{Y} \mid \Pi^{<b} = \Pi^{<b}, \mathsf{X})\right].
\end{aligned}$$

Consider each term in the RHS above. By conditioning on $\Pi^{<b}$, the player that transmit Π_b would become fix. If this player is Alice, then $\mathbb{I}(\Pi_b\,;\mathsf{Y} \mid \Pi^{<b} = \Pi^{<b}, \mathsf{X}) = 0$, because Π_b is only a function of $(\Pi^{<b}, \mathsf{X})$ in this case; similarly, if this player is Bob, then $\mathbb{I}(\Pi_b\,;\mathsf{X} \mid \Pi^{<b} = \Pi^{<b}, \mathsf{Y}) = 0$. Moreover, $\mathbb{I}(\Pi_b\,;\mathsf{X} \mid \Pi^{<b} = \Pi^{<b}, \mathsf{Y}) \leq \mathbb{H}(\Pi_b) \leq 1$ and similarly $\mathbb{I}(\Pi_b\,;\mathsf{Y} \mid \Pi^{<b} = \Pi^{<b}, \mathsf{X}) \leq 1$. As such, the above term can be upper bounded by $\mathsf{CC}_{\text{dist}}(\pi)$. To finalize the proof, note that by Proposition 2.5.3, for any public-coin protocol π, $\mathsf{IC}_{\text{dist}}(\pi) = \mathbb{E}_{R\sim\mathsf{R}}\left[\mathsf{IC}_{\text{dist}}(\pi^R)\right] \leq \mathbb{E}_{R\sim\mathsf{R}}\left[\mathsf{CC}_{\text{dist}}(\pi^R)\right] \leq \mathsf{CC}_{\text{dist}}(\pi)$, where the first inequality is by the first part of the argument. □

Proposition 2.5.4 provides a convinent way of proving communication complexity lower bounds by lower bounding information cost of any protocol.

Rectangle Property of Communication Protocols

We conclude this section by mentioning some basic properties of communication protocols. For any protocol π and inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we define $\Pi_{x,y}$ as the transcript of the

protocol conditioned on the input x to Alice and input y to Bob. Note that for randomized protocols, $\Pi_{x,y}$ is a random variable which we denote by $\mathsf{\Pi}_{x,y}$.

The following is referred to as the rectangle property of deterministic protocols.

Fact 2.5.5 (Rectangle property)**.** *For any deterministic protocol π and inputs $x, x' \in \mathcal{X}$ to Alice and $y, y' \in \mathcal{Y}$ to Bob, if $\Pi_{x,y} = \Pi_{x',y'}$, then $\Pi_{x,y'} = \Pi_{x',y}$.*

Fact 2.5.5 implies that the set of inputs consistent with any transcript $\Pi_{x,y}$ of a deterministic protocol forms a combinatorial rectangle. One can also extend the rectangle property of deterministic protocols to randomized protocols using the following fact.

Fact 2.5.6 (Cut-and-paste property; cf. [30])**.** *For any randomized protocol π and inputs $x, x' \in \mathcal{X}$ to Alice and $y, y' \in \mathcal{Y}$ to Bob,* $\mathrm{h}(\mathsf{\Pi}_{x,y}, \mathsf{\Pi}_{x',y'}) = \mathrm{h}(\mathsf{\Pi}_{x,y'}, \mathsf{\Pi}_{x',y})$.

CHAPTER 3

Sublinear Algorithms for $(\Delta+1)$-Coloring

In this chapter, we study the $(\Delta+1)$ coloring problem in the context of processing massive graphs. The $(\Delta+1)$ coloring problem admits a text-book greedy algorithm that runs in linear time and space. However, when processing massive graphs, even this algorithm can be computationally prohibitive. This is due to various limitations arising in processing massive graphs such as requiring to process the graph with limited space in a streaming fashion or in parallel due to storage constraints, or not having enough time for even reading the entire input. In these scenarios, we are interested in *sublinear algorithms* – these are algorithms that use computational resources that are substantially smaller than the size of the input on which they operate. A natural question is then:

> *Can we design* **sublinear algorithms** *for* $(\Delta+1)$ *coloring problem in modern models of computation for processing massive graphs?*

We answer this fundamental question in the affirmative for several canonical classes of sublinear algorithms including (dynamic) graph streaming algorithms, sublinear time algorithms, and massively parallel computation (MPC) algorithms. We also prove new lower bounds to contrast the complexity of the $(\Delta+1)$ coloring problem in these models with two other closely related problems of maximal independent set and maximal matching (see the family of Locally Checkable Labeling (LCL) problems [213] for more on the connection between these problems)[1].

3.1. Main Results

At the core of our results is a remarkably simple meta-algorithm for the $(\Delta+1)$ coloring problem that is based on a key sparsification result for this problem that we establish. The

[1] Another closely related LCL problem is the $(2\Delta-1)$ *edge* coloring problem. However, as the output in the edge-coloring problem is linear in the input size, one cannot hope to achieve non-trivial algorithms for this problem in models such as streaming or sublinear time algorithms, and hence we ignore this problem in this chapter.

sublinear algorithms are then obtained by efficiently implementing this meta-algorithm in each model separately.

Palette Sparsification and a Meta-Algorithm for $(\Delta+1)$ Coloring

Our approach is to "sparsify" the $(\Delta+1)$ coloring problem to a list-coloring problem with lists/palettes of size $O(\log n)$ for every vertex – in the list-coloring problem, every vertex is given a list of colors and the goal is to find a proper coloring of the graph in which the color of each vertex is chosen from its designated list. We prove the following key structural result.

Theorem 3.1 (Palette Sparsification Theorem). *Let $G(V,E)$ be any n-vertex graph with maximum degree Δ. Suppose for any vertex $v \in V$, we sample $O(\log n)$ colors $L(v)$ from $\{1,\ldots,\Delta+1\}$ independently and uniformly at random. Then with high probability there exists a proper $(\Delta+1)$ coloring of G in which the color for every vertex v is chosen from $L(v)$.*

In Theorem 3.1, as well as throughout the chapter, "with high probability" means with probability $1-1/\text{poly}(n)$ for some large polynomial in n.

Theorem 3.1 can be seen as a *sparsification* result for $(\Delta+1)$ coloring: after sampling $O(\log n)$ colors for each vertex, the total number of edges that share a color in their list is only $O(n \cdot \log^2(n))$ with high probability; at the same time, by computing a list-coloring of G using only these $O(n \cdot \log^2(n))$ edges—which is promised to exist by Theorem 3.1—we obtain a $(\Delta+1)$ coloring of G. As such, Theorem 3.1 provides a way of sparsifying the graph into only $\widetilde{O}(n)$ edges, while still allowing for recovery of a $(\Delta+1)$ coloring of the original graph. This sparsification serves as the central tool in our sublinear algorithms for the $(\Delta+1)$ coloring problem.

We shall remark that, as stated, Theorem 3.1 only promise the existence of a coloring (which can be found in exponential time), but in fact we show that there is an efficient procedure to find the corresponding list-coloring and that this will also be used by our algorithms in

each model.

Sublinear Algorithms for $(\Delta+1)$ Coloring

We use Theorem 3.1 to present new sublinear algorithms for the $(\Delta+1)$ coloring problem which are either the first non-trivial ones or significantly improve upon the state-of-the-art. Table 3.1 contains a summary of our sublinear algorithms and the most closely related work.

Streaming Algorithms. Our Theorem 3.1 can be used to design a *single-pass* semi-streaming algorithm for the $(\Delta+1)$ coloring problem in the most general setting of graph streams, namely, *dynamic* streams that allow both insertions and deletions of edges.

Theorem 3.2. *There exists a randomized single-pass dynamic streaming algorithm for the $(\Delta+1)$ coloring problem using $\widetilde{O}(n)$ space.*

The only previous semi-streaming algorithm for $(\Delta+1)$ coloring was in fact the folklore $O(\log n)$-pass streaming simulation of the standard $O(\log n)$-round parallel (PRAM) algorithms for this problem (see, e.g. the classical results of Alon, Babai, and Itai [9], and Luby [200]). No $o(n^2)$ space single-pass streaming algorithm was known for this problem even in insertion-only streams.

The state-of-affairs for $(\Delta+1)$ coloring was very similar to the case of the closely related maximal matching problem (in dynamic streams): the best known semi-streaming algorithm for this problem on dynamic streams uses $\Theta(\log n)$ passes [5, 192] and it is provably impossible to solve this problem using $o(n^2)$-space in a single pass over a dynamic stream [21] (although this problem is trivial in insertion-only streams). We further prove a lower bound of $\Omega(n^2)$ space on the space complexity of single-pass streaming algorithms for computing a maximal independent set even in insertion-only streams. Considering these lower bounds, one might have guessed a similar lower bound also holds for the $(\Delta+1)$ coloring problem. Theorem 3.2 however is in sharp contrast to these results as it shows that $(\Delta+1)$ coloring indeed admits a single-pass semi-streaming algorithm.

Model	Our Results	Previous Work
Streaming	$\widetilde{O}(n)$ space – single pass	$O(n)$ space – $O(\log n)$ passes (folklore)
Sublinear-Time	$\widetilde{O}(\min\{n\Delta, n^2/\Delta\}) = \widetilde{O}(n^{3/2})$ time	$O(n\Delta)$ (text-book greedy)
MPC	$\widetilde{O}(n)$ memory – $O(1)$ rounds	$O(n)$ memory – $O(\log\log\Delta \cdot \log^* n)$ rounds [226]

Table 3.1: A summary of our sublinear algorithms and the most closely related previous work.

Sublinear Time Algorithms. The text-book greedy algorithm for $(\Delta+1)$ coloring runs in time linear in the input size, i.e., $O(m+n)$ time. Surprisingly, we show that one can improve upon the running time of this age-old algorithm on even mildly dense graphs by using Theorem 3.1. Before we get to state our result let us clarify the model. We assume the standard query model for sublinear time algorithms on general graphs (see, e.g., Goldreich's book [131, Chapter 10]) which allow for three types of queries: (i) what is the degree of a given vertex v, (ii) what is the i-th neighbor of a given vertex v, and (iii) whether a given pair of vertices (u,v) are neighbor to each other or not.

Theorem 3.3. *There exists a randomized $\widetilde{O}(n\sqrt{n})$ time algorithm for the $(\Delta+1)$ coloring problem. Furthermore, any algorithm for this problem requires $\Omega(n\sqrt{n})$ time.*

To our knowledge, this is the first sublinear time algorithm for the $(\Delta+1)$ coloring problem. We also note that an important feature of our algorithm in Theorem 3.3 is that it is *non-adaptive*, i.e., it chooses all the queries to the graph beforehand and thus queries are done in parallel.

In yet another contrast to the $(\Delta+1)$ coloring problem, we show that computing a maximal independent set or a maximal matching requires $\Omega(n^2)$ queries to the graph and hence $\Omega(n^2)$ time.

Massively Parallel Computation (MPC) Algorithms. Another application of Theo-

rem 3.1 is a *constant-round* algorithm for the $(\Delta+1)$ coloring problem in the MPC model, which is a common abstraction of MapReduce-style computation frameworks. In this model, the input is partitioned across several machines with limited memory initially. Computation happens in synchronous rounds wherein the machines can communicate with each other subject to sending and receiving messages that fit their limited local memory .

Theorem 3.4. *There exists a randomized MPC algorithm for the* $(\Delta+1)$ *coloring problem in* $O(1)$ *MPC rounds with* $\widetilde{O}(n)$ *per-machine memory and only* $\widetilde{O}(n)$ *global memory (beside the input).*

Two recent papers considered graph coloring problems in the MPC model. Harvey, Liaw, and Liu [150] designed algorithms that use $n^{1+\Omega(1)}$ memory per machine and find a $(\Delta+o(\Delta))$ coloring of a given graph—an algorithmically (considerably) easier problem than $(\Delta+1)$ coloring in $O(1)$ MPC rounds. Furthermore, Parter [226] designed an MPC algorithm that uses $O(n)$ memory per machine and $\Theta(n^2)$ global memory and finds a $(\Delta+1)$ coloring in $O(\log\log\Delta\cdot\log^*(n))$ rounds[2]. Our Theorem 3.4 improves upon these results significantly: both the number of colors and per machine memory compared to [150], and round-complexity and global memory compared to [226].

Maximal matching and maximal independent set problems have also been studied previously in the MPC model [5, 48, 126, 187, 192]. Currently, the best known algorithms with $\widetilde{O}(n)$ memory per machine require $O(\log\log n)$ rounds for both maximal independent set [126, 187] and maximal matching [48], and similarly for the related problems of $O(1)$-approximating the maximum matching and the minimum vertex cover [13, 15, 48, 99, 126]. Our Theorem 3.4 hence is the first example that gives a constant round MPC algorithm for one of the "classic four local distributed graph problems", i.e., maximal independent set, maximal matching, $(\Delta+1)$ vertex coloring, and $(2\Delta-1)$ edge coloring (see, e.g. [35, 120, 222]) when the memory per machine is as small as $\widetilde{O}(n)$.

[2]The algorithm of Parter [226] is stated in the Congested-Clique model, but using the well-known connections between this model and the MPC model, see, e.g. [46, 126], this algorithm immediately extends to the MPC model.

Before we move on from this section, let us make several remarks about our sublinear algorithms.

Optimality of Our Sublinear Algorithms. Our algorithms obtain essentially optimal bounds in each model considered. Indeed, space-complexity of our streaming algorithm in Theorem 3.2 and round-complexity of our MPC algorithm in Theorem 3.4 are clearly optimal (to within poly-log factors and constant factors, respectively). We further prove that query and time complexity of our sublinear time algorithm in Theorem 3.3 are also optimal up to poly-log factors.

3.1.1. Our Techniques

The main technical ingredient of our contribution is Theorem 3.1. For intuition about this result, consider two extreme cases: when the graph is very dense, say is a clique on $\Delta+1$ vertices, and when the graph is relatively sparse, say every vertex (except one) have degree at most $\Delta/2$. Theorem 3.1 is easy to prove for either case albeit by using entirely different arguments as we sketch below.

For the former case, consider the bipartite graph consisting of vertices in V on one side and set of colors $\{1,\ldots,\Delta+1\}$ on the other side, where each vertex v in the V-side is connected to vertices in $L(v)$ in the color-side. Using standard results from random graph theory, one can argue that this graph with high probability has a perfect matching, thus implying the list-coloring of G (by coloring each vertex by its "matched color"). For the latter case, consider the following simple (distributed-style) greedy algorithm: iteratively sample a color for every vertex from the set $\{1,\ldots,\Delta+1\}$ and assign the color to the vertex if it is not chosen by any of its neighbors so far; remove the colored vertices and repeat the same exact process until the entire graph is colored. It is well-known (and easy to prove) that this algorithm only requires $O(\log n)$ rounds when number of colors is a constant factor larger than the degree. As such, the set of colors sampled in the list $L(v)$ for vertices $v \in V$ is enough to "simulate" this algorithm in this case (note that for the purpose of this simulation, it is crucial the colors are sampled from $\{1,\ldots,\Delta+1\}$ in every iteration).

To prove Theorem 3.1 in general, we need to interpolate between these two extreme cases. To do so, we decomposes the graph into "sparse" and "dense" components. The proof for coloring the sparse components then more or less follows by simulating standard distributed algorithms in [109, 235] as discussed above. The main part, and where we concentrate bulk of our efforts, is to prove the result for dense components. Note that in general, we can always reduce the problem of finding a $(\Delta+1)$ coloring to an instance of the assignment problem on the bipartite graph $V \times \{1,\ldots,\Delta+1\}$ discussed above. The difference is that we need to allow some vertices in $\{1,\ldots,\Delta+1\}$ to be assigned to more than one vertex in V when $|V| > \Delta+1$ (as opposed to the case of cliques above that only required finding a perfect matching). We show that if the original graph is "sufficiently close" to being a clique, then with high probability, such an assignment exists in this bipartite graph and use this to prove the existence of the desired list-coloring of G.

We remark that similar-in-spirit graph decompositions and analyzing sparse and dense parts of the graph separately in the context of $(\Delta+1)$ coloring have been studied previously both in graph theory literature (see, e.g. [206, 207, 208, 231, 232]) and distributed computing (see, e.g. [80, 148, 226]). However, both the particular decomposition we use and more importantly the handling of the dense parts of the decomposition are entirely new to this problem. In particular, while these previous results color the dense parts of the graph using *adaptive* iterative procedures (the so-called "Rödl Nibble" [233] for former results and multi-round distributed algorithms for the latter), our approach based the assignment problem is *nonadaptive* and "one shot" and is inspired by random graph theory ideas.

Theorem 3.1 implies the sublinear algorithms we design in each model with a simple caveat: The list-coloring problem is in general NP-hard and hence using Theorem 3.1 directly does not allow for a polynomial time implementation of our algorithms to find the list-coloring of the sparsified graph. To fix this, we design an algorithm based on the proof of Theorem 3.1 that given the sparsified graph, *and* the decomposition of the original graph used in the proof, can find the desired list-coloring in polynomial time (in fact even linear time in the

size of the sparsified graph).

3.1.2. Related Work

Graph coloring has a rich history in both graph theory and computer science. We refer the interested reader to excellent texts by Molloy and Reed [205] and by Barenboim and Elkin [36] for an extensive background on this problem.

The study of graph coloring as an algorithmic problem dates back to at least half a century ago: Finding the minimum number of colors needed for proper coloring, i.e., the *chromatic number*, is one of Karp's 21 NP-hard problems [180] and various exponential-time algorithms were designed for this problem since then [49, 113, 193]. It turned out it is NP-hard to even approximate the chromatic number to within a factor of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ [115, 257] and a series of approximation algorithms [55, 163, 249] culminated in the currently best ratio of $O(n \cdot \frac{(\log\log n)^2}{\log^3 n})$ [144]. On the other hand, a $(\Delta+1)$ coloring can be found via a simple greedy algorithm and polynomial time algorithms are also known for Δ-coloring [69, 199] and even smaller number of colors down to $\approx \Delta - \sqrt{\Delta}$ in graphs that admit such colorings [110, 208].

Finally, the $(\Delta+1)$ coloring problem has been studied extensively in different models, for instance, in distributed settings, e.g., [9, 34, 38, 80, 122, 148, 200, 235], dynamic graphs, e.g., [33, 39, 57], and local computation algorithms, e.g., [37, 80, 225] (this is by no means a comprehensive list).

3.1.3. Recent Developments

Independently and concurrently to our work, two other papers also considered the vertex coloring problem in settings related to this paper. Firstly, Parter and Su [227], improving upon the previous algorithm of Parter [226], gave $O(\log^*(\Delta))$-round congested-clique and MPC algorithms with $O(n)$ per-machine memory for $(\Delta+1)$ coloring. Moreover, Bera and Ghosh [54] studied graph coloring in the streaming model and gave a single-pass algorithm that for any $\varepsilon \in (0,1)$, outputs a $(1+\varepsilon)\Delta$ coloring of the input graph using $\widetilde{O}(n/\varepsilon)$ space.

Note that for the $(\Delta+1)$ coloring problem, this algorithm requires $\Omega(n\Delta)$ space which is equal to the input size.

Subsequent to our work, Alon and Assadi [8] studied graph theoretic aspects of the palette sparsification theorem. They show that for $(1+\varepsilon)\Delta$ coloring problem, it is sufficient and necessary to sample $O(\sqrt{\log n}/\varepsilon^{1.5})$ colors per vertex. The authors also show that for triangle free graph, it is sufficient and necessary to sample $O(\Delta^{\gamma}+\sqrt{\log n})$ colors per vertex to obtain a proper $O(\frac{\Delta}{\gamma \log \Delta})$-coloring.

Also, Bera, Chakrabarti, and Ghosh [53] extended the previous work of [54] and gave sublinear algorithms for $(\kappa + o(\kappa))$-coloring of any graph with *degeneracy* κ in the models considered in this paper with similar resource requirement as our algorithms. We shall remark that while in every graph $\kappa \leq \Delta$, our bounds and those of [53] are *incomparable* as $\Delta+1$ and $(\kappa+o(\kappa))$ are incomparable (interestingly, and in contrast to our results, the authors of [53] also proved that obtaining $(\kappa+1)$-coloring via sublinear streaming or query algorithms is not possible even though every graph admits a $(\kappa+1)$ coloring).

For streaming algorithms, Assadi, Chen and Sun [16] consider deterministic streaming algorithms for graph coloring. They show that deterministic single-pass semi-streaming streaming algorithms need exponential colors available to each vertex. They also show that there exist semi-streaming algorithms runs in two passes using $O(\Delta^2)$ colors and semi-streaming algorithms runs in $O(\log \Delta)$ passes using $O(\Delta)$ colors. For MPC model, Chang *et al.* [81], among other results, gave an $O(\sqrt{\log \log n})$ round MPC algorithm for this problem on machines with memory $n^{\Omega(1)}$.

Finally, motivated by our impossibility results for computing MIS and maximal matching in sublinear time, Assadi and Solomon [22] studied these problems on graphs with *bounded neighborhood independence* and gave sublinear time algorithms for both problems on such graphs.

3.2. A Sparse-Dense Decomposition

We present our sparse-dense decomposition in this section. Similar decompositions have been studied in graph theory starting from the influential work of [231] (see, e.g., [206, 207, 208, 231, 232]) and more recently in distributed algorithms [80, 148, 226]. Our decomposition can be seen as "best of both worlds" in that it *simultaneously* provides the stronger guarantees of the former line of work (in fact, even more nuanced than those) while also admitting a simple algorithm for its recovery similar to the latter ones (although these work focus on its recovery via a local algorithm while we focus on sublinear algorithms).

Lemma 3.2.1. *Let $\varepsilon < 1/50$ and $G = (V, E)$ be any arbitrary graph. We can decompose the vertices of G into the sets $V_{sparse}, C_1, \ldots, C_k$ with the following properties:*

1. *For every vertex $v \in V_{sparse}$, the total number of edges between the neighbors of v is <u>at most</u> $(1-\varepsilon^2) \cdot \Delta^2/2$; we refer to these vertices as* **sparse** *vertices.*
2. *Each set of vertices C_i, called an* **almost-clique**, *has the following properties:*
   (a) *$(1-\varepsilon)\Delta \leq |C_i| \leq (1+3\varepsilon)\Delta$;*
   (b) *Any vertex $v \in C_i$ has at most $3\varepsilon\Delta$ neighbors outside of C_i;*
   (c) *Any vertex $v \in C_i$ has at most $6\varepsilon\Delta$ non-neighbors inside of C_i.*

We start with some definitions first. We say that a vertex $v \in V$ is **sparse** if there are at most $(1-\varepsilon)\Delta$ neighbors of $u \in N(v)$ such that $|N(u) \cap N(v)| \geq (1-\varepsilon)\Delta$. Note that for any sparse vertex v, the total number of edges between the neighbors of v is at most

$$(1-\varepsilon) \cdot \Delta^2/2 + (\varepsilon\Delta)(1-\varepsilon) \cdot \Delta/2 = (1-\varepsilon^2) \cdot \Delta^2/2,$$

as desired by the guarantee of V_{sparse} in Lemma 3.2.1. Thus, we shall ensure that every vertex in V_{sparse} is a sparse vertex according to the above definition (although we emphasize that *not* all sparse vertices will end up in V_{sparse}).

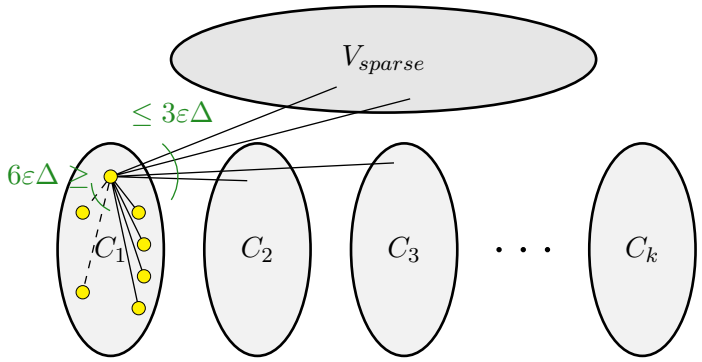


Figure 3.1: Illustration of the sparse-dense-decomposition given by Lemma 3.2.1. Solid lines denote the neighbors of vertices and dashed lines denote the non-neighbors inside the component. Here size of each component is both upper and lower bounded by $(1 \pm \Theta(\varepsilon)) \cdot \Delta$.

We refer to any vertex which is not sparse as a **dense** vertex and let D denote the set of dense vertices. For any dense vertex $v \in D$, we define the following set of other "similar" vertices:

$$S_v := \{u \in V \mid |N(u) \cap N(v)| \geq (1 - 2\varepsilon) \cdot \Delta\}.$$

By the definition of v being dense, we have $|S_v| \geq (1 - \varepsilon)\Delta$.

Consider the following graph H over the same set of vertices V: for any two vertices u and v, there is an edge between u and v if and only if $u \in D$ and $v \in S_u$ or $v \in D$ and $u \in S_v$ (note that an edge in H may not be an edge in G, because vertices in S_v are chosen from *all* the graph not only neighbors of u). Let $C_1, \ldots, C_k$ be the connected components of size more than one in H, and let V_{sparse} be the isolated vertices in H. In the following, we prove such a decomposition satisfies the conditions in Lemma 3.2.1.

We start with the following two helper claims.

Claim 3.2.2. *If two vertices $u, v \in D$ have a common neighbor in H, then (u, v) belongs to H.*

Proof. Let w be a common neighbor of u and v. By the definition of H, $w \in S_u$ and $w \in S_v$. Now, since $w \in S_u$, we have $|N(w) \cap N(u)| \geq (1 - 2\varepsilon)\Delta$. On the other hand, since u is a

dense vertex, it has at most $\varepsilon\Delta$ neighbors that have less than $(1-\varepsilon)\Delta$ common neighbors with u. This means w and u have at least $(1-3\varepsilon)\Delta$ common neighbors C_u such that for each $z \in C_u$, $|N(z) \cap N(u)| \geq (1-\varepsilon)\Delta$. By symmetry, we can also argue the same thing for a set C_v of the vertex v.

Since $\varepsilon < 1/50$, we have that $C_u \cap C_v$ is non-empty as they are both subsets of size at least $(1-3\varepsilon)\Delta$ from $N(w)$ that has size at most Δ. Let z be any vertex in this intersection. Thus,

$$|N(u) \cap N(v)| \geq |N(u) \cap N(z)| - |N(v) \setminus N(z)| \geq (1-\varepsilon)\cdot\Delta - \varepsilon\Delta = (1-2\varepsilon)\cdot\Delta.$$

Thus, u belongs to S_v and v to S_u and so (u,v) is an edge in H by definition. □

Claim 3.2.3. *For any dense vertex $v \in D$ and any $u \in V$ in the connected component of v in H, we have $|N(v) \cap N(u)| \geq (1-4\varepsilon)\Delta$.*

Proof. We first consider the case when u is also a dense vertex, i.e., $u \in D$. We prove that in this case, the edge (u,v) belongs to H, which is sufficient to prove the lemma as that means we have $|N(u) \cap N(v)| \geq (1-2\varepsilon)\Delta$, by the definition of edges of H.

Suppose towards a contradiction that (u,v) is not in H and let $(u = w_1, w_2, \ldots, w_\ell = v)$ be the shortest path between u and v, which by our assumption has length at least three. By the definition of H, for any edge, at least one endpoint belongs to D. Thus, there are vertices w_i and w_{i+2} in this path for some $i \in [\ell]$ such that both vertices belong to D. Since w_i and w_{i+2} are both in D and have a common neighbor in H (namely, w_{i+1}), we can apply Claim 3.2.2 to get that the edge (w_i, w_{i+2}) belongs to H also. But then we can shortcut the previous path by going from w_i to w_{i+2} directly, contradicting that the original path was a shortest path. Thus, (u,v) belongs to H.

Suppose now that u is not a dense vertex. Each edge in H has at least one dense endpoint and u belongs to the connected component of v in H. Thus, there is a vertex w in this connected

component which is both dense and also is a neighbor to u. Thus, $|N(u) \cap N(w)| \geq (1-2\varepsilon)\Delta$ by the definition of edges of H. At the same time, by the previous part, since w is a dense vertex, we have $|N(v) \cap N(w)| \geq (1-2\varepsilon)\Delta$. Putting these two together, plus the fact that $N(w)$ has size at most Δ implies that $|N(u) \cap N(v)| \geq (1-4\varepsilon)\Delta$, concluding the proof. □

Proof of Lemma 3.2.1. For any vertex $v \in D$, we have $|S_v| \geq (1-\varepsilon)\Delta > 1$. Since v is neighbor to S_v in H, no singleton connected component of H can be a dense vertex. Thus, all those components are sparse vertices and can safely be added to V_{sparse}, while satisfying the condition (1) of Lemma 3.2.1.

Consider a non-singleton connected component C_i in H. Each edge in H has a dense endpoint and thus there is a dense vertex v in C_i. As v is neighbor to S_v in H, we have $|C_i| \geq |S_v| \geq (1-\varepsilon)\Delta$. This proves the lower bound in the condition $(2.a)$. We now prove the upper bound.

Define $U_v := C_i \setminus N(v)$ to be the vertices in C_i that are not from $N(v)$. Since v is a dense vertex, it has at least $(1-\varepsilon)\Delta$ neighbors w such that $|N(w) \cap N(v)| \geq (1-\varepsilon)\Delta$. Any such vertex w can have at most $\varepsilon\Delta$ neighbors outside $N(v)$ and in particular to U_v, so these vertices, even if we have Δ of them, can provide at most $\varepsilon\Delta^2$ edges in total to U_v. The remaining $\leq \varepsilon\Delta$ neighbors of v in $N(v)$ can provide another $\varepsilon\Delta^2$ edges to U_v. On the other hand, by Claim 3.2.3, any vertex $u \in U_v \subseteq C_i$ has $\geq (1-4\varepsilon)\Delta$ edges to $N(v)$, and thus "consumes" $\geq (1-4\varepsilon)\Delta$ edges going from $N(v)$ to U_v. Hence, $|U_v| \leq 2\varepsilon\Delta^2/(1-4\varepsilon)\Delta < 3\varepsilon\Delta$ for $\varepsilon < 1/50$. Thus, $|C_i| \leq |N(v)| + |U_v| \leq (1+3\varepsilon)\Delta$ as desired by the condition $(2.a)$. We can now prove the remaining conditions easily.

We first prove condition $(2.b)$. This time, fix a dense vertex $u \in C_i$ and consider any $v \in C_i$. If v is also dense, since u and v have a common neighbor by Claim 3.2.3, we can apply Claim 3.2.2 to have that $|N(u) \cap N(v)| \geq (1-2\varepsilon)\Delta$ by the definition of H. Since all of S_u of size $\geq (1-\varepsilon)\Delta$ belongs to C_i, this means that v can have at most $2\varepsilon\Delta$ neighbors outside $N(u)$ plus an additional $\varepsilon\Delta$ out of $S_u \subseteq C_i$. Thus, u has $\leq 3\varepsilon\Delta$ neighbors outside

of C_i. On the other hand, if v is not dense, then there should be a dense vertex $w \in C_i$ such that $v \in S_w$. This forces $|N(w) \cap N(v)| \geq (1-2\varepsilon)\Delta$. We can now do the same argument as above by replacing the role of u with w, and get that v can only have $\leq 3\varepsilon\Delta$ neighbors out of S_w and thus out of C_i as well.

Finally, to prove condition $(2.c)$, note that for any vertex $v \in C_i$ there is another dense vertex $u \in C_i$ such that $|N(v) \cap N(u)| \geq (1-2\varepsilon)\Delta$. Moreover, we already proved that C_i can have at most $3\varepsilon\Delta$ vertices from outside of $N(u)$. Putting these two together, implies that v can have at most $2\varepsilon\Delta$ non-edges to $N(u)$ and another $3\varepsilon\Delta$ non-edges to $C_i \setminus N(u)$, thus v has at most $5\varepsilon\Delta$ non-neighbors in C_i. This concludes the proof of Lemma 3.2.1. □

3.3. The Palette Sparsification Theorem

We prove our Theorem 3.1 (restated formally below) in this section.

Theorem 3.5 (Palette Sparsification Theorem). *Let $G = (V, E)$ be any graph with n vertices and maximum degree Δ. Suppose for every vertex $v \in V$, we independently pick a set $L(v)$ of colors of size $\Theta(\log n)$ uniformly at random from $[\Delta + 1]$. Then, with high probability, there exists a proper coloring $\chi : V \to [\Delta + 1]$ of G such that for all vertices $v \in V$, $\chi(v) \in L(v)$.*

Recall the decomposition of graph into sparse and dense parts introduced in Section 3.2. We prove Theorem 3.5 in two parts, one for coloring the sparse vertices and one for dense ones. The first part for coloring sparse vertices is relatively easy. Roughly speaking, sparse vertices "can be made low-degree" by coloring a small fraction of the graph randomly (to our knowledge, this observation was first made in [206]) and for coloring low-degree graphs we already saw (a sketch of) an easy proof in Section 3.1.1. Indeed this part of the argument is mostly a "simulation" of the distributed algorithms of [80, 109, 148] using the sampled colors.

In the second part, we extend the argument to dense vertices. For these vertices, it will be evident that the "per vertex" approach in the first part would not work and thus we work with

a more "global" argument, one that takes into account all vertices of a single almost-clique (in the decomposition) at the same time. We start this part with an argument that allows us to color a sufficiently large fraction of vertices in each almost-clique at a rate of two vertices per color (assuming the number of non-edges in the almost-clique is not too small). This approach in turn "saves" us extra colors for coloring the remainder of the almost-clique which brings us to the main part of the argument. Using the assignment formulation of the coloring problem discussed already in Section 3.1.1, we reduce this problem to proving existence of a perfect matching in certain families of "random graphs" – these are random subgraphs of a complete bipartite graph minus an *adversarially* chosen "sparser" subgraph. Finally, we prove existence of this perfect matching using ideas inspired by proofs for existence of perfect matchings in random graphs (see, e.g. [61]) combined with some combinatorial arguments.

3.3.1. Notation and Parameters

We start with setting up our notation and parameters. For the ease of reference, let us collect all our main parameters here (note that both ε, α are *constants*)[3].

$$\varepsilon := 10^{-4} \quad , \quad \alpha := e^5 \quad , \quad \ell := \left(10\alpha/\varepsilon^2\right) \cdot \ln n \quad , \quad p := \frac{\ell}{2 \cdot (\Delta + 1)}. \tag{3.1}$$

For concreteness, in Theorem 3.5, we assume that every vertex v samples a list of size $\ell = \Theta(\log n)$. However, in some places in the argument, it would be more convenient to assume each vertex v *independently* samples each color c w.p. p in $L(v)$: This is without loss of generality as by the choice of p, with high probability the total number of sampled colors is $< \ell$ (when this is not the case, we can simply charge the probability of this event to the probability of error in Theorem 3.5). As such, throughout the argument we freely interchange between these two notions of sampling.

In the proof, we fix a dense-sparse-decomposition $V_{sparse}, C_1, \ldots, C_k$ of G described in Sec-

[3] In the interest of simplifying the exposition of the proof, we made no attempt in optimizing the constants and instead chose the most straightforward values in every step. Our result continues to hold with much smaller constants. However, we also do not see a way to reduce the leading constant of $O(\log n)$ sampled colors to below 1000.

tion 3.2 for parameter ε of Eq (3.1). Recall that vertices in V_{sparse} are called *sparse* vertices (satisfying the first condition of Lemma 3.2.1) and each C_i is referred to as an *almost-clique* (satisfying the second condition in Lemma 3.2.1).

We define a *partial coloring* as $\chi : V \to [\Delta+1] \cup \{\perp\}$: for every $v \in V$, $\chi(v) \in [\Delta+1]$ means that v is colored by $\chi(v)$ and $\chi(v) = \perp$ means v is not colored yet; moreover, for any edge (u, v) either at least one of $\chi(u) = \perp$ or $\chi(v) = \perp$ holds, otherwise $\chi(u) \neq \chi(v)$. It is clear that any partial coloring in which $\chi(v) \neq \perp$ for every v is a proper coloring. For sampled-lists L of vertices, we say that χ is *L-compatible* iff $\chi(v) \in L(v) \cup \{\perp\}$ for every v.

Under a partial coloring χ, we set $\Psi_\chi(v) \subseteq [\Delta+1]$ to denote the set of colors in $[\Delta+1]$ that are *available* to v, i.e., $\Psi_\chi(v) := \{c \in [\Delta+1] \mid \forall u \in N(v) : \chi(u) \neq c\}$. Similarly, define $\mathsf{deg}_\chi(v)$ as the degree of v to vertices $u \in N(v)$ with $\chi(u) = \perp$.

Finally, we shall assume throughout the proof that $\Delta \geq \beta \cdot \log n$ for some sufficiently large constant $\beta > 0$ (this is needed to make some of the concentration bounds work). This assumption is without loss of generality as otherwise sampling $\Delta + 1 = O(\log n)$ colors would trivially lead to the desired list-coloring.

3.3.2. The Setup

The first (and the easy) part of the argument is to color sparse vertices, ignoring entirely all the dense vertices. This will be done using the following lemma.

Lemma 3.3.1. *Suppose for every vertex $v \in V_{sparse}$, we sample a set $L(v)$ of $\ell = \Theta(\varepsilon^{-2} \cdot \log n)$ colors independently and uniformly at random from $[\Delta+1]$. Then, with high probability, the induced subgraph $G[V_{sparse}]$ can be properly colored from the sampled lists $L(v)$ for $v \in V_{sparse}$.*

Equipped with this lemma, we can then color all vertices in $V_\star^{\text{sparse}}$ in the decomposition using the sampled lists in the palette sparsification theorem. For the remainder of the proof, we *fix* this coloring and condition on the event of Lemma 3.3.1 (which happens w.h.p.).

The heart of the argument is to color almost-cliques, which is done using the following lemma (recall that $\mathsf{outdeg}(v)$ for v in an almost-clique C is number of neighbors of v outside C).

Lemma 3.3.2. *Let C be an almost-clique in G. Suppose for every $v \in C$, we adversarially pick a set $B(v)$ of size at most $\mathsf{outdeg}(v)$ colors from $[\Delta+1]$ (referred to as* blocked *colors for v). Now, if for every vertex $v \in V$, we sample a set $L(v)$ of $\ell = \Theta(\varepsilon^{-2} \cdot \log n)$ colors independently from $[\Delta+1]$, then, with high probability, $G[C]$ can be properly colored from the lists $L(v) \setminus B(v)$ for $v \in C$.*

Let us interpret Lemma 3.3.2 as follows. Suppose an adversary colors the graph $G \setminus C$ for some almost-clique C. For each vertex v, let $B(v)$ denote the colors used by the adversary to color neighbors of v outside the almost-clique. Then, Lemma 3.3.2 states that even in this case, the randomness of sampled lists L is sufficient for obtaining a proper coloring of C w.h.p.

We shall note that while Lemma 3.3.1 works for every choice of $\varepsilon \in (0,1)$, for Lemma 3.3.2, we need ε to be sufficiently small (and we state the necessary conditions on ε throughout the proof explicitly) – by taking ε to be the largest value that makes Lemma 3.3.2 works (which we show is a *constant* $> 10^{-4}$), we would be able to combine these two lemmas and prove Theorem 3.5 as follows.

Proof of Theorem 3.5. We first color all sparse vertices using Lemma 3.3.1. Then, we simply go over the almost-cliques one by one and color each almost-clique C using Lemma 3.3.2 by fixing the coloring of so-far-colored vertices in the blocked-lists (even assumed adversarially). The fact that Lemma 3.3.2 holds even against adversarial coloring of the rest of the graph, allows us to color C w.h.p. We iterate like this until we find a proper coloring of G. Taking a union bound on the events of these lemmas concludes the proof.

□

In the following two sections, we give a proof of each of these lemmas separately.

3.3.3. Warm-Up: Coloring Sparse Vertices (Lemma 3.3.1)

We now prove Lemma 3.3.1 (restated below).

Lemma (Restatement of Lemma 3.3.1)**.** *Suppose for every vertex $v \in V_{sparse}$, we sample a set $L(v)$ of $\ell = \Theta(\varepsilon^{-2} \cdot \log n)$ colors independently and uniformly at random from $[\Delta + 1]$. Then, with high probability, the induced subgraph $G[V_{sparse}]$ can be properly colored from the sampled lists $L(v)$ for $v \in V_{sparse}$.*

Proof of this lemma follows the familiar approach of creating "excess" colors for sparse-vertices, hence effectively turning the problem into a one on sufficiently-low-degree graphs (see, e.g. [80, 109, 148, 206] for different variants of this strategy). This is done by picking a random color for each vertex from $L(v)$ (which in turn would be a random color from $[\Delta+1]$) and coloring any vertex that has no neighbor that sampled the same color. As each sparse vertex has $\Omega(\varepsilon^2 \cdot \Delta^2)$ *non-edges* in its neighborhood, we would expect some $\Omega(\varepsilon^2 \cdot \Delta)$ non-edges to sample the same exact color on both endpoints; additionally, we expect each vertex to retain its color with some constant probability. Hence (ignoring dependency issues for the moment) we should also expect $\Omega(\varepsilon^2 \cdot \Delta)$ colors to appear *twice* in the neighborhood of v; this is enough to argue that after this step, every remaining vertex has $\Omega(\varepsilon^2 \cdot \Delta)$ extra colors compared to its remaining degree.

To conclude, we prove that sampled lists are enough to color the remaining low-degree vertices. This is simply because no matter how the rest of the graph is colored, for a vertex of degree $(1-\Omega(\varepsilon^2)) \cdot \Delta$, we have $\Omega(\varepsilon^2 \cdot \Delta)$ colors that sampling even one in $L(v)$, would allow us to color this vertex; but, this happens with high probability once we sample $\Theta(\varepsilon^{-2} \cdot \log n)$ colors per vertex.

Let us now formalize this intuition in the following two parts.

Creating Excess Colors

Consider the following process for coloring sparse vertices:

OneShotColoring.

Input: Graph G with lists L. **Output:** An L-compatible partial coloring χ_1.

(i) Sample a color $c(v)$ uniformly at random from $L(v)$ for every $v \in V$.

(ii) Let $\chi_1(v) = c(v)$ if $c(v) \neq c(u)$ for every $u \in N(v)$, and otherwise $\chi_1(v) = \perp$.

Recall that $\Psi_{\chi_1}(v)$ denotes the set of available colors to v under the partial coloring χ_1 and $\mathsf{deg}_{\chi_1}(v)$ denotes the number of uncolored neighbors of v. We have,

Lemma 3.3.3. *W.h.p., for every vertex v with $\chi_1(v) = \perp$, $|\Psi_{\chi_1}(v)| \geq \mathsf{deg}_{\chi_1}(v) + (1/\alpha) \cdot \varepsilon^2 \cdot \Delta$.*

Proof. Fix any vertex $v \in V_{sparse}$. By Lemma 3.2.1, there are at least $t := \varepsilon^2 \cdot \binom{\Delta}{2}$ non-edges in the neighborhood of v[4]. Let $f_1, \ldots, f_t$ denote these non-edges. Let us further define the following random variable:

- X: number of colors in $[\Delta + 1]$ that are sampled by *at least* two neighbors of v and are additionally retained by *all* these neighbors.

Since any color counted in X is used more than once to color a neighbor of v, we have,

$$|\Psi_{\chi_1}(v)| \geq \mathsf{deg}_{\chi_1}(v) + X. \tag{3.2}$$

We now lower bound the expectation of X and further prove it is concentrated.

Claim 3.3.4. $\mathbb{E}[X] \geq e^{-4} \cdot \varepsilon^2 \Delta$.

Proof. Let us define X' as the number of colors that are sampled by the endpoints of exactly one of f_i's and are retained by both endpoints. Clearly, $X \geq X'$. We thus can lower bound X' instead. For every non-edge $f_i := (u_i, w_i)$, define the indicator random variable X'_i where $X'_i = 1$ iff $c(u_i) = c(w_i)$ and for all $z \in N(v) \cup N(u_i) \cup N(w_i) \setminus \{u_i, w_i\}$, $c(z) \neq c(u_i)$;

[4] If $|N(v)| < \Delta$, we can imagine there are $\Delta - N(v)$ dummy vertices that only connects to v

otherwise $X'_i = 0$. By definition, $X' = \sum_{i=1}^{t} X'_i$. We have,

$$\begin{aligned}\Pr\left[X'_i = 1\right] &= \Pr\left[c(u_i) = c(v_i) \wedge \forall z \in N(u_i) \cup N(v_i) \cup N(v) \setminus \{u_i, v_i\} : c(w) \neq c(u_i)\right] \\ &\geq \frac{1}{\Delta+1} \cdot \left(1 - \frac{1}{\Delta+1}\right)^{\Delta+(\Delta-1)+(\Delta-2)} \\ &\text{(the color of each vertex is chosen uniformly at random from } [\Delta+1] \text{ by randomness of } L(v)) \\ &\geq \frac{1}{\Delta+1} \cdot \exp\left(-\frac{3.003\Delta}{\Delta+1}\right) \geq \frac{e^{-3.003}}{\Delta+1}. \\ &\qquad (1 - x \geq e^{-1.001x} \text{ for sufficiently small } x \in (0,1) \text{ and since } \Delta \text{ is } \omega(1))\end{aligned}$$

By linearity of expectation, $\mathbb{E}[X] \geq t \cdot \frac{e^{-3.003}}{\Delta} \geq e^{-4} \cdot \varepsilon^2 \Delta$ as $t = \varepsilon^2 \cdot \binom{\Delta}{2}$.

□

Let us now prove X is concentrated. The proof uses Talagrand's inequality (Proposition 2.2.4) although with an interesting twist (this part is standard; see, e.g. [205, Chapter 10]). We start by defining the following two additional variables:

- A: number of colors in $[\Delta+1]$ that are sampled by *at least* two neighbors of v.
- D: number of colors in $[\Delta+1]$ that are sampled by *at least* two neighbors of v but are *not* retained by *at least one* of them.

Firstly, it is clear that $X = A - D$. Also notice that both A and D are functions of *independent* random variables that define the choices of random colors $c(v)$ for every $v \in V$. The problem with applying Talagrand's inequality to X directly is that it is not easily certifiable from these variables (recall the definition from Section 2.2); however, both A and D are $\Theta(1)$-certifiable (for A point to two neighbors of v that sampled the color; for D additionally point to one of the neighbors of this pair that also sampled the color, hence not allowing one of them to retain it). They are also both $\Theta(1)$-Lipschitz: changing choice of one color for a vertex can only affect the two colors involved (the original one and the changed one). As such, we can apply Talagrand's inequality (Proposition 2.2.4) to obtain

that (we only write the bound for A; the same exact argument works also for D):

$$\Pr\Big(|A - \mathbb{E}[A]| \geq \mathbb{E}[X]/10\Big) \leq \exp\Big(-\Theta(1) \cdot \frac{(\mathbb{E}[X] - \Theta(1)\sqrt{\Delta})^2}{\Delta}\Big) \quad (A \leq \Delta/2 \text{ always})$$

$$\Pr\Big(|A - \mathbb{E}[A]| \geq \mathbb{E}[X]/10\Big) \leq \exp\Big(-\Theta(\varepsilon^4) \cdot \Delta\Big) \ll n^{-10}.$$

(by Claim 3.3.4 on expected value of X and since we can assume Δ to be $\gg \varepsilon^{-4} \cdot \ln n$)

As such, we obtain that w.h.p. both A and D are concentrated and thus also w.h.p.,

$$X = A - D \geq \mathbb{E}[A] - \mathbb{E}[X]/10 - (D + \mathbb{E}[X]/10) = \mathbb{E}[X] - \mathbb{E}[X]/5 = (4/5) \cdot \mathbb{E}[X].$$

As $\alpha = e^5$ in Eq (3.1), we are done by Claim 3.3.4 and a union bound.

□

Coloring the Remaining Sparse Vertices

We now color the remaining vertices in V_{sparse}, i.e., vertices v with $\chi_1(v) = \perp$. This is done via the following procedure. In the following, let $L'(v)$ denote the list $L(v)$ minus the sampled color $c(v)$ that was used in OneShotColoring for vertex v.

GreedyColoring.

Input: Graph G with coloring χ_1 and lists L'. **Output:** An L'-compatible partial coloring χ_2.

1. Let $\chi_2 \leftarrow \chi_1$ initially and assume an arbitrary ordering of colors in $L'(v)$ for any v.
2. For $i = 2$ to ℓ *iterations*:
   (a) For every vertex $v \in V_{sparse}$ with $\chi_2(v) = \perp$, let $c_i(v)$ be the i-th color in $L'(v)$.
   (b) If $c_i(v) \in \Psi_{\chi_2}(v)$ and no vertex u in $N(v)$ has also $c_i(u) = c_i(v)$, let $\chi_2(v) = c_i(v)$.

We argue that after running GreedyColoring w.h.p. all vertices in V_{sparse} are assigned a color, i.e., at the end, for every $v \in V_{sparse}$, $\chi_2(v) \in [\Delta + 1]$.

Lemma 3.3.5. *W.h.p., after running* GreedyColor*, for every vertex* $v \in V_{sparse}$*,* $\chi_2(v) \in [\Delta+1]$*, i.e., v is assigned a valid color.*

Proof. Fix any vertex $v \in V_{sparse}$ with $\chi_1(v) = \perp$, i.e., a one not colored by OneShotColoring. We argue that with high probability, in one of the $\ell - 1$ iterations of GreedyColoring, v is assigned a color $c(v)$ which additionally results in $\chi_2(v) = c(v)$; as once a vertex is colored we never change its color, this plus a union bound on all vertices finalizes the proof.

For every iteration $i \in \{2, \ldots, \ell\}$, the color $c_i(v)$ considered by GreedyColoring is chosen uniformly at random from $[\Delta+1]$ minus the (at most) $i-1$ colors of v in $L'(v)$ that are seen already (one in OneShotColoring and $i-2$ in GreedyColoring). Moreover, v starts GreedyColoring with colors in $\Psi_1(v)$ available to it (to sample from) and throughout this process at most $\mathsf{deg}_1(v)$ of these colors may become unavailable due to their assignment to neighbors of v. As such,

$$\begin{aligned}
\Pr\left(\chi_2(v) \text{ is set to } c_i(v) \mid c_1(v), \ldots, c_{i-1}(v)\right) &\geq \frac{|\Psi_{\chi_1}(v)| - \mathsf{deg}_1(v) - (i-1)}{\Delta + 1 - (i-1)} \\
&\geq \frac{(1/\alpha) \cdot \varepsilon^2 \cdot \Delta - (i-1)}{\Delta} && \text{(by Lemma 3.3.3)} \\
&\geq (1/2\alpha) \cdot \varepsilon^2. && \text{(as } \Delta \gg i\text{)}
\end{aligned}$$

As such, the probability that v is never colored by GreedyColoring is

$$\begin{aligned}
\Pr\left(\chi_2(v) = \perp \text{ after the last iteration}\right) &\leq \left(1 - (1/2\alpha) \cdot \varepsilon^2\right)^{\ell-1} \\
&\leq \exp\left(((1/2\alpha) \cdot \varepsilon^2) \cdot ((10\alpha/\varepsilon^2) \cdot \ln n - 1)\right) \ll n^{-4}. \\
&\qquad \text{(by the choice of } \ell \text{ in Eq (3.1))}
\end{aligned}$$

Taking a union bound over at most n vertices concludes the proof.

□

Lemma 3.3.1 now follows immediately from Lemmas 3.3.3 and 3.3.5.

Remark 3.3.6. We remark that if our goal was to prove Theorem 3.5 for $(\Delta+o(\Delta))$ coloring as opposed to $(\Delta+1)$ coloring, we would only need Lemma 3.3.5 because in that case, every vertex already has a sufficiently larger number of available colors than its degree. Hence, the main challenge in proving Theorem 3.5 is to obtain the result for $(\Delta+1)$ coloring.

3.3.4. Main Part: Coloring Dense Vertices (Lemma 3.3.2)

We now prove Lemma 3.3.2 (restated below).

Lemma (Restatement of Lemma 3.3.2)**.** *Let C be an almost-clique in G. Suppose for every $v \in C$, we* adversarially *pick a set $B(v)$ of size at most* $\mathsf{outdeg}(v)$ *colors from $[\Delta+1]$ (referred to as* blocked *colors for v). Now, if for every vertex $v \in V$, we sample a set $L(v)$ of ℓ colors independently from $[\Delta+1]$, then, with high probability, $G[C]$ can be properly colored from the lists $L(v) \setminus B(v)$ for $v \in C$.*

Before we get to the proof, let us emphasize that in the setting of Lemma 3.3.2, if one set $L(v) = [\Delta+1]$ (instead of the sampled list), it is easy to see that C can be colored from the lists – this is because any partial coloring of a graph G can be extended to a $(\Delta+1)$ coloring of G. We will now show that even when lists $L(v)$ are of much smaller size and are chosen randomly, we still obtain such a coloring w.h.p.

As stated earlier, the first step in proving Lemma 3.3.2 is to "pair up vertices" inside C that can both be colored with the same color from $L(v)$. On the surface, this is similar to Lemma 3.3.3 that was used for coloring sparse vertices. However, both the purpose of this step and the approach in proving it are quite different from Lemma 3.3.3. Roughly speaking, our goal here is to simply color enough number of vertices from the almost-clique C (at a rate of two vertex per color) so that number of uncolored vertices in C becomes sufficiently smaller than number of colors *not* assigned to vertices in C (recall that originally, $|C|$ can be as large as $(1+\Theta(\varepsilon))\Delta$). Once this happens, we can focus on finding a *matching* of colors to vertices (at a rate of one color per vertex).

The final step of the argument is to show that there exists a matching of colors to uncolored

vertices of C at this point that is compatible with lists L. Recall the assignment formulation we discussed in Section 3.1.1: we will place the remaining uncolored vertices of C in the left-side of a bipartite graph (called the *vertex-side*) and the colors not assigned to C so far on the right-side (the *color-side*); then we connect every vertex v in the vertex-side to every color c in the color-side iff $c \in L(v) \setminus B(v)$. We will prove that this graph has a left-saturating matching (a one that matches every vertex on the left), implying that we can find a unique color for every vertex remaining in C. We shall emphasize that in the assignment formulation in general one may need to assign a color to multiple vertices; however, by considering almost-cliques one at a time and performing the first step of coloring of this almost-clique described above, we can focus on the "easier to handle" case when each color needs to be assigned to exactly one vertex, i.e., a *matching* problem.

Before we move on, let us make a remark on our notation in this part.

Remark 3.3.7. Recall that in Section 3.3.1, we argued that one can alternatively consider the process of sampling the list $L(v)$ of each vertex $v \in V$ as sampling each color $c \in [\Delta+1]$ w.p. p. In this process, we can consider two separate lists $L_1(v)$ and $L_2(v)$ for every vertex v, where we sample each color independently w.p. $p/2$ in each one (this way, the probability that a color is sampled overall is $2p/2 - (p/2)^2 < p$). We use the lists $L_1(v)$ in the first part of the argument and $L_2(v)$ in the second part to ensure the necessary independence between the two parts.

Reducing Size of the Almost-Clique

For a partial coloring χ and almost-clique C, define:

- $V_\chi^{\mathsf{uncolored}}(C)$: uncolored vertices in C, i.e., $V_\chi^{\mathsf{uncolored}}(C) := \{v \in C \mid \chi(v) = \perp\}$;
- $\Psi_\chi(C)$: colors not used in C, i.e., $\Psi_\chi(C) := \{c \in [\Delta+1] \mid \nexists v \in C, \chi(v) = c\}$;
- $\mathrm{NE}(C)$: set of non-edges inside C, i.e., $\mathrm{NE}(C) := \{(u,v) \mid u,v \in C \wedge (u,v) \notin E\}$.

We prove the following lemma in this part (recall the definition of $L_1(v)$ in Remark 3.3.7).

Lemma 3.3.8. *Consider the setting of Lemma 3.3.2. W.h.p., there exists a partial coloring χ_3 s.t:*

(i) for every $v \in C$, either $\chi_3(v) \in L_1(v) \setminus B(v)$ or $\chi_3(v) = \perp$;

(ii) $|\Psi_{\chi_3}(C)| = (\Delta+1) - \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor$;

(iii) $\left|V_{\chi_3}^{\mathsf{uncolored}}(C)\right| = |V(C)| - 2 \cdot \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor$.

(We emphasize that the randomness in this lemma is only over the lists L_1 and not *the entire L.)*

Lemma 3.3.8 allows us to partially color an almost-clique in a way that number of colors used is *half* the number of colored vertices and both are proportional to number of non-edges in C. This allows us to "save" extra colors. As such, the "further" C is from a $(\Delta+1)$ clique (hence having a larger number of non-edges), we will also have "more room" in terms of available colors in the next step; this balancing is crucial for the next step of our proof to work.

For the purpose of proving Lemma 3.3.8, we can focus on $|\text{NE}(C)| \geq 200\varepsilon\Delta$; otherwise, there is nothing to do as we can color everything in χ_3 by $\perp$ and satisfy the lemma trivially. Hence, throughout this proof, we may and will assume that $|\text{NE}(C)| \geq 200\varepsilon\Delta$ whenever needed.

Let us further make the following key definition (see Figure 3.2 for an example).

Definition 3.3.9 (Colorful Non-Edge Matching)**.** *A* matching $\overline{M}$ *of* non-edges *in* $\text{NE}(C)$ *is called a* colorful (non-edge) matching *iff:*

(i) For any $(u,v) \in \overline{M}$ there exists a color $c_{u,v} \in (L_1(u) \setminus B(u)) \cap (L_1(v) \setminus B(v))$.

(ii) For any pairs of edge $(u_i, v_i), (u_j, v_j) \in \overline{M}$, $c_{u_i,v_i} \neq c_{u_j,v_j}$.

Suppose we find a colorful matching $\overline{M}$ in C. For any non-edge $(u_i, v_i) \in \overline{M}$, we can color

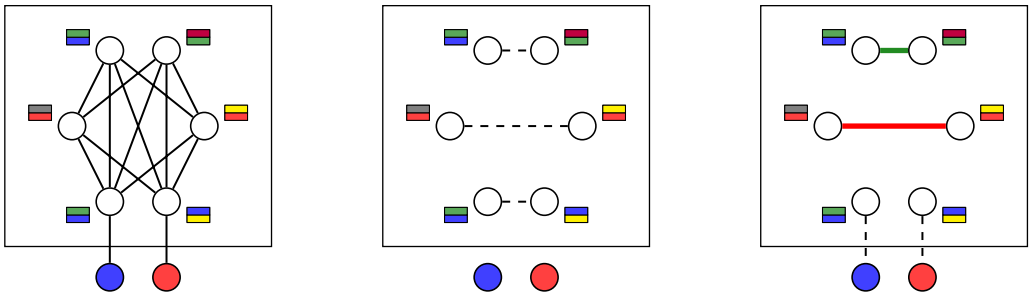


Figure 3.2: Illustration of a colorful matching in Definition 3.3.9. The vertices inside each box depict an almost-clique and remaining vertices are neighboring colored vertices that are outside this almost-clique (and define blocked colors). The sampled list $L(\cdot)$ is also shown next to each vertex. The bottom non-edge cannot be part of any colorful-matching because the (blue) color shared by both of its endpoints is blocked for the left vertex.

both $\chi_3(u_i) = \chi_3(v_i) = c_{u_i,v_i}$. By Definition 3.3.9, χ_3 would be a valid partial coloring of C. Moreover,

$$|\Psi_{\chi_3}(C)| = (\Delta + 1 - |\overline{M}|) \qquad \text{and} \qquad \left|V_{\chi_3}^{\mathsf{uncolored}}(C)\right| = |V(C)| - 2\,|\overline{M}|\,. \tag{3.3}$$

Consequently, we only need to find a suitably sized $\overline{M}$ to prove Lemma 3.3.8. In order to prove this, we give the following procedure for finding a colorful matching with a sufficiently large size.

Let us first make one more definition in spirit of Remark 3.3.7: We further consider the process of sampling the list $L_1(v)$ by instead sampling $k := p/2q$ lists $L_{1,1}(v), \ldots, L_{1,k}(v)$ where we sample each color c in each $L_{1,j}$ w.p. $q := 1/\sqrt{40\varepsilon}\Delta$ and set $L_1(v) := L_{1,1}(v) \cup \ldots \cup L_{1,k}(v)$. By the same argument in Remark 3.3.7, this is without loss of generality. For each list $L_{1,j}$, we define:

ColorfulMatching.

Input: Almost-clique C with lists $L_{1,j}$ and blocked-lists B – **Output:** A colorful matching $\overline{M}$.

1. Let $\text{NE}_1 := \text{NE}(C)$ and iterate over colors $c_i \in [\Delta+1]$ in an arbitrary order:
   (a) If there exists $f := (u,v) \in \text{NE}_i$ s.t. $c \in (L_{1,j}(u) \setminus B(u)) \cap (L_{1,j}(v) \setminus B(v))$, then:
      i. add f to $\overline{M}$ and let NE_{i+1} be NE_i minus all non-edges incident on u, v;
      ii. continue to iteration $i+1$.
   (b) Otherwise, let $\text{NE}_{i+1} := \text{NE}_i$.

It is easy to verify that $\overline{M}$ of ColorfulMatching is a colorful non-edge matching (Definition 3.3.9). We now prove that $\overline{M}$ is also sufficiently large with *constant* probability. We then boost this probability of success by considering all $L_{1,j}$'s simultaneously.

Lemma 3.3.10. *W.p.* $\geq 1/50$*, for* $\overline{M}$ *output by* ColorfulMatching*,* $\left|\overline{M}\right| \geq \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor$.

Proof. Define $t := |\text{NE}(C)|/(100\varepsilon\Delta)$. Our goal is to prove that $\left|\overline{M}\right| \geq t/2$ w.p. $\geq 1/50$. For any iteration $i \in [\Delta+1]$ of ColorfulMatching and any color c, we further define:

- $\text{Present}_i(c)$: the set of non-edges in NE_i that do not block c in either of the endpoints, i.e., $\text{Present}_i(c) := \{(u,v) \in \text{NE}_i \mid c \notin B(u) \cup B(v)\}$ – let $\text{pres}_i(c) := |\text{Present}_i(c)|$.
- $\overline{M}_i$: the non-edge matching $\overline{M}$ at the beginning of iteration i – let $\overline{m}_i := \left|\overline{M}_i\right|$.

We first show that for most colors, $\text{pres}_1(c)$ is sufficiently large.

Claim 3.3.11. *For at least* $\Delta/2$ *colors* $c \in [\Delta+1]$*, we have* $\text{pres}_1(c) \geq (0.9) \cdot |\text{NE}_1|$.

Proof. For any $f = (u,v) \in \text{NE}$, we have,

$$|B(u) \cup B(v)| \leq \mathsf{outdeg}(u) + \mathsf{outdeg}(v) \leq 6\varepsilon\Delta.$$

(in Lemma 3.3.8, $|B(w)| \leq \mathsf{outdeg}(w)$ and by definition of almost-cliques in Lemma 3.2.1)

As such,

$$\sum_{c\in[\Delta+1]} \text{pres}_1(c) = \sum_{f=(u,v)\in \text{NE}_1} (\Delta+1) - |B(u)\cup B(v)| \geq \sum_{f=(u,v)\in \text{NE}_1} (1-6\varepsilon)\cdot(\Delta+1)$$

(by the equation above)

$$\geq \sum_{f=(u,v)\in \text{NE}_1} \frac{19}{20}\cdot(\Delta+1) = |\text{NE}_1|\cdot\frac{19}{20}\cdot(\Delta+1).$$

(for $\varepsilon < 1/(20\cdot 6) = 1/120$)

As $\text{pres}_i(c) \leq |\text{NE}_1|$ for all c, an application of (reverse) Markov bound implies the claim as otherwise,

$$\sum_{c\in[\Delta+1]} \text{pres}_1(c) < (\Delta/2)\cdot|\text{NE}_1| + (\Delta/2)\cdot(0.9)\cdot|\text{NE}_1| < |\text{NE}_1|\cdot\frac{19}{20}\cdot(\Delta+1),$$

contradicting the above.

□

We refer to each of the $(\Delta/2)$ colors in Claim 3.3.11 as *heavy* colors. The following claim lower bounds $\text{pres}_i(c)$ for the heavy colors at the beginning of the iteration i in which ColorfulMatching is processing them (assuming we already have not found a large enough $\overline{M}$).

Claim 3.3.12. *For any heavy color $c_i \in [\Delta+1]$, if $\overline{m}_i < t$ then, $\text{pres}_i(c_i) \geq (0.75)\cdot|\text{NE}_1|$.*

Proof. Any non-edge $f = (u,v)$ in $\overline{M}_i$ is responsible for deleting $\leq \overline{\mathsf{indeg}}(u) + \overline{\mathsf{indeg}}(v)$ non-edges of $\text{pres}_1(c_i)$ from NE_i (recall that $\overline{\mathsf{indeg}}(\cdot)$ denotes the number of non-neighbors in C). As such,

$$\text{pres}_i(c_i) \geq \text{pres}_1(c_i) - \overline{m}_i\cdot(\overline{\mathsf{indeg}}(u) + \overline{\mathsf{indeg}}(v)) \geq (0.9)\,|\text{NE}_1| - t\cdot(12\varepsilon\Delta)$$

(by definition of heavy colors and bound on $\overline{\mathsf{indeg}}(\cdot)$ in Lemma 3.2.1)

$$\geq (0.9)\,|\mathrm{NE}_1| - (|\mathrm{NE}_1|/100\varepsilon\Delta)\cdot(12\varepsilon\Delta) > (0.75)\cdot|\mathrm{NE}_1|\,,$$

(by the choice of $t = |\mathrm{NE}_1|/100\varepsilon\Delta$)

finishing the proof.

□

We now use the above claim to say that whenever we are processing a heavy color, there is a "good" probability that we add a new edge to $\overline{M}$ (again assuming $\overline{M}$ is already not large enough).

Claim 3.3.13. *For any heavy color $c_i \in [\Delta+1]$:* $\Pr\left(\overline{m}_{i+1} = \overline{m}_i + 1 \mid \overline{m}_i < t\right) \geq (3/200)\cdot |\mathrm{NE}_1|/\varepsilon\Delta^2$.

Proof. $\overline{m}_{i+1}$ increases to $\overline{m}_i+1$ if at least one $(u,v) \in \mathrm{Present}_i(c_i)$ have $c_i \in L_{1,j}(u)\cap L_{1,j}(v)$. By a simple application of the inclusion-exclusion principle,

$$\begin{aligned}\Pr\left(\overline{m}_{i+1} = \overline{m}_i + 1 \mid \overline{m}_i < t\right) &\geq \Pr\left(\text{at least one } (u,v) \in \mathrm{Present}_i(c_i) \text{ have } c_i \in L_{1,j}(u)\cap L_{1,j}(v)\right)\\ &\geq \sum_{\substack{(u,v)\in\\ \mathrm{Present}_i(c_i)}} \Pr\left(c_i \in L_{1,j}(u)\cap L_{1,j}(v)\right) - \sum_{\substack{(u,v),(u',v')\in\\ \mathrm{Present}_i(c_i)}} \Pr\left(c_i \in L_{1,j}(u)\cap L_{1,j}(v)\cap L_{1,j}(u')\cap L_{1,j}(v')\right).\end{aligned} \tag{3.4}$$

The first term above is easy to bound as each c_i belongs to $L_{1,j}(u)\cap L_{1,j}(v)$ w.p. q^2, hence,

$$\sum_{\substack{(u,v)\in\\ \mathrm{Present}_i(c_i)}} \Pr\left(c_i \in L_{1,j}(u)\cap L_{1,j}(v)\right) = \mathrm{pres}_i(c_i)\cdot q^2.$$

For the second term of Eq (3.4), there are two types of pairs of (distinct) non-edges $(u,v),(u',v')$ that we need to take into account: the ones that share exactly one endpoint and the ones that do not share any endpoint. There are at most $\mathrm{pres}_i(c_i)\cdot 6\varepsilon\Delta$ many edges of the first type as maximum non-degree of any vertex is at most $6\varepsilon\Delta$ by Lemma 3.2.1; there

are also at most $\text{pres}_i(c_i)^2$ many edges of the second type. Hence,

$$\sum_{\substack{(u,v),(u',v')\in \\ \text{Present}_i(c_i)}} \Pr\left(c_i \in L_{1,j}(u) \cap L_{1,j}(v) \cap L_{1,j}(u') \cap L_{1,j}(v')\right) \leq \text{pres}_i(c_i) \cdot 6\varepsilon\Delta \cdot q^3 + \text{pres}_i(c_i)^2 \cdot q^4.$$

We can plugin the above two bounds in Eq (3.4), and have,

$$\begin{aligned}
\Pr\left(\overline{m}_{i+1} = \overline{m}_i + 1 \mid \overline{m}_i < t\right) &\geq \text{pres}_i(c_i) \cdot q^2 - \text{pres}_i(c_i) \cdot 6\varepsilon\Delta \cdot q^3 - \text{pres}_i(c_i)^2 \cdot q^4 \\
&\geq \frac{9}{10} \cdot \text{pres}_i(c_i) \cdot q^2 \cdot (1 - \text{pres}_i(c_i) \cdot q^2) \\
&\qquad (6\varepsilon\Delta \cdot q^3 < (1/10) \cdot q^2 \text{ for } q = 1/\sqrt{40\varepsilon}\Delta \text{ and } \varepsilon < 1/90) \\
&\geq \frac{9}{10} \cdot \text{pres}_i(c_i) \cdot q^2 \cdot (1 - 4\varepsilon\Delta^2 \cdot q^2) \\
(\text{pres}_i(c_i) \leq |\text{NE}_1| \leq (1/2) \cdot (1+3\varepsilon)\Delta \cdot 6\varepsilon\Delta \leq 4\varepsilon\Delta^2 &\text{ by Lemma 3.2.1 for } \varepsilon < 1/18) \\
&= \frac{9}{10} \cdot \text{pres}_i(c_i) \cdot q^2 \cdot (1 - 4\varepsilon\Delta^2 \cdot \frac{1}{40\varepsilon\Delta^2}) \quad (\text{by } q = 1/\sqrt{40\varepsilon}\Delta) \\
&\geq \frac{4}{5} \cdot \text{pres}_i(c_i) \cdot q^2 \\
&\geq \frac{4}{5} \cdot (\frac{3}{4} \cdot |\text{NE}_1|) \cdot \frac{1}{40\varepsilon\Delta^2} \\
&\qquad (\text{by Claim 3.3.12 and choice of } q = 1/\sqrt{40\varepsilon}\Delta) \\
&= (3/200)\,|\text{NE}_1|\,/\varepsilon\Delta^2,
\end{aligned}$$

as desired.

□

We are now ready to conclude the proof of Lemma 3.3.10. Let $\theta := (3/200) \cdot |\text{NE}_1|\,/\varepsilon\Delta^2$ (the RHS of Claim 3.3.13); note that $\theta < 1$ because $|\text{NE}_1| < 4\varepsilon\Delta^2$ as calculated in Claim 3.3.13. Let Z be a random variable sampled from binomial distribution $\mathcal{B}(\Delta/2, \theta)$. By Claim 3.3.13 and the fact that there are $\Delta/2$ heavy colors, plus a straightforward coupling argument, for

every $t' \leq t$:

$$\Pr\left(\left|\overline{M}\right| \geq t'\right) \geq \Pr\left(Z \geq t'\right). \tag{3.5}$$

On the other hand, $\mathbb{E}[Z] = \Delta/2 \cdot \theta = (3/400) \cdot |\mathrm{NE}_1| / \varepsilon\Delta$. By Chernoff bound (Proposition 2.2.2),

$$\begin{aligned}
\Pr\left(Z < (1/200) \cdot |\mathrm{NE}_1| / \varepsilon\Delta\right) &= \Pr\left(Z < 2/3 \cdot \mathbb{E}[Z]\right) \leq \exp\left(-\frac{\mathbb{E}[Z]}{27}\right) \\
&= \exp\left(-\frac{|\mathrm{NE}_1|}{3600 \cdot \varepsilon\Delta}\right) \leq \exp\left(-\frac{200\varepsilon\Delta}{3600\varepsilon\Delta}\right) \\
&\text{(by the discussion after Lemma 3.3.8, } |\mathrm{NE}_1| \geq 200 \cdot \varepsilon\Delta) \\
&\leq \exp\left(-\frac{1}{18}\right) \leq (1 - 1/36). \\
&\text{(as } e^{-x} \leq 1 - x/2 \text{ for } x \in (0, 1/2))
\end{aligned}$$

As such, with probability at least $1/36 > 1/50$, $Z \geq |\mathrm{NE}_1| / 200\varepsilon\Delta$. As RHS of this bound $< t$ and by Eq (3.5), we also obtain that with probability at least $1/50$, $\left|\overline{M}\right| \geq |\mathrm{NE}_1| / 200 \cdot \varepsilon\Delta$.

□

We can now conclude the proof of Lemma 3.3.8 using Lemma 3.3.10 as follows.

Proof of Lemma 3.3.8. Eq (3.3) allows us to prove Lemma 3.3.8 by showing existence of a sufficiently large colorful (non-edge) matching $\overline{M}$ in almost-clique C (and lists L_1). Let $\overline{M}$ be the largest colorful matching C and call the event that $\left|\overline{M}\right| \geq \left\lfloor \frac{|\mathrm{NE}(C)|}{200\varepsilon\Delta} \right\rfloor$ as "success". Recall that we partitioned the process of sampling each lists $L_1(v)$ into *independently* sampling $k = p/2q$ lists $L_{1,1}(v), \ldots, L_{1,k}(v)$ as described before ColorfulMatching. By Lemma 3.3.10, each of these lists would imply the event success with w.p. at least $1/50$. As such,

$$\left(1 - \Pr(\text{"success"})\right) \leq (1 - 1/50)^k \leq \exp\left(-\frac{p}{100q}\right) = \exp\left(-\frac{10\alpha \cdot \ln n \cdot \sqrt{40\varepsilon} \cdot \Delta}{100 \cdot \varepsilon^2 \cdot 2 \cdot (\Delta + 1)}\right)$$

(by definition of $p = \left(10\alpha/\varepsilon^2\right) \cdot \ln n / (2 \cdot (\Delta + 1))$ in Eq (3.1) and $q = 1/\sqrt{40\varepsilon}\Delta$)

$$\leq \exp\left(-\alpha \cdot \ln n\right) \ll n^{-10}.$$

(by a crude calculation of constants and using $\sqrt{\varepsilon} < 1$, $\varepsilon < 1/10$, and $\alpha > 10$)

Hence, with high probability, we have a colorful matching $\overline{M}$ with $\left|\overline{M}\right| \geq \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor$. The lemma now follows from this and Eq (3.3) as we can simply pick the first $\left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor$ edges of $\overline{M}$ as the colorful (non-edge) matching in Eq (3.3). This concludes the proof.

□

Final Coloring of the Almost-Clique

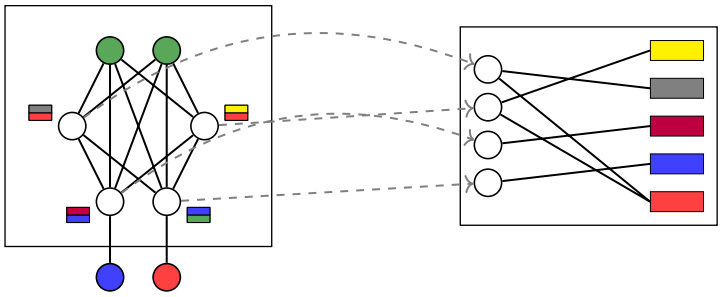
We now finalize the coloring of the almost-clique. Recall that in the previous part, we already obtained a partial coloring χ_3 of C in Lemma 3.3.8. We now extend this coloring to a proper coloring of C entirely from lists L, satisfying the properties required by Lemma 3.3.2.

Lemma 3.3.14. *Consider the setting of Lemma 3.3.2 and let χ_3 be the partial coloring of Lemma 3.3.8. With high probability, there exists a partial coloring χ_4 such that for every $v \in V_{\chi_3}^{\mathsf{uncolored}}(C)$, we have $\chi_4(v) \in L(v) \cap \Psi_{\chi_3}(C) \setminus B(v)$.*

It is easy to see that once we have χ_4 from Lemma 3.3.14 we are done as we colored every vertex v of C from $L(v) \setminus B(v)$. We prove this lemma in the rest of this part. Throughout this section, we fix the partial coloring χ_3 obtained from Lemma 3.3.8 and only consider randomness of the lists $L_2(\cdot)$ (recall the definition from Remark 3.3.7) which are *independent* of this conditioning on χ_3. To continue, we need the following key definition (see Figure 3.3 for an illustration).

Definition 3.3.15 (Sampled-Palette-Graph)**.** *We define the* palette-graph $\mathcal{G} := \mathcal{G}(\mathcal{L}, \mathcal{R}, \mathcal{E})$ *as the following bipartite graph:*

- $\mathcal{L} := V_{\chi_3}^{\mathsf{uncolored}}(C)$ *and* $\mathcal{R} := \Psi_{\chi_3}(C)$*. $\mathcal{L}$ is called the* vertex-side *and $\mathcal{R}$ is called the* color-side.
- *for any $v \in V_{\chi_3}^{\mathsf{uncolored}}(C)$ and $c \in \Psi_{\chi_3}(C)$, the edge $(v, c) \in \mathcal{E}$ iff $c \notin B(v)$.*



Almost-Clique Sampled-Palette-Graph

Figure 3.3: Illustration of the sampled-palette-graph in Definition 3.3.15. The figure on the left is a partially colored almost-clique plus the neighboring colored vertices that are outside this almost-clique. The sampled list $L(\cdot)$ is also shown next to each vertex.

We refer to the subgraph $\mathcal{G}_{L_2} := \mathcal{G}_{L_2}(\mathcal{L}, \mathcal{R}, \mathcal{E}_{L_2})$ *of* $\mathcal{G}$ *with the same vertices and edges* (v, c) *where now* c *additionally belongs to* $L_2(v)$ *as the* sampled-palette-graph. *By definition of* L_2, $\mathcal{G}_{L_2}$ *is obtained from* $\mathcal{G}$ *by sampling each edge w.p.* $p/2$.

Consider the sampled-palette-graph $\mathcal{G}_{L_2}$. We show that there exists a left-saturating matching (a one that matches every vertex on left, i.e., $\mathcal{L}$) in this graph with high probability. Having obtained this matching, we will be done as we can color each vertex in $v \in V_{\chi_3}^{\mathsf{uncolored}}(C)$ by the color $c \in \Psi_{\chi_3}(C)$ which v is matched to; by Definition 3.3.15 color $c \in L(v) \setminus B(v)$ and thus this gives us a proper coloring χ_4 of C from lists $L(v) \setminus B(v)$ as required in Lemma 3.3.14.

We start the proof by recounting the useful properties of the graph $\mathcal{G}$ itself (from which $\mathcal{G}_{L_2}$ is sampled) and then use these properties in the next part to show the existence of this matching.

Claim 3.3.16. *There exists an integer* $N \geq 1$ *such that the palette-graph* $\mathcal{G}(\mathcal{L}, \mathcal{R}, \mathcal{E})$ *of* C *and* χ_3 *satisfies the following properties:*

(i) $|\mathcal{L}| = N$ *and* $|\mathcal{R}| \leq 2N$;

(ii) minimum degree of vertices in $\mathcal{L}$ *is* $\min_{v \in \mathcal{L}} \mathsf{deg}_{\mathcal{G}}(v) \geq 9N/10$;

(iii) for any $S \subseteq \mathcal{L}$ *of size* $|S| \geq 4N/5$, $\sum_{v \in S} \mathsf{deg}_{\mathcal{G}}(v) \geq (|S| \cdot N) - N/3$.

Proof. Let $N := \left|V_{\chi_3}^{\mathsf{uncolored}}(C)\right|$ which by definition implies that $|\mathcal{L}| = N$. Let us relate N to Δ:

$$\begin{aligned}
N = \left|V_{\chi_3}^{\mathsf{uncolored}}(C)\right| &= |V(C)| - 2 \cdot \left\lfloor \frac{|\mathrm{NE}(C)|}{200\varepsilon\Delta} \right\rfloor && \text{(by Lemma 3.3.8)} \\
&\geq |V(C)| - |V(C)| \cdot \frac{6\varepsilon\Delta}{200\varepsilon\Delta} && (\overline{\mathsf{indeg}}(v) \leq 6\varepsilon\Delta \text{ for all } v \in C \text{ by Lemma 3.2.1}) \\
&\geq (1-\varepsilon) \cdot \Delta \cdot (194/200) && (|V(C)| \geq (1-\varepsilon)\Delta \text{ by Lemma 3.2.1}) \\
&\geq (3/4) \cdot \Delta. && (\text{for } \varepsilon < 1/10)
\end{aligned}$$

On the other hand, $|\mathcal{R}| \leq \Delta + 1$ (there are $\Delta + 1$ colors to begin with), and hence by the above inequality, $|\mathcal{R}| \leq 2N$. This proves Item (i).

Let us now calculate $\mathsf{deg}_{\mathcal{G}}(v)$ of any vertex $v \in \mathcal{L}$ in $\mathcal{G}$. We have,

$$\begin{aligned}
\mathsf{deg}_{\mathcal{G}}(v) &\geq |\mathcal{R}| - |B(v)| && (\text{any } v \in \mathcal{L} \text{ is connected to } c \in \mathcal{R} \text{ iff } c \notin B(v)) \\
&\geq |\mathcal{R}| - \mathsf{outdeg}_C(v) && (|B(v)| \leq \mathsf{outdeg}_C(v) \text{ in the statement of Lemma 3.3.2}) \\
&\geq |\mathcal{R}| - \left(\Delta - \left(|V(C)| - \overline{\mathsf{indeg}}_C(v) - 1\right)\right) \\
&\qquad (v \text{ has exactly } |V(C)| - \overline{\mathsf{indeg}}_C(v) - 1 \text{ neighbors inside } C \text{ and at most } \Delta \text{ neighbors in total}) \\
&= |\mathcal{R}| - (\Delta + 1) + |V(C)| - \overline{\mathsf{indeg}}_C(v) && (\text{by reorganizing the terms}) \\
&= |\mathcal{R}| - \left(|\mathcal{R}| + \left\lfloor \frac{|\mathrm{NE}(C)|}{200\varepsilon\Delta} \right\rfloor\right) + \left(|\mathcal{L}| + 2 \cdot \left\lfloor \frac{|\mathrm{NE}(C)|}{200\varepsilon\Delta} \right\rfloor\right) - \overline{\mathsf{indeg}}_C(v) \\
&\qquad (\text{by Lemma 3.3.8 as } |\mathcal{R}| = |\Psi_{\chi_3}(C)| \text{ and } |\mathcal{L}| = \left|V_{\chi_3}^{\mathsf{uncolored}}(C)\right|) \\
&= |\mathcal{L}| + \left\lfloor \frac{|\mathrm{NE}(C)|}{200\varepsilon\Delta} \right\rfloor - \overline{\mathsf{indeg}}_C(v).
\end{aligned}$$

By this equation, we have,

$$\begin{aligned}
\min_{v \in \mathcal{L}} \mathsf{deg}_{\mathcal{G}}(v) \geq |\mathcal{L}| - \max_{v \in V(C)} \overline{\mathsf{indeg}}_C(v) &\geq N - 6\varepsilon\Delta && \text{(by definition of } N \text{ and Lemma 3.2.1)} \\
&\geq N - 8\varepsilon N \geq 9N/10. && (\text{for } \varepsilon < 1/80)
\end{aligned}$$

This proves Item (ii).

Let us now prove Item (iii). Instead of proving this item directly, we prove it for a *subgraph* $\mathcal{G}'$ of $\mathcal{G}$ that we define shortly; clearly, such a bound would also hold for $\mathcal{G}$ as $\mathsf{deg}_{\mathcal{G}}(v) \geq \mathsf{deg}_{\mathcal{G}'}(v)$ for every subgraph $\mathcal{G}'$ of $\mathcal{G}$. As for the definition of $\mathcal{G}'$, this is a subgraph obtained by simply picking $|\mathcal{L}| + \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor - \overline{\mathsf{indeg}}_C(v)$ (RHS of the equation for $\mathsf{deg}_{\mathcal{G}}(v)$ above) many edges incident on each vertex v *arbitrarily* and discard the remaining edges. This way, the maximum degree of vertices in $\mathcal{L}$ in $\mathcal{G}'$ can be bounded as:

$$\max_{v \in \mathcal{L}} \mathsf{deg}_{\mathcal{G}'}(v) \leq |\mathcal{L}| + \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor = N + \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor.$$

On the other hand, using the bound on each $\mathsf{deg}_{\mathcal{G}'}(v)$,

$$\begin{aligned}\sum_{v \in \mathcal{L}} \mathsf{deg}_{\mathcal{G}'}(v) &\geq \sum_{v \in \mathcal{L}} (|\mathcal{L}| + \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor - \overline{\mathsf{indeg}}_C(v)) \\ &= |\mathcal{L}|^2 + |\mathcal{L}| \cdot \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor - 2\,|\text{NE}(C)|\,.\end{aligned}$$

By the bound on maximum degree of vertices in $\mathcal{L}$ in $\mathcal{G}'$ and the equation above, for any set S of size at least $4N/5$,

$$\begin{aligned}\sum_{v \in S} \mathsf{deg}_{\mathcal{G}'}(v) &\geq \sum_{v \in L} \mathsf{deg}_{\mathcal{L}} - |\mathcal{L} \setminus S| \cdot \left(N + \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor \right) \\ &\geq N^2 + N \cdot \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor - 2\,|\text{NE}(C)| - (N - |S|) \cdot \left(N + \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor \right) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\text{as } N = |\mathcal{L}|) \\ &= |S| \cdot N + |S| \cdot \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor - 2\,|\text{NE}(C)| \\ &\geq |S| \cdot N + (4N/5) \cdot \left\lfloor \frac{|\text{NE}(C)|}{200\varepsilon\Delta} \right\rfloor - 2\,|\text{NE}(C)|\,. \qquad (3.6)\end{aligned}$$

Let us now consider two cases. Suppose first $|\text{NE}(C)| > 400\varepsilon\Delta$; in this case, we can remove

the $\lfloor \cdot \rfloor$ in the RHS above as follows and write,

$$\begin{aligned}
\text{RHS of Eq (3.6)} &\geq |S| \cdot N + (4N/5) \cdot \frac{|\mathrm{NE}(C)|}{400\varepsilon\Delta} - 2\,|\mathrm{NE}(C)| \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\text{as } \lfloor x \rfloor \geq x - 1 \geq x/2 \text{ for } x \geq 2) \\
&\geq |S| \cdot N + \frac{3 \cdot |\mathrm{NE}(C)|}{2000\varepsilon} - 2\,|\mathrm{NE}(C)| \qquad (\text{as } \Delta \leq 4/3N \text{ calculated above}) \\
&\geq |S| \cdot N. \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\text{for } \varepsilon < 3/4000)
\end{aligned}$$

Now suppose $|\mathrm{NE}(C)| \leq 400\varepsilon\Delta$ instead. In this case, we can simply ignore the second term in the RHS of Eq (3.6) and write,

$$\begin{aligned}
\text{RHS of Eq (3.6)} &\geq |S| \cdot N - 2\,|\mathrm{NE}(C)| \geq |S| \cdot N - 800\varepsilon\Delta \geq |S| \cdot N - 3200/3 \cdot \varepsilon N \geq N^2 - N/3. \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\text{as } \Delta \leq 4/3N \text{ calculated above and for } \varepsilon < 1/3200)
\end{aligned}$$

Hence by Eq (3.6), in both cases, $\sum_{v \in S} \mathsf{deg}_{\mathcal{G}'}(v) \geq |S| \cdot N - N/3$, proving Item (iii).

□

Claim 3.3.16 describes our desired properties of the palette-graph $\mathcal{G}$. As stated in Definition 3.3.15, $\mathcal{G}_{L_2}$ can be seen as a *pseudo-random graph* obtained by independently sampling each edge of $\mathcal{G}$ w.p. $p/2$ independently (instead of sampling each edge of a bipartite-clique in the standard Erdős-Rényi (bipartite) random graphs). In the following, we prove that these pseudo-random graphs have a left-saturating matching w.h.p. We shall note that existence of a left-saturating perfect matching in truly random graphs is a standard fact (see [61]) – here we prove a generalization of this for random subgraphs of any graph that satisfies properties of Claim 3.3.16 (the graphs in Claim 3.3.16 may be missing up to $N^2 + N/3$ edges of a bipartite clique and hence this extension requires a non-trivial argument).

Lemma 3.3.17. *Let $H(L, R, E)$ be a bipartite graph such that for some integer $N \leq n$:*

(i) $|L| = N$ and $|R| \leq 2N$;

(ii) minimum degree of vertices in L is at least $9N/10$;

(iii) for any $S \subseteq L$ of size $|S| \geq 4N/5$, $\sum_{v \in S} \mathsf{deg}_H(v) \geq (|S| \cdot N) - N/3$.

Suppose $\widetilde{H}(L, R, \widetilde{E})$ is obtained from H by sampling each edge of E w.p. $\widetilde{p} \geq \frac{100 \ln n}{N}$. Then, with probability at least $1 - 1/n^5$, $\widetilde{H}$ has a matching that matches every vertex in L.

The proof of Lemma 3.3.17 is technical and requires a detour and hence we postpone it to the next part to keep the flow of the current argument. Let us now show that this lemma proves Lemma 3.3.14 and the entire proof of coloring for dense vertices (Lemma 3.3.2).

Proof of Lemma 3.3.14. As stated earlier, existence of a left-saturating matching in the sampled palette-graph $\mathcal{G}_{L_2}$ allows us to color every vertex in $v \in V_{\chi_3}^{\mathsf{uncolored}}(C)$ by the color $c \in \Psi_{\chi_3}(C)$ which v is matched to in $\mathcal{G}_{L_2}$; by Definition 3.3.15 color $c \in L(v) \setminus B(v)$ and thus this gives us the desired coloring χ_4 in Lemma 3.3.14. Moreover, $\mathcal{G}_{L_2}$ is obtained from $\mathcal{G}$ by sampling each edge of G with probability,

$$\frac{p}{2} = \frac{\left(5\alpha/\varepsilon^2\right) \cdot \ln n}{\Delta + 1} \geq \frac{100 \ln n}{N}.$$

(by Eq (3.1) for the first term and since $\Delta \leq 4/3N$ as calculated in Claim 3.3.16 and for $\varepsilon < 1/10$)

We can thus use Claim 3.3.16 and Lemma 3.3.17 (by taking $H = \mathcal{G}$ and $\widetilde{H} = \mathcal{G}_{L_2}$ and $\widetilde{p} = p/2$); this implies that $\mathcal{G}_{L_2}$ has a left-saturating matching w.h.p., finalizing the proof.

□

Lemma 3.3.2 for coloring the almost-clique C now follows immediately from Lemma 3.3.8 and Lemma 3.3.14 as χ_3 and χ_4 form a proper coloring of all vertices in C. The only thing left to do is thus proving Lemma 3.3.17 which we do in the next part.

3.3.5. Existence of Left-Saturating Matchings in Pseudo-Random Graphs

We now prove Lemma 3.3.17 restated below.

Lemma (Restatement of Lemma 3.3.17)**.** *Let $H(L, R, E)$ be a graph such that for some $N \leq n$:*

(i) $|L| = N$ and $|R| \leq 2N$;

(ii) minimum degree of vertices in L is at least $9N/10$;

(iii) for any $S \subseteq L$ of size $|S| \geq 4N/5$, $\sum_{v \in S} \mathsf{deg}_H(v) \geq (|S| \cdot N) - N/3$

Suppose $\widetilde{H}(L, R, \widetilde{E})$ is obtained from H by sampling each edge of E w.p. $\widetilde{p} \geq \frac{100 \ln n}{N}$. Then, with probability at least $1 - 1/n^5$, $\widetilde{H}$ has a matching that matches every vertex in L.

By Hall's Theorem in Fact 2.1.1, in order to prove Lemma 3.3.17, we only need to show that for every set $S \subseteq L$, $\left|N_{\widetilde{H}}(S)\right| \geq |S|$. We prove this in the following by considering two different cases based on the size of S. The first and easy case is when S is not too large; the proof in this part follows the standard arguments for showing existence of a perfect matching in a random graph.

Claim 3.3.18. *W.h.p., for every set $S \subseteq L$ of size $|S| < 4N/5$, $\left|N_{\widetilde{H}}(S)\right| \geq |S|$.*

Proof. Fix any choice of set $S \subseteq L$ with $|S| < 4N/5$ and any set $T \subseteq R$ with $|T| = |S| - 1$; we also define $\overline{T} := R \setminus T$. We prove that there exists an edge in $\widetilde{H}$ between S and $\overline{T}$, thus ensuring that $T \neq N_{\widetilde{H}}(S)$; moreover, this happens with such a high probability that we can take a union bound over all choices for both S and T.

Let $E(S, \overline{T})$ denote the set of edges between S and $\overline{T}$ in H (the base graph). We have,

$$\left|E(S, \overline{T})\right| \geq \sum_{v \in S} (\mathsf{deg}_H(v) - |T|) \geq \sum_{v \in S} (9N/10 - 4N/5) = |S| \cdot N/10.$$

(by Item (ii) $\mathsf{deg}_H(v) \geq 9N/10$ and since $|T| < |S| < 4N/5$)

Using this, we can calculate,

$$\Pr\left(\widetilde{E} \cap E(S, \overline{T}) = \emptyset\right) = (1 - \widetilde{p})^{|E(S, \overline{T})|}$$

$$\leq \exp\Big(-\frac{100\log n}{N}\cdot|S|\cdot N/10\Big) = \exp\Big(-|S|\cdot 10\ln n\Big).$$

(by definition of $\widetilde{p}$ and equation above)

We now take a union bound over all choices of T and obtain that,

$$\Pr\left(\left|N_{\widetilde{H}}(S)\right| < |S|\right) \leq \sum_{\substack{T\subseteq R\\ |T|=|S|-1}} \Pr\left(\widetilde{E}\cap E(S,\overline{T}) = \emptyset\right) \leq (2N)^{|S|}\cdot\exp\Big(-|S|\cdot 10\ln n\Big)$$

(as $|R|\leq 2N$ by Item (i) and $|T|<|S|$ by definition)

$$\leq \exp\left(|S|\cdot\ln n + |S| - |S|\cdot 10\log n\right) \ll n^{-8\cdot|S|}. \qquad \text{(as } N\leq n\text{)}$$

We can now also take a union bound over all S by bundling them based on their size:

$$\Pr\left(\exists S : \left|N_{\widetilde{H}}(S)\right| < |S| \ , \ |S| < 4N/5\right) \leq \sum_{\substack{S\subseteq L\\ |S|<4N/5}} \Pr\left(\left|N_{\widetilde{H}}(S)\right| < |S|\right)$$

$$\leq \sum_{k=1}^{4N/5-1}\binom{N}{k}\cdot n^{-8k} \leq \sum_{k=1}^{4N/5-1} n^{-7k} \leq n^{-6}.$$

(by the equation above and since $\binom{N}{k}\leq N^k\leq n^k$)

As such, for all sets S considered in this case, with high probability, $\left|N_{\widetilde{H}}(S)\right|\geq|S|$.

□

We now prove that even when size of S is large, i.e., is between $4N/5$ and N, still the neighborhood of S is of size at least equal to S. The proof of this part deviates from the previous approach and instead crucially use Item (iii) of Lemma 3.3.17 that states that for large enough S, there are many edges incident on S in H (much more than by using only the min-degree bound in Item (ii)).

Claim 3.3.19. *W.h.p., for every set $S\subseteq L$ of size $|S|\geq 4N/5$, $\left|N_{\widetilde{H}}(S)\right|\geq|S|$.*

Proof. Fix any choice of set $S \subseteq L$ with $|S| \geq 4N/5$. Recall that by Item (iii), we have,

$$\sum_{v \in S} \mathsf{deg}_H(v) \geq (s \cdot N) - N/3.$$

For simplicity, in this proof, we *arbitrarily* remove incident edges on vertices of S such that the total number of edges incident on S becomes *exactly* the RHS above.

Let $s := |S|$, $r := |R|$, and define r indicator random variables $X_1, \ldots, X_r$ where $X_i = 1$ iff $v_i \in R$ does *not* belong to $N_{\widetilde{H}}(S)$. Additionally, define d_i as the number of edges that $v_i \in R$ has to vertices in S. Let $X := \sum_{i=1}^{r} X_i$ and note that $\left|N_{\widetilde{H}}(S)\right| = r - X$. Hence, our goal is to compute $\Pr(X > r - s)$. Let us define $\widetilde{q} := (1 - \widetilde{p})$. We have the following properties for X_i's and d_i's (the last equality is by the modification above):

$$\begin{aligned} &\forall\ i \in [r] \qquad \Pr(X_i = 1) = \widetilde{q}^{d_i}; \\ &\forall\ i \in [r] \qquad d_i \leq s; \\ &\sum_{i=1}^{r} d_i = (s \cdot N) - N/3. \end{aligned} \tag{3.7}$$

Working with variables $X_1, \ldots, X_r$ directly is quite cumbersome and we instead define the random variables $Y_1, \ldots, Y_r$ where:

$$\begin{aligned} &\forall\ i \in [N-1] && \Pr(Y_i = 1) = \widetilde{q}^{s}; \\ & && \Pr(Y_N = 1) = \widetilde{q}^{s - N/3}; \\ &\forall\ i \in [N+1 : r] && \Pr(Y_i = 1) = 1. \end{aligned}$$

For some intuition, notice that random variables Y_i would be equal to X_i's *if* the base graph H is such that there are $N-1$ vertices with degree s to S, one vertex with degree $s - N/3$, and all other vertices in R have degree zero to S (such a choice satisfies Eq (3.7)). In the following, we first prove that such a base graph H is the "worst case example" in proving $\left|N_{\widetilde{H}}(S)\right| \geq |S|$ and then focus on this case directly. More formally, we prove that

for $Y = \sum_{i=1}^{r} Y_i$ and every $t \geq 1$,

$$\Pr(Y \geq t) \geq \Pr(X \geq t). \tag{3.8}$$

Again for intuition, notice that in defining Y's and corresponding worst-case base graph H, we simply moved degrees of all vertices in R to a *minimal* set of N vertices while satisfying the degree requirements of Eq (3.7); one may expect this to be the worst-case example as any deviation from this can only increase the chance of another vertex also joining $N_{\widetilde{H}}(S)$ while decreasing the chance of an original vertices to be in $N_{\widetilde{H}}(S)$ by a *lower* amount. Let us formalize this now.

We prove Eq (3.8) by showing that we can transform random variables $\{X_i\}_{i=1}^{r}$ into $\{Y_i\}_{i=1}^{r}$ through multiple iterations with intermediate variables $\left\{Z_i^j\right\}_{i=1}^{r}$ and degree sequences $\left\{d_i^j\right\}_{i=1}^{r}$ such that $\Pr\left(Z_i^j = 1\right) = \widetilde{q}^{d_i^j}$ always. Originally, we set $Z_i^1 = X_i$ and $d_i^1 = d_i$. We update Z_i^j's and d_i^j's in iteration j to $j+1$ as follows. Define two indices λ and γ:

$$\lambda \in \arg\min_i \left\{d_i^j : d_i^j > 0\right\};$$
$$\gamma \in \arg\max_i \left\{d_i^j : d_i^j < s\right\}.$$

We then define:

$$Z_\lambda^{j+1} : \Pr\left(Z_\lambda^{j+1}\right) = \widetilde{q}^{d_\lambda^j - 1} \qquad d_\lambda^{j+1} = d_\lambda^j - 1;$$
$$Z_\gamma^{j+1} : \Pr\left(Z_\gamma^{j+1}\right) = \widetilde{q}^{d_\gamma^j + 1} \qquad d_\gamma^{j+1} = d_\gamma^j + 1;$$
$$Z_i^{j+1} = Z_i^j \qquad d_i^{j+1} = d_i^j \qquad \text{for all } i \notin \{\lambda, \gamma\}$$

The intuition is that we are moving one edge of the base graph from vertex v_λ with "small" degree to S to vertex v_γ with a "larger" degree to S without violating any constraint in Eq (3.7).

Through this transformation, only the variables $Z_\lambda^{j+1}, Z_\gamma^{j+1}$ changed from Z_λ^j, Z_γ^j and the rest are the same. As these variables are binary, we only need to show that for any $t' \in \{1,2\}$,

$$\Pr\left(Z_\lambda^{j+1} + Z_\gamma^{j+1} \geq t'\right) \geq \Pr\left(Z_\lambda^j + Z_\gamma^j \geq t'\right).$$

For $t' = 1$,

$$\begin{aligned}
\Pr\left(Z_\lambda^{j+1} + Z_\gamma^{j+1} \geq 1\right) &= \widetilde{q}^{d_\lambda^j - 1} + \widetilde{q}^{d_\gamma^j + 1} - \widetilde{q}^{d_\lambda^j - 1 + d_\gamma^j + 1} = \widetilde{q}^{-1} \cdot \widetilde{q}^{d_\lambda^j} + \widetilde{q} \cdot \widetilde{q}^{d_\gamma^j} - \widetilde{q}^{d_\lambda^j + d_\gamma^j} \\
&\geq \widetilde{q}^{d_\lambda^j} + \widetilde{q}^{d_\gamma^j} - \widetilde{q}^{d_\lambda^j + d_\gamma^j} \qquad \text{(as } \widetilde{q}^{d_\lambda^j} \geq \widetilde{q}^{d_\gamma^j} \text{ by definition)} \\
&= \Pr\left(Z_\lambda^j + Z_\gamma^j \geq 1\right).
\end{aligned}$$

For $t' = 2$,

$$\Pr\left(Z_\lambda^{j+1} + Z_\gamma^{j+1} \geq 2\right) = \widetilde{q}^{d_\lambda^j - 1} \cdot \widetilde{q}^{d_\gamma^j + 1} = \widetilde{q}^{d_\lambda^j} \cdot \widetilde{q}^{d_\gamma^j} = \Pr\left(Z_\lambda^j + Z_\gamma^j \geq 2\right).$$

As such, we obtain that for every $t \geq 1$, $\Pr\left(Z^{j+1} \geq t\right) \geq \Pr\left(Z^j \geq t\right)$. We can thus continue this process iteration by iteration until we end up with variables $Z_i^j = Y_i$, hence proving Eq (3.8).

Let us now finalize the proof by showing that $\Pr(Y > r - s)$ is sufficiently small (this part is similar to the proof of Claim 3.3.18). Note that all Y_i's for $i > N$ are already 1 and hence there is nothing to do there. Let T be a subset of size $N - s + 1$ from $[N]$. We prove the probability that all Y_i's in T are also one is exponentially small so that no such T has this property. This implies that $Y \leq (r - N) + (N - s + 1) - 1 = r - s$ with high probability. In particular, for any such T,

$$\begin{aligned}
\Pr(\forall i \in T\ Y_i = 1) = \prod_{i \in T} \Pr(Y_i = 1) &= (1 - \widetilde{p})^{s \cdot (|T| - 1) + s - N/3} \qquad \text{(by definition of } Y_i\text{'s)} \\
&\leq \exp\Big(-\widetilde{p} \cdot (s \cdot (N - s + 1) - N/3)\Big)
\end{aligned}$$

$$\leq \exp\Big(-\widetilde{p}\cdot(s\cdot(N-s+1))\cdot 2/3\Big)$$

$$(s\cdot(N-s+1)\geq N \text{ for all } s=|S|\leq N)$$

$$\leq \exp\Big(-\frac{100\ln n}{N}\cdot\frac{8N}{15}\cdot(N-s+1)\Big)$$

(by the choice of $\widetilde{p}$ and $|S|\geq 4N/5$)

$$\leq \exp\Big(-50\ln n\cdot(N-s+1)\Big).$$

We can now take a union bound over all choices of T and obtain that,

$$\Pr\left(Y>r-s\right)\leq \sum_{\substack{T\subseteq[N]\\|T|=N-s+1}}\Pr\left(\forall i\in T\ Y_i=1\right)$$

$$\leq \exp\Big((N-s+1)\cdot\ln N\Big)\cdot\exp\Big(-50\ln n\cdot(N-s+1)\Big)$$

$$\leq n^{-48\cdot(N-s+1)}. \qquad (\text{as } N\leq n)$$

Combining this with Eq (3.8), we have that $\Pr\left(\left|N_{\widetilde{H}}(S)\right|<|S|\right)\leq n^{-48\cdot(N-|S|+1)}$ for a *fixed* S. To conclude, we take a union bound over all S by bundling them based on their size:

$$\Pr\left(\exists S:\left|N_{\widetilde{H}}(S)\right|<|S|\ ,\ |S|\geq 4N/5\right)\leq \sum_{\substack{S\subseteq L\\|S|\geq 4N/5}}\Pr\left(\left|N_{\widetilde{H}}(S)\right|<|S|\right)$$

$$\leq \sum_{k=4N/5}^{N}\binom{N}{N-k}\cdot n^{-48\cdot(N-k+1)}$$

$$\leq \sum_{k=4N/5}^{N} n^{-47\cdot(N-k+1)}\leq n^{-46}.$$

(by the equation above and since $\binom{N}{k}\leq N^k\leq n^k$ and $(N-k+1)\geq 1$ for all k)

This concludes the proof.

□

Lemma 3.3.17 now follows from Claim 3.3.18 and Claim 3.3.19 and Hall's Theorem in Fact 2.1.1.

3.4. Meta Algorithm

In this section, we use our palette sparsification theorem to design a "meta-algorithm" for $(\Delta+1)$ coloring, called ColoringAlgorithm. In the next section, we show how to implement this algorithm in each model of sublinear computation to obtain our final algorithms.

3.4.1. The Meta-Algorithm

Our meta-algorithm ColoringAlgorithm is as follows:

ColoringAlgorithm(G, Δ): A meta-algorithm for finding a $(\Delta+1)$-coloring in a graph $G(V, E)$ with maximum degree Δ.

1. Sample $\Theta(\log n)$ colors $L(v)$ uniformly at random for each vertex $v \in V$ (as in Theorem 3.5).
2. Define, for each color $c \in [\Delta+1]$, a set $\chi_c \subseteq V$ where $v \in \chi$ iff $c \in L(v)$.
3. Define E_{conflict} as the set of all edges (u, v) where both $u, v \in \chi_c$ for some $c \in [\Delta+1]$.
4. Construct the *conflict graph* $G_{\text{conflict}}(V, E_{\text{conflict}})$.
5. Find a proper list-coloring of $G_{\text{conflict}}(V, E_{\text{conflict}})$ with $L(v)$ being the color list of vertex $v \in V$.

We refer to ColoringAlgorithm as a "meta-algorithm" since constructing the conflict graph as well as finding its list-coloring are *unspecified* steps in ColoringAlgorithm. To implement this meta-algorithm in different models, we need to come up with an efficient way of performing these two tasks which are model-specific and are hence not fixed in ColoringAlgorithm. The following lemma establishes the main properties of ColoringAlgorithm.

Lemma 3.4.1. *Let $G(V, E)$ be a graph with maximum degree Δ. In* ColoringAlgorithm(G, Δ)*, with high probability:*

1. *The output is a valid $(\Delta+1)$ coloring of the graph G.*
2. *For any $c \in [\Delta+1]$, size of χ_c is $O(n \log n/\Delta)$.*

3. *The maximum degree in graph* G_{conflict} *is* $O(\log^2 n)$.

Proof. We show that each part holds with high probability. Taking a union bound over the three parts finalizes the proof.

1. We apply Theorem 3.5 to the sets $L(v)$ chosen for each $v \in V$, obtaining that with high probability, G can be list-colored with $L(v)$ being the list of vertex v. Now notice that since E_{conflict} contains all possible monochromatic edges that arise in any list-coloring of G with lists $L(\cdot)$, any proper list-coloring of G is a proper list-coloring of G_{conflict} and vice versa. As such, we know that G_{conflict} contains a proper list-coloring and this list-coloring is also a feasible $(\Delta+1)$ coloring of the graph G.

2. Fix any color $c \in [\Delta+1]$. Let K be the number of colors sampled by each vertex. The probability that any specific vertex v chooses c in $L(v)$ is $K/(\Delta+1)$. As such, the expected number of vertices in χ_c is $n \cdot K/(\Delta+1)$. As $K = \Theta(\log n)$ and the choice of $L(\cdot)$ is independent across all vertices, by Chernoff bound, the total number of vertices in χ_c is with high probability $2n \cdot K/\Delta = O(n \log n/\Delta)$. Taking a union bound on all $\Delta+1$ classes, finalizes the proof of this part.

3. Fix any vertex $v \in V$ and again let K be the number of colors sampled by each vertex. We fix these colors, say, $c_1, \ldots, c_K$. For any neighbor of v, say $u \in N(v)$ and $i \in [K]$, let $X_{u,i}$ be an indicator random variable which is one iff $c_i \in L(u)$. Let $X := \sum_{u \in N(v)} \sum_{i=1}^{K} X_{u,i}$ and thus $\mathbb{E}[X] = \Delta \cdot K \cdot K/(\Delta+1) \leq K^2$. Note that X is an upper bound on degree of v in G_{conflict}. As the $X_{u,i}$'s are negatively correlated, by Chernoff bound, we have that $X \leq O(K^2) = O(\log^2 n)$ with high probability. Taking a union bound on all n vertices finalizes the proof of this part.

Lemma 3.4.1 now follows from a union bound over the three events above. □

Lemma 3.4.1 is sufficient for the purpose of obtaining sublinear algorithms *if we do not care about the computation time of the resulting algorithm*, e.g., only aim to minimize the space

or the query complexity of the algorithm in streaming and the query model, respectively. Indeed, all one has to do is to construct the conflict-graph (which we show is easily doable in each model in subsequent sections) and then simply find a proper list-coloring of it using the chosen lists $L(\cdot)$.

Nevertheless, this approach on its own is not enough to obtain efficient algorithms in terms of computation time. Our palette sparsification theorem only implies the existence of the list-coloring in the conflict graph but list-coloring a graph with lists of size $O(\log n)$ is in general an NP-hard problem and a-priori it is not clear how can one we color the conflict-graph efficiently. To remedy this, we show that our palette sparsification theorem can be made algorithmic and use this to obtain an efficient algorithm for list-coloring the conflict-graph. In the following sections, we give a meta algorithm that computes a sparse-dense decomposition, and gives an algorithm for list-coloring the conflict-graph given a sparse-dense decomposition.

3.4.2. Meta Algorithm for Finding a sparse-dense decomposition

In this section, we give a meta algorithm for computing a sparse-dense decompostion, and we will show how to implement it in the next section. Although Lemma 3.2.1 only proves the existance of the decompostion, the proof itself gives a way to construction the decomposition. In fact, the only information we need is the number of common neighbors between each pair of vertices. Although it is hard to get the exact number in sublinear settings, we show that getting an approximation of the number of common neighbors is also enough to construct a sparse-dense decomposition with slightly worse parameters.

Our meta-algorithm DecompositionAlgorithm is as follows:

DecompositionAlgorithm(G, ε): A meta-algorithm for finding a a sparse-dense decompostion given a parameter $\varepsilon < 1/200$.

1. Uniformly sample $\Theta(n \log n/\Delta)$ vertices, and find all of their neighbors.

2. For any pair of vertices (u, v), compute $\mathsf{Com}(u, v)$, an approximation of the number of common neighbors between them.
3. For any vertes v, compute $T(v)$, an approximate of the number of neighbors u such that $\mathsf{Com}(u, v) \geq (1 - 2\varepsilon)\Delta$.
4. Construct a set V' that contains all vertex v such that $T(v) \geq (1 - 2\varepsilon)\Delta$.
5. For each vertex $v \in V'$, construct a set S_v that contains all vertices u such that $\mathsf{Com}(u, v) \geq (1 - 7\varepsilon)\Delta$.
6. Construct a graph G': for any two vertices u and v, there is an edge between u and v if and only if $v \in S_u$ or $u \in S_v$.
7. Output $C_1, \ldots, C_k$, the connected components of size at least 2 in G', and V_{sparse}, the set of isolate vertices in G'.

We first prove that we can get good approximation in Step 2 and Step 3 given the information obtained from step 1.

Claim 3.4.2. *In step 2, we can compute* $\mathsf{Com}(u, v)$ *to be an approximateion of the number of common neighbors between* u *and* v *to within an additive error of* $\varepsilon\Delta$ *for any pair of vertices* u *and* v *with high probability. In step 3, we can compute* $T(v)$ *to be an approximation of the number of neighbors* u *such that* $\mathsf{Com}(u, v) \geq (1 - 2\varepsilon)\Delta$ *to within an additive error of* $\varepsilon\Delta$ *with high probability.*

Proof. Let V_S be the vertices sampled in step 1. For any pair of vertices u and v, we approximate the number of common neighbors by the number of common neighbors in V_S. Since $|V_S| = \Theta(n \log n/\Delta)$ and there are n vertices in total, by Chernoff bound, we can approximate the number of common neighbors to within an additive error $\varepsilon\Delta$ with high probability. The first half of the claim is proved by taking a union bound on all $\binom{n}{2}$ pair of vertics.

The second half of the claim can be proved similarly. For any vertex v, we approximate

the number of vertices u such that $u \in N(v)$ and $\mathsf{Com}(u,v) \geq (1-2\varepsilon)\Delta$ by computing the number of such vertices in V_S. Again, since $|V_S| = \Theta(n \log n/\Delta)$ and there are n vertices in total, by Chernoff bound, we can approximate the number of such u to within an additive error $\varepsilon\Delta$ with high probability. Taking a union bound on all n vertices finalizes the proof of the second part. □

By Claim 3.4.2, the only unspecified step in DecompositionAlgorithm is the first step, and we will implement this step in different models. The following lemma shows that the output of DecompositionAlgorithm has similar propoties to the decompostion in Lemma 3.2.1. The prove of the lemma is similar to the prove of Lemma 3.2.1 given Claim 3.4.2.

Lemma 3.4.3. *For any $\varepsilon < 1/200$ and $G = (V, E)$ be any arbitrary graph. Let $\varepsilon_1 = 4\varepsilon$, the output of* DecompositionAlgorithm(G, ε) *has the following propoties:*

1. *For every vertex $v \in V_{sparse}$, the total number of edges between the neighbors of v is at most $(1-\varepsilon^2) \cdot \Delta^2/2$;*
2. *Any set of vertices C_i, called an almost-clique, has the following properties:*
   (a) *$(1-\varepsilon_1)\Delta \leq |C_i| \leq (1+3\varepsilon_1)\Delta$;*
   (b) *Any vertex $v \in C_i$ has at most $3\varepsilon_1\Delta$ neighbors outside of C_i;*
   (c) *Any vertex $v \in C_i$ has at most $6\varepsilon_1\Delta$ non-neighbors inside of C_i.*

Proof. By Claim 3.4.2, for any pair of vertices u and v such that $\mathsf{Com}(u,v) \leq (1-2\varepsilon)\Delta$, we have $|N(u) \cap N(v)| \leq (1-\varepsilon)\Delta$. Moreover, for any $v \notin V'$, there are at least $\varepsilon\Delta$ neighbors of v such that $\mathsf{Com}(u,v) \leq (1-2\varepsilon)\Delta$. Thus, for any $v \notin V'$, the total number of edges between the neighbors of v is $(1-\varepsilon^2) \cdot \Delta^2/2$.

Similarly, for any $v \in V'$, there are at least $(1-3\varepsilon)\Delta$ neighbor $u \in N(v)$ such that $|N(u) \cap N(v)| \geq (1-3\varepsilon)\Delta$. On the other hand, for any u such that $|N(u) \cap N(v)| \geq (1-6\varepsilon)\Delta$, we have $u \in S_v$. Also for any $u \in S_v$, we have $|N(u) \cap N(v)| \geq (1-8\varepsilon)\Delta = (1-2\varepsilon_1)\Delta$.

By the same proof as Claim 3.2.2, if a pair of vertices $u, v \in V'$ has at least one common neighbor in G', then $(u, v) \in G'$. The remaining arguments of the proof is the same as the proof of Lemma 3.2.1 after we get Claim 3.2.2. □

To make the proof of Theorem 3.5 work using the sparse-dense decomposition given by Lemma 3.4.3 instead of Lemma 3.2.1, we just need to work on ε_1 instead of ε when coloring almost cliques.

3.4.3. Algorithmic Palette Sparsification

In the following, we show that the proof of Theorem 3.5 can be tweaked slightly to turn it into an efficient algorithm for list-coloring the conflict-graph assuming we are also given the sparse-dense decomposition of the original graph G given by DecompositionAlgorithm. In particular, we prove the following lemma.

Lemma 3.4.4. *Let $G(V, E)$ be a graph with maximum degree Δ. There exists an algorithm that given the conflict-graph G_{conflict} with lists $L(v)$ for each $v \in V$ constructed by* ColoringAlgorithm(G, Δ) *plus the sparse-dense decomposition of the graph G given by* DecompositionAlgorithm*, outputs a list-coloring of the graph G_{conflict} with lists $L(\cdot)$ with high probability in $\widetilde{O}(n\sqrt{\Delta})$ time.*

Proof. Let $V_{sparse}, C_1, C_2, \ldots, C_k$ be the spare-dense decomposition of G as in Lemma 3.4.3. The proof consists of three phases, which correspond to the three phases in the proof of Theorem 3.5.

Coloring Sparse Vertices. We use the process given by the proof of Lemma 3.3.1 to color the sparse vertices in V_{sparse}. We run OneShotColoring and GreedyColor with $O(\log n)$ rounds. In OneShotColoring and each round of GC, each vertex v in V_{sparse} which have not been colored picks a color in $L'(v)$. Then we check if the color is different from all the vertices in $N(v)$. If so, we color v by the chosen color. So we only need to iterate over the edges in G_{conflict}, which takes $O(n \log^2 n)$ time. Hence, this phase of the algorithm takes $O(n \log^3 n)$ time in total.

Initial Coloring of Almost Cliques. In this phase, for each almost-clique C_i in the decomposition, we find a colorful matching (as in Definition 3.3.9). We run ColorfulMatching. For each color, if we can find a pair of vertices u, v such that (u, v) is not in G_{conflict}, u and v are not in the colorful matching yet, and $L(u)$ and $L(v)$ both contain the same color, then we add (u, v) with this color to the colorful matching. Hence, this phase also takes $\widetilde{O}(n)$ time.

Final Coloring of Almost-Cliques. In this phase, we color the remaining vertices inside almost-cliques. To color these vertices, we color the almost-cliques one by one. When coloring almost-clique C_i, we construct the palette-graph (as in Definition 3.3.15) between the vertices and the colors, and find a maximum matching of this graph. By Claim 3.3.16 and Lemma 3.3.17 there is a matching that pairs each vertex to an available color. To construct the *palette-graph*, we need to connect each vertex v with all colors in $L(v)$. Then we iterate over the edges of G_{conflict}, delete the edges between a vertex and an unavailable color. The construction of the palette-graph for one almost-clique takes $O(\Delta \log^2 n)$ time. Finding the matching also require $O(\Delta^{3/2})$ time by using the standard Hopcraft-Karp algorithm [152] for bipartite matching. There are at most $O(n/\Delta)$ near-cliques in G, so this phase takes at most $\widetilde{O}(n\sqrt{\Delta})$ time with high probability. □

3.4.4. A Faster Algorithm for List-Coloring the Conflict Graph

Finally, we show that the post-processing step in Lemma 3.4.4 for coloring the conflict-graph can be done in near-linear time in size of the conflict-graph, which results in an $\widetilde{O}(n)$ time algorithm. As is evident from the proof of Lemma 3.4.4, in order to obtain such an algorithm, we only need an algorithm for finding a maximum matching in the palette-graph in near-linear time (the rest of the algorithm is linear-time already). While obtaining a near-linear time algorithm for matching in general is a long standing open problem, we are helped with the fact here that palette-graph is almost a random graph (as was exploited crucially in the proofs of Lemma 3.3.17). In particular, it is known that Hopcroft-Karp algorithm [152] for bipartite matching runs in near-linear time on random bipartite graphs [40, 211]. In

the following lemma, we build on the results of [40, 211] to prove a similar result for the palette-graphs which are almost-random graphs.

Lemma 3.4.5. *Let C_i be any almost-clique in G and χ_3 be the partial coloring obtained after processing almost-cliques $C_1, \ldots, C_{i-1}$. With high probability (over the randomness of the third batch), a maximum matching of the palette-graph H_i of C_i can be found in $\widetilde{O}(n)$ time.*

The proof of this lemma, similar to [40, 211] is by showing that the underlying graph has good expansion properties and then use the fact that Hopcroft-Karp algorithms runs faster on expanders.

Lemma 3.4.6. *Let $H(L, R, E)$ be a graph such that for some $N \leq n$:*

(i) $|L| = N$ and $|R| \leq 2N$;

(ii) minimum degree of vertices in L is at least $9N/10$;

(iii) for any $S \subseteq L$ of size $|S| \geq 4N/5$, $\sum_{v \in S} \mathsf{deg}_H(v) \geq (|S| \cdot N) - N/3$

Suppose $\widetilde{H}(L, R, \widetilde{E})$ is obtained from H by sampling each edge of E w.p. $\widetilde{p} \geq \frac{400 \ln n}{N}$. Then, with probability at least $1 - 1/n^5$, $\widetilde{H}$ contains a matching that matches all vertices in L. Furthermore, such a matching can be found in $\tilde{O}(N)$ time.

The proof of Lemma 3.4.6 is similar to the proof of Lemma 3.3.17, except that we also prove expansion properties of the graph, and use the idea in [40, 211] to prove the fast running time.

We first prove that for any subset $S \subseteq L$ and $T \subseteq R$ that are not "too large", the sizes of their neighbor sets are much larger than the size of themselves. Specifically, let $\lambda = 1.05$, for any $S \subseteq L$ with size at most $0.82N$, $N_{\widetilde{H}}(S) \geq \lambda |S|$; for any $T \subseteq R$ with size $|R| - |L| < |T| \leq |R| - |L| + 0.18N$, $N_{\widetilde{H}}(T) \geq \lambda(|T| - (|R| - |L|))$.

The following claim proves the expansion of any small subset of L. The proof is similar to

the proof of Claim 3.3.18, we give the proof in Section 3.4.5 for completeness.

Claim 3.4.7. *W.h.p., for every set $S \subseteq L$ of size $|S| \leq 4N/5$, $\left|N_{\widetilde{H}}(S)\right| \geq \lambda |S|$.*

We then prove the expansion of the subsets of R. The proof of this part is similar to the proof of Claim 3.4.8. We give the proof in Section 3.4.6 for completeness.

Claim 3.4.8. *W.h.p., for every set $T \subseteq R$ of size $|R| - |L| < |T| \leq |R| - |L| + 0.18N$, $\left|N_{\widetilde{H}}(T)\right| \geq \lambda(|T| - |R| + |L|)$.*

Claim 3.4.7 and Claim 3.4.8 immediately imply that $\widetilde{H}$ conatins a left-saturating matching by Hall's Theorem in Fact 2.1.1. We use Hopcroft-Karp algorithm [152] to find such a matching. To prove the algorithm runs in $\tilde{O}(N)$ time, we use Claim 3.4.7 and Claim 3.4.8 to argue that the lengths in of augmenting paths throughout the running of algorithm is $\tilde{O}(1)$, which means the algorithm only takes $\tilde{O}(1)$ iterations.

Proof of Lemma 3.4.6. Since H has $O(N^2)$ edges and $\widetilde{p} = O(\log n/N)$, the number of edges in $\widetilde{H}$ is $\tilde{O}(N)$ with high probability. From this point, we assume $\widetilde{H}$ has $\tilde{O}(N)$ edges and the high probability events in Claim 3.4.7 and Claim 3.4.8 are both true.

To prove that Hopcroft-Karp algorithm finds a left-saturating matching in $\tilde{O}(N)$ time, it is sufficient to prove that for any partial matching with size less than N, there exists an augmenting path of length $\tilde{O}(1)$ in the residue graph.

Fix an arbitrary matching M which does not match all vertices in L. For any set $S \subseteq L$ and $T \subseteq R$ which only contain matched vertices, denote $M(S)$ and $M(T)$ as the set of vertices match to them.

Let S_0 be the set of unmatched vertices in L, and for any integer $i > 0$, S_i be the set of vertices that can be reached within $2i$ steps in the residue graph. For any i, if $|S_i| \leq 0.82N$, by Claim 3.4.7, $\left|N_{\widetilde{H}}(S_i)\right| \geq \lambda |S_i|$. If $N_{\widetilde{H}}(S_i)$ contains no unmatched vertices in R, then $M(N_{\widetilde{H}}(S_i))$ contains the vertices that are either in S_i or can be reached by the vertices in S_i

in two steps in the residue graph. So $|S_{i+1}| \geq \lambda |S_i|$. Let k be the smallest integer such that $\lambda^k > 0.82N$. One of the following two events must happen: 1. there is an unmatched vertices in R that can be reached by S_0 within $2k$ steps; 2. $S_k > 0.82N$. If the first event happen, then there is an augmenting path of length at most $2k \leq 2(\lfloor \log_\lambda 0.82N \rfloor + 1) = O(\log N) = \tilde{O}(1)$.

If the first event does not happen, there is no augmenting path of length at most $2k$. Let T_0 be the set of unmatched vertices in R. Since the size of M is less than N, $|T| > |R| - |L|$. Similarly, for any $j > 0$, let T_j be the set of vertices in R that can reach T_0 within $2j$ steps in the residue graph. For any j such that $T_j \leq 0.18N + |R| - |L|$ by Claim 3.4.8, $\left|N_{\widetilde{H}}(T_j)\right| \geq \lambda(|T_j| - |R| + |L|)$. If $N_{\widetilde{H}}(T_j)$ contains no unmatched vertices in L, then $M(N_{\widetilde{H}}(T_j))$ contains the matched vertices that are either in T_j or can reach the vertices in T_j in two steps in the residue graph. So $|T_{j+1}| \geq \left|M(N_{\widetilde{H}}(T_j))\right| + |T_0| \geq \lambda(|T_j| - |R| + |L|) + |R| - |L|$. Let ℓ be the smallest integer such that $\lambda^\ell > 0.18N$. Since $l < k$, for any $j \leq l$, T_j can not be reached by any unmatched vertex in the residue graph. So $\left|N_{\widetilde{H}}(T_{\ell-1})\right| > 0.18N$, which means there is a common vertex that is in both S_k and $N_{\widetilde{H}}(T_{\ell-1})$. Let v be such a vertex, then S_0 can reach v within $2k$ steps, and v can reach T_0 within $2\ell - 1$ steps in the residue graph, which means there is an augmenting path of size at most $2k + 2\ell - 1 = O(\log N) = \tilde{O}(1)$.

We proved that in any residue graph, there exists an augmenting path of length $\tilde{O}(1)$. This means Hopcroft-Karp algorithm takes $\tilde{O}(1)$ iterations. Since the number of edges in $\widetilde{H}$ is $\tilde{O}(N)$, each iteration takes $\tilde{O}(N)$ time. So the running time of Hopcroft-Karp algorithm is $\tilde{O}(N)$.

□

Lemma 3.4.5 immediately follows from Claim 3.3.16 and Lemma 3.4.6.

3.4.5. Proof of Claim 3.4.7

Proof. Fix any choice of set $S \subseteq L$ with $|S| \leq 4N/5$ and any set $T \subseteq R$ with $|T| = \lceil \lambda |S| \rceil - 1$; we also define $\overline{T} := R \setminus T$. We prove that there exists an edge in $\widetilde{H}$ between S and $\overline{T}$, thus ensuring that $N_{\widetilde{H}}(S)$ is not a subset of T; moreover, this happens with such a high probability

that we can take a union bound over all choices for both S and T.

Let $E(S,\overline{T})$ denote the set of edges between S and $\overline{T}$ in H (the base graph). We have,

$$\left|E(S,\overline{T})\right| \geq \sum_{v\in S}(\mathsf{deg}_H(v) - |T|) \geq \sum_{v\in S}(9N/10 - 1.05\cdot 0.82N) > |S|\cdot N/40.$$

(by Item (ii) $\mathsf{deg}_H(v) \geq 9N/10$ and since $|T| = \lceil \lambda\,|S|\rceil - 1 \leq \lambda\,|S| \leq 1.05\cdot 0.82N$)

Using this, we can calculate,

$$\begin{aligned}\Pr\left(\widetilde{E}\cap E(S,\overline{T}) = \emptyset\right) &= (1-\widetilde{p})^{|E(S,\overline{T})|}\\ &\leq \exp\left(-\frac{400\log n}{N}\cdot|S|\cdot N/40\right) = \exp\left(-|S|\cdot 10\ln n\right).\end{aligned}$$

(by definition of $\widetilde{p}$ and equation above)

We now take a union bound over all choices of T and obtain that,

$$\Pr\left(\left|N_{\widetilde{H}}(S)\right| < \lambda\,|S|\right) \leq \sum_{\substack{T\subseteq R\\ |T|=\lceil\lambda|S|\rceil-1}} \Pr\left(\widetilde{E}\cap E(S,\overline{T}) = \emptyset\right) \leq (2N)^{\lambda|S|}\cdot\exp\left(-|S|\cdot 10\ln n\right)$$

(as $|R| \leq 2N$ by Item (i) and $|T| \leq \lambda\,|S|$ by definition)

$$\leq \exp\left(\lambda\,|S|\cdot\ln n + \lambda\,|S| - |S|\cdot 10\log n\right) \ll n^{-8\cdot|S|}. \qquad \text{(as } N\leq n\text{)}$$

We can now also take a union bound over all S by bundling them based on their size:

$$\begin{aligned}\Pr\left(\exists S : \left|N_{\widetilde{H}}(S)\right| < \lambda\,|S|\ ,\ |S| < 4N/5\right) &\leq \sum_{\substack{S\subseteq L\\ |S|<4N/5}} \Pr\left(\left|N_{\widetilde{H}}(S)\right| < \lambda\,|S|\right)\\ &\leq \sum_{k=1}^{4N/5}\binom{N}{k}\cdot n^{-8k} \leq \sum_{k=1}^{4N/5} n^{-7k} \leq n^{-6}.\end{aligned}$$

(by the equation above and since $\binom{N}{k} \leq N^k \leq n^k$)

As such, for all sets S considered in this case, with high probability, $\left|N_{\widetilde{H}}(S)\right| \geq \lambda\,|S|$.

□

3.4.6. Proof of Claim 3.4.8

Proof. To prove the claim, it is sufficient to prove that for any integer $0 < t \leq 0.18N$, and for any $T \subseteq R$ with size $|T| = t + |R| - |L|$, $N_{\widetilde{H}}(T) \geq \lambda t$. Let $s = \lfloor |L| - \lambda t \rfloor + 1$. We prove that for any $S \subseteq L$ with size $|S| = s$, $N_{\widetilde{H}}(S) > N - t$, which means for any $S \subseteq L$ with $|S| = s$ and $T \subseteq R$ with $|T| = t + |R| - |L|$, there is at least one edge between S and T in $\widetilde{H}$. So for any $T \subseteq R$ with $|T| = t + |R| - |L|$, $N_{\widetilde{H}}(T) \geq N - s + 1 \geq \lambda k$.

Fix any choice of set $S \subseteq L$ with $|S| = s$. Since $t \leq 0.18N$ and $\lambda = 1.05$, $s = \lfloor |L| - \lambda t \rfloor + 1 \geq N - \lambda t > 4N/5$. Recall that by Item (iii), we have,

$$\sum_{v \in S} \mathsf{deg}_H(v) \geq (s \cdot N) - N/3.$$

For simplicity, in this proof, we *arbitrarily* remove incident edges on vertices of S such that the total number of edges incident on S becomes *exactly* the RHS above.

Let $r := |R|$, and define r indicator random variables $X_1, \ldots, X_r$ where $X_i = 1$ iff $v_i \in R$ does *not* belong to $N_{\widetilde{H}}(S)$. Additionally, define d_i as the number of edges that $v_i \in R$ has to vertices in S. Let $X := \sum_{i=1}^{r} X_i$ and note that $\left|N_{\widetilde{H}}(S)\right| = r - X$. Hence, our goal is to compute $\Pr(X \geq r - N + t)$. Let us define $\widetilde{q} := (1 - \widetilde{p})$. We have the following properties for X_i's and d_i's (the last equality is by the modification above):

$$\begin{aligned} &\forall\ i \in [r] \qquad \Pr(X_i = 1) = \widetilde{q}^{d_i}; \\ &\forall\ i \in [r] \qquad d_i \leq s; \\ &\sum_{i=1}^{r} d_i = (s \cdot N) - N/3. \end{aligned} \tag{3.9}$$

Working with variables $X_1, \ldots, X_r$ directly is quite cumbersome and we instead define the random variables $Y_1, \ldots, Y_r$ where:

$$\forall\ i \in [N-1] \qquad \Pr(Y_i = 1) = \widetilde{q}^{s};$$

$$\Pr(Y_N = 1) = \widetilde{q}^{s-N/3};$$
$$\forall\, i \in [N+1:r] \qquad \Pr(Y_i = 1) = 1.$$

For some intuition, notice that random variables Y_i would be equal to X_i's *if* the base graph H is such that there are $N-1$ vertices with degree s to S, one vertex with degree $s - N/3$, and all other vertices in R have degree zero to S (such a choice satisfies Eq (3.9)). In the following, we first prove that such a base graph H is the "worst case example" in proving $\left|N_{\widetilde{H}}(S)\right| > N - t$ and then focus on this case directly. More formally, we prove that for $Y = \sum_{i=1}^{r} Y_i$ and every $k \geq 1$,

$$\Pr(Y \geq k) \geq \Pr(X \geq k). \tag{3.10}$$

Again for intuition, notice that in defining Y's and corresponding worst-case base graph H, we simply moved degrees of all vertices in R to a *minimal* set of N vertices while satisfying the degree requirements of Eq (3.9); one may expect this to be the worst-case example as any deviation from this can only increase the chance of another vertex also joining $N_{\widetilde{H}}(S)$ while decreasing the chance of an original vertices to be in $N_{\widetilde{H}}(S)$ by a *lower* amount. Let us formalize this now.

We prove Eq (3.10) by showing that we can transform random variables $\{X_i\}_{i=1}^{r}$ into $\{Y_i\}_{i=1}^{r}$ through multiple iterations with intermediate variables $\left\{Z_i^j\right\}_{i=1}^{r}$ and degree sequences $\left\{d_i^j\right\}_{i=1}^{r}$ such that $\Pr\left(Z_i^j = 1\right) = \widetilde{q}^{d_i^j}$ always. Originally, we set $Z_i^1 = X_i$ and $d_i^1 = d_i$. We update Z_i^j's and d_i^j's in iteration j to $j+1$ as follows. Define two indices λ and γ:

$$\lambda \in \arg\min_i \left\{d_i^j : d_i^j > 0\right\};$$
$$\gamma \in \arg\max_i \left\{d_i^j : d_i^j < s\right\}.$$

We then define:

$$
\begin{aligned}
&Z_\lambda^{j+1} : \Pr\left(Z_\lambda^{j+1}\right) = \widetilde{q}^{d_\lambda^j - 1} \qquad d_\lambda^{j+1} = d_\lambda^j - 1; \\
&Z_\gamma^{j+1} : \Pr\left(Z_\gamma^{j+1}\right) = \widetilde{q}^{d_\gamma^j + 1} \qquad d_\gamma^{j+1} = d_\gamma^j + 1; \\
&Z_i^{j+1} = Z_i^j \qquad d_i^{j+1} = d_i^j \qquad \text{for all } i \notin \{\lambda, \gamma\}
\end{aligned}
$$

The intuition is that we are moving one edge of the base graph from vertex v_λ with "small" degree to S to vertex v_γ with a "larger" degree to S without violating any constraint in Eq (3.9).

Through this transformation, only the variables $Z_\lambda^{j+1}, Z_\gamma^{j+1}$ changed from Z_λ^j, Z_γ^j and the rest are the same. As these variables are binary, we only need to show that for any $k' \in \{1, 2\}$,

$$
\Pr\left(Z_\lambda^{j+1} + Z_\gamma^{j+1} \geq k'\right) \geq \Pr\left(Z_\lambda^j + Z_\gamma^j \geq k'\right).
$$

For $k' = 1$,

$$
\begin{aligned}
\Pr\left(Z_\lambda^{j+1} + Z_\gamma^{j+1} \geq 1\right) &= \widetilde{q}^{d_\lambda^j - 1} + \widetilde{q}^{d_\gamma^j + 1} - \widetilde{q}^{d_\lambda^j - 1 + d_\gamma^j + 1} = \widetilde{q}^{-1} \cdot \widetilde{q}^{d_\lambda^j} + \widetilde{q} \cdot \widetilde{q}^{d_\gamma^j} - \widetilde{q}^{d_\lambda^j + d_\gamma^j} \\
&\geq \widetilde{q}^{d_\lambda^j} + \widetilde{q}^{d_\gamma^j} - \widetilde{q}^{d_\lambda^j + d_\gamma^j} \qquad \text{(as } \widetilde{q}^{d_\lambda^j} \geq \widetilde{q}^{d_\gamma^j} \text{ by definition)} \\
&= \Pr\left(Z_\lambda^j + Z_\gamma^j \geq 1\right).
\end{aligned}
$$

For $k' = 2$,

$$
\Pr\left(Z_\lambda^{j+1} + Z_\gamma^{j+1} \geq 2\right) = \widetilde{q}^{d_\lambda^j - 1} \cdot \widetilde{q}^{d_\gamma^j + 1} = \widetilde{q}^{d_\lambda^j} \cdot \widetilde{q}^{d_\gamma^j} = \Pr\left(Z_\lambda^j + Z_\gamma^j \geq 2\right).
$$

As such, we obtain that for every $k \geq 1$, $\Pr\left(Z^{j+1} \geq k\right) \geq \Pr\left(Z^j \geq k\right)$. We can thus continue this process iteration by iteration until we end up with variables $Z_i^j = Y_i$, hence proving Eq (3.10).

Let us now finalize the proof by showing that $\Pr(Y \geq r - N + t)$ is sufficiently small (this part is similar to the proof of Claim 3.3.18). Note that all Y_i's for $i > N$ are already 1 and hence there is nothing to do there. Let I be a subset of size t from $[N]$. We prove the probability that all Y_i's in I are also one is exponentially small so that no such I has this property. This implies that $Y \leq (r - N) + t - 1 < r - N + t$ with high probability. In particular, for any such I,

$$\begin{aligned}
\Pr(\forall i \in I\ Y_i = 1) &= \prod_{i \in I} \Pr(Y_i = 1) = (1 - \widetilde{p})^{s \cdot (|I| - 1) + s - N/3} && \text{(by definition of } Y_i\text{'s)} \\
&\leq \exp\Big(- \widetilde{p} \cdot (s \cdot t - N/3)\Big) \\
&\leq \exp\left(-\widetilde{p} \cdot ((4N/5) \cdot t - N/3)\right) && (s > 4N/5) \\
&= \exp\left(-\widetilde{p} \cdot ((4N/5) \cdot (t - 5/12))\right) \\
&\leq \exp\Big(- \widetilde{p} \cdot (N \cdot t \cdot 7/15)\Big) && (t \geq 1) \\
&\leq \exp\left(- \frac{400 \ln n}{N} \cdot \frac{7N}{15} \cdot t\right) && \text{(by the choice of } \widetilde{p}) \\
&\leq \exp\Big(- 150 \ln n \cdot t\Big).
\end{aligned}$$

We can now take a union bound over all choices of I and obtain that,

$$\begin{aligned}
\Pr(Y \geq r - N + t) &\leq \sum_{\substack{I \subseteq [N] \\ |I| = t}} \Pr(\forall i \in I\ Y_i = 1) \\
&\leq \exp\Big(t \cdot \ln N\Big) \cdot \exp\Big(- 150 \ln n \cdot t\Big) \\
&\leq n^{-149t}. && (\text{as } N \leq n)
\end{aligned}$$

Combining this with Eq (3.10), we have that $\Pr\left(\left|N_{\widetilde{H}}(S)\right| \leq N - t\right) \leq n^{-149t}$ for a *fixed* S. To conclude, we take a union bound over all S with size s:

$$\Pr\left(\exists S : \left|N_{\widetilde{H}}(S)\right| \leq N - t\ ,\ |S| = s\right) \leq \sum_{\substack{S \subseteq L \\ |S| = s}} \Pr\left(\left|N_{\widetilde{H}}(S)\right| \leq N - t\right)$$

$$
\begin{aligned}
&\leq \binom{N}{s} \cdot n^{-149t} \\
&\leq n^{-149t+N-s} && (\text{since } \tbinom{N}{s} \leq N^{N-s} \leq n^{n-s}) \\
&\leq n^{-146} && (\text{since } s = \lfloor N - \lambda t \rfloor + 1 \text{ and } t \geq 1)
\end{aligned}
$$

This menas with probability n^{-146}, for any $S \subseteq L$ with $|S| = s$, $N_{\widetilde{H}}(S) > N - t$. In other words, for any $T \subseteq R$ with $|T| = t + r - N$, $N_{\widetilde{H}}(T) \geq N - s + 1 \geq \lambda t$. Finally, we take a union bound over all t with $1 \leq t \leq 0.18N$:

$$
\Pr\left(\exists T : \left|N_{\widetilde{H}}(T)\right| < \lambda(|T| - r + N) \ , \ r - N < |T| \leq r - N + 0.18N\right) \leq n^{-145}
$$

as $N \leq n$. This concludes the proof.

□

3.5. Sublinear Algorithms for $(\Delta + 1)$ Coloring

We now use our palette sparsification theorem to design sublinear algorithms for $(\Delta + 1)$ coloring in different models of computation, formalizing Theorem 3.2, Theorem 3.3, and 3.4.

3.5.1. A Single-Pass Streaming Algorithm for $(\Delta + 1)$ Coloring

We first give an application of our palette sparsification theorem in designing a dynamic streaming algorithm for the $(\Delta+1)$ coloring problem. In the dynamic streaming model, the input graph is presented as an arbitrary sequence of edge insertions and deletions and the goal is to analyze properties of the resulting graph using memory that is sublinear in the input size, which is proportional to the number of edges in the graph. We are particularly interested in algorithms that use $O(n \cdot \text{polylog}(n))$ space, referred to as *semi-streaming* algorithms.

Theorem 3.6. *There exists a randomized single-pass semi-streaming algorithm that given a graph G with maximum degree Δ presented in a dynamic stream, with high probability finds*

a $(\Delta+1)$ *coloring of* G *with using* $\widetilde{O}(n)$ *space and* $\widetilde{O}(n)$ *time*[5].

We prove Theorem 3.6 by implementing ColoringAlgorithm and DecompositionAlgorithm in dynamic streams. We first implement ColoringAlgorithm. Recall that implementing ColoringAlgorithm requires us to specify (i) how we construct the conflict-graph, and (ii) how we find a list-coloring in this conflict graph using the lists $L(\cdot)$. Throughout the proof, we condition on the high probability event in Lemma 3.4.1. We first show how to construct the conflict-graph. To do so, we rely on the by now standard primitive of ℓ_0-samplers for sampling elements in dynamic streams (see, e.g. [121, 165, 173]) captured in the following proposition.

Proposition 3.5.1 (ℓ_0-samplers; cf. [165, 203])**.** *There exists an streaming algorithm that given a subset* $P \subseteq V \times V$ *of pairs of vertices and an integer* $k \geq 1$ *at the beginning of a dynamic stream, outputs with high probability a set* S *of* k *edges from the edges in* P *that appear in the final graph (it outputs all edges if their number is smaller than* k*). The set* S *of edges can be either chosen uniformly at random with replacement or without replacement. The space of algorithm is* $O(k \cdot \log^3 n)$.

Using Proposition 3.5.1, we show how to construct the conflict graph in the streaming model.

Lemma 3.5.2. $G_{\mathsf{conflict}}(V, E_{\mathsf{conflict}})$ *can be constructed in* $\widetilde{O}(n)$ *space and polynomial time in dynamic streams with high probability.*

Proof. We construct the sets $\chi_1, \ldots, \chi_{\Delta+1}$ and store the sets in $O(n \log n)$ space. For any vertex $v \in V$, we define the set P_v of all edge slots between v and $\bigcup_{c \in L(v)} \chi_c$, i.e., all vertices that may have an edge to v in E_{conflict}, and run the algorithm in Proposition 3.5.1 with $P = P_v$ and parameter $k = O(\log^2(n))$.

As we conditioned on the event in Lemma 3.4.1, the degree of each vertex is at most k in G_{conflict}. Hence, by Proposition 3.5.1, with high probability, we find all neighbors of this vertex in G_{conflict}. Taking a union bound over all vertices in V, with high probability, we

[5]Here Δ is the maximum degree of the graph at the end of the stream and we assume no upper bound on degree of vertices throughout the stream, which can be as large as $\Omega(n)$ even when Δ is much smaller.

can construct the graph G_{conflict}. As all these steps can be implemented in polynomial time, we obtain the final result. □

As we argued before, Lemma 3.5.2 together with Lemma 3.4.1 are already enough to achieve a semi-streaming algorithm with exponential-time (i.e., prove Theorem 3.6 if we do not want a polynomial time algorithm): after constructing the conflict-graph G_{conflict}, we can simply use an exponential time algorithm to find a proper list-coloring of G_{conflict} which would be a $(\Delta+1)$ coloring of G by Lemma 3.4.1.

To obtain a polynomial time algorithm, we only need to implement DecompositionAlgorithm in dynamic streams. By Claim 3.4.2, it is sufficient to implement the first step of DecompositionAlgorithm in dynamic streams. To do so, we sample $\Theta(n \log n/\Delta)$ vertices, and for each sampled vertices, we use Δ ℓ_0 sampler given by Proposition 3.5.1 to obtain all edges incident on them. The space we use is $\Theta((n \log n/\Delta) \cdot \Delta \cdot \log^3 n) = \widetilde{O}(n)$. And this completes the algorithm for Theorem 3.6. The time complexity of the algorithm follows from Lemma 3.4.5.

Removing the Assumption on Knowledge of Δ

The semi-streaming algorithm we described so far assumes the knowledge of parameter Δ beforehand. A potential criticism to this assumption is that such an algorithm is not "truly single-pass" as it require further knowledge about the graph than is given typically in the streaming model. In the following, we show that this assumption can be easily avoided at a cost of an extra $O(\log n)$ factor in the space complexity of the algorithm.

Firstly, we use $O(n)$ space during the stream to track the degree of every vertex. This allows us to compute Δ precisely by the end of the stream. Next, in parallel, we run the following algorithm for $O(\log n)$ choices of $\beta = 2^i$ for $i \in [\log n]$: For every vertex $v \in V$, we sample each color in 1 to β independently and w.p. $\Theta(\log n)/\beta$ and use Proposition 3.5.1 with parameter $k = \Theta(\log^2(n))$ to store up to k monochromatic edges incident on every vertex. At the end of the stream, once we know the precise value of Δ, we consider the choice of β such that $\beta/2 \leq \Delta+1 \leq \beta$. We only consider the first $\Delta+1$ colors among 1 to β

and discard the remaining colors. A basic application of Chernoff bound ensures that with high probability, for every vertex $v \in V$, we still have sampled $\Theta(\log n)$ colors uniformly at random and independently from $\{1, \ldots, \Delta+1\}$. Moreover, it is easy to see that the proof of Lemma 3.5.2 implies that for this choice of β, we can recover the conflict-graph G_{conflict} from the ℓ_0-samplers. Hence, one can immediately verify that we can apply the previous proof and obtain a $(\Delta+1)$ coloring of G with high probability.

To turn this algorithm into a polynomial time algorithm, we implement DecompositionAlgorithm using the same trick of guessing Δ through $O(\log n)$ different choices for β. For each β, we sample $\Theta(n \log n/\beta)$ vertices, and for each vertices, we use 2β ℓ_0 samplers to get 2β neighbors of it (we get all its neighbors if its degree is at most 2β). At the end of the stream, we choose a β such that $\beta \leq \Delta \leq 2\beta$, and thus we can obtain all neighbors of $\Theta(n \log n/\Delta)$ randomly sampled vertices. And thus we can get a sparse-dense decomposition and obtain an $\widetilde{O}(n)$ time algorithm.

3.5.2. A Sublinear Time Algorithm for $(\Delta+1)$ Coloring

We now show another application of our palette-sparsification theorem to design sublinear algorithms. Consider the following standard query model for sublinear time algorithms on general graphs (see, e.g., Chapter 10 of Goldreich's book [131]): The vertex set of the graph is $V := [n]$ and the algorithm can make the following queries: (i) Degree queries: given $v \in V$, outputs degree $d(v)$ of v, (ii) Neighbor queries: given $v \in V$ and $i \leq d(v)$, outputs the i-th neighbor of v (the ordering of neighbors are arbitrary), and (iii) Pair queries: given $u, v \in V$, outputs whether the edge (u, v) is in E or not. We give a sublinear time algorithm (in size of the graph) for finding a $(\Delta+1)$ coloring in this query model.

Theorem 3.7. *There exists an algorithm that given a query access to a graph $G(V, E)$ with maximum degree Δ, can find a $(\Delta+1)$ coloring of G with high probability in $\widetilde{O}(n\sqrt{n})$ time and queries.*

We prove Theorem 3.7 by combining two separate algorithms and picking the best of the two

depending on the value of Δ. One is the straightforward (deterministic) greedy algorithm that takes $O(n\Delta)$ time to find a $(\Delta+1)$ coloring. This algorithm only uses neighbor queries. We use this algorithm when $\Delta \leq \sqrt{n}$. The other one is an implementation of ColoringAlgorithm in the query model which takes $\widetilde{O}(n^2/\Delta)$ time. This algorithm only uses pair queries. By using this algorithm when $\Delta \geq \sqrt{n}$, we achieve an $\widetilde{O}(n\sqrt{n})$ time algorithm for any graph (with potentially $\Omega(n^2)$ edges), proving Theorem 3.7.

To implement ColoringAlgorithm, we need to specify (i) how to construct the conflict-graph, and (ii) how to find a list-coloring in this conflict graph using the lists $L(\cdot)$. Throughout the proof, we condition on the high probability event in Lemma 3.4.1. The first part of the argument is quite easy as is shown below.

Claim 3.5.3. *$G_{\mathsf{conflict}}(V, E_{\mathsf{conflict}})$ can be constructed in $O(n^2 \cdot \log^2 n/\Delta)$ queries and time.*

Proof. As the vertices are known, we only need to construct the edges E_{conflict}. In order to do this, we simply query all pairs between vertices inside each set χ_c for $c \in [\Delta+1]$. By Lemma 3.4.1, size of each χ_c is $O(n \log n/\Delta)$ and so we need $O(n^2 \log^2 n/\Delta^2 \cdot (\Delta+1)) = O(n^2 \log^2 n/\Delta)$ queries. □

Claim 3.5.3 is already sufficient to obtain an $\widetilde{O}(n^2/\Delta)$ *query* (but not time) algorithm: by Lemma 3.4.1, ColoringAlgorithm outputs the correct answer by finding a list-coloring of G_{conflict} and accessing G_{conflict} does not require further queries to G. However, finding such a list-coloring problem in general is NP-hard and hence to find this coloring in sublinear time, we need to design an algorithm which further queries the graph G to obtain additional information for performing the coloring. To do so, we just need to obtain implement DecompositionAlgorithm as we shown in Section 3.4. To implement DecompositionAlgorithm in query model, we just need to sample $\Theta(n \log n/\Delta)$ random vertices, and find all their neighbors. It requires $O(n)$ pair queries or $O(\Delta)$ neighbor queries to find all neighbors of a vertices. So the total query complexity is $\Theta(n^2 \log n/\Delta)$ pair queries and $\Theta(n \log n)$ neighbor queries. After we get the spase-dense decomposition given by DecompositionAlgorithm,

we only need $\widetilde{O}(n)$ time to color the graph.

3.5.3. An MPC Algorithm for $(\Delta+1)$ Coloring

This section contains yet another application of our palette sparsification theorem to design sublinear algorithms, namely a massively parallel (MPC) algorithm for $(\Delta+1)$ coloring. In the MPC model of [42] (see also [12, 42, 135, 179]), the input is partitioned across multiple machines which are inter-connected via a communication network. The computation proceeds in synchronous rounds. During a round each machine runs a local algorithm on the data assigned to the machine. No communication between machines is allowed during a round. Between rounds, machines are allowed to communicate so long as each machine sends or receives a communication no more than its memory. Any data output from a machine must be computed locally from the data residing on the machine and initially the input data is distributed across machines adversarially. The goal is to minimize the total number of rounds subject to a small (sublinear) memory per machine and a small global memory (beside the memory used for storing the input).

We show that ColoringAlgorithm and DecompositionAlgorithm can be easily implemented in this model also and prove the following theorem.

Theorem 3.8. *There exists a randomized MPC algorithm that given a graph $G(V,E)$ with maximum degree Δ can find a $(\Delta+1)$ coloring of G with high probability in $O(1)$ MPC rounds with $\widetilde{O}(n)$ per-machine memory and $\widetilde{O}(n)$ global memory. Furthermore, if the machines have access to* shared randomness*, the algorithm only requires* one *MPC round.*

In the following, we assume that the machines have access to shared randomness and show how to solve the problem in only one MPC round. We then show that by spending $O(1)$ additional rounds, we can remove the assumption of public randomness.

The proof of this theorem is very similar to that of Theorem 3.6 and uses the close connection between dynamic streaming algorithms (in particular linear sketching algorithms) and MPC algorithms. As before, if we do not insist on achieving a polynomial time al-

gorithm, proving Theorem 3.8 from Lemma 3.4.1 is straightforward: we sample the color classes $\chi_1, \ldots, \chi_{\Delta+1}$ using public randomness, and every machine sends its edges in $G_{\textsf{conflict}}$ to a central designated machine, called the coordinator. As the total number of edges in $G_{\textsf{conflict}}$ is $\widetilde{O}(n)$ by Lemma 3.4.1, we only need $\widetilde{O}(n)$ memory on the coordinator. To implement DecompositionAlgorithm, we only need to sample $\Theta(n \log n/\Delta)$ vertices, and each machine send each edge that is incident on these vertices to the coordinator. Again, we only need $\widetilde{O}(n)$ memory on the coordinator. The coordinator machine can then locally find a list-coloring of the graph $G_{\textsf{conflict}}$ and find the $(\Delta+1)$ coloring in $\widetilde{O}(n)$ time by Lemma 3.4.1 and Lemma 3.4.4.

Finally, we show how to remove the public randomness. We first dedicate one machine M_v to each vertex v of the graph and spend the first round to send all the edges incident on v to the machine M_v. This can be done on machines of memory $O(\Delta)$. In the next round, each machine v samples the set of colors $L(v)$ for v and sends this information to all the machines M_u where (u, v) is an edge in the graph. This can again be done with $O(\Delta \cdot \text{polylog}(n))$ size messages and hence on machines of memory $\widetilde{O}(n)$. The machines can now send all edges in $E_{\textsf{conflict}}$ to a central coordinator and the coordinator can construct the graph $G_{\textsf{conflict}}$. To implement DecompositionAlgorithm also, the coordinate can sample $\Theta(n \log n/\Delta)$ vertices, and notify these machines whose corresponding vertices gets sampled. These machines then send all its incident edges to the central coordinator, and then the coordinator do the remaining work in $\widetilde{O}(n)$ time.

3.6. Optimality of Our Sublinear Algorithms

We now discuss the optimality of the bounds achieved by our sublinear algorithms.

Streaming Algorithms. Our dynamic streaming algorithm in Theorem 3.6 makes a single pass over the input and uses $\widetilde{O}(n)$ space. Obviously, the number of passes of our algorithm is optimal. Moreover, as simply storing the coloring of the graph requires $\Omega(n \log \Delta)$ bits, the space of our algorithm is optimal up to poly-log factors as well. As our algorithm works in dynamic streams, it can be directly implemented in insertion-only streams as well. More-

over, it is straightforward to verify that the same algorithm with minor modifications can be implemented in sliding-window streams as well by simply maintaining the conflict-graph and deleting any of its edges that goes outside the sliding window; we omit the details.

Sublinear Time Algorithms. Our algorithm in Theorem 3.7 makes $\widetilde{O}(\min\{n\Delta, n^2/\Delta\})$ non-adaptive queries to the graph and uses $\widetilde{O}(n\sqrt{n})$ time. Unlike the case for streaming algorithms, a-priori it is not clear whether this query complexity (and runtime) is optimal or not. However, in the following theorem whose proof appears in Section 3.6.1, we show that this is indeed the case in a strong sense, i.e., even for $O(\Delta)$-coloring and even for algorithms that query the graph adaptively.

Theorem 3.9. *For any constant $c > 1$, any algorithm (possibly randomized) that outputs a $(c \cdot \Delta)$ coloring of an input graph with sufficiently large constant probability requires $\Omega(n\sqrt{n})$ queries.*

MPC Algorithms. Finally, our MPC algorithm in Theorem 3.8 needs $O(1)$ rounds (and in fact just one round assuming access to public randomness) and $\widetilde{O}(n)$ memory per machine. The number of rounds is asymptotically optimal in our algorithm but it seems plausible that the memory per machine can be reduced further to n^{α} for constant $\alpha > 0$. However, an important aspect of our algorithm is that beside the input, it only needs to use $\widetilde{O}(n)$ extra memory (namely, has $\widetilde{O}(n)$ global memory). Again as $\Omega(n \log \Delta)$ global memory is needed to simply store the output, the global memory of our algorithm is also optimal up to poly-log factors.

3.6.1. A Query Lower Bound for $(\Delta + 1)$ Coloring

In this section, we prove Theorem 3.9 by showing that there exists a family of n-vertex graphs such that for any constant $c > 1$, any randomized algorithm that outputs a valid $(c \cdot \Delta)$ coloring on this family with probability at least $1 - o(1)$, requires $\Omega(n\sqrt{n})$ queries. For ease of notation, we will work with graphs with $2n$ vertices and focus on proving that finding a (1.99Δ) coloring requires $\Omega(n\sqrt{n})$ queries; essentially the same proof argument also implies an identical lower bound for a $(c \cdot \Delta)$ coloring.

The maximum degree Δ of each graph in our family will be $\sqrt{n}+1$. Since for large enough n, $1.99(\sqrt{n}+1) < 2\sqrt{n}$, it suffices to show that any randomized algorithm for finding a $(2\sqrt{n})$ coloring with large constant probability requires $\Omega(n\sqrt{n})$ queries.

By Yao's minimax principle [253], it suffices to create a distribution over graphs with $2n$ vertices such that any deterministic algorithm requires $\Omega(n\sqrt{n})$ queries to find a valid coloring with probability at least $1-o(1)$. A graph from our distribution is generated as follows. The vertex set is divided into $\sqrt{n}+1$ sets $V_0, V_1, \ldots, V_{\sqrt{n}}$ where V_0 has n vertices, and each set V_i, for $1 \leq i \leq \sqrt{n}$, has exactly $\sqrt{n}$ vertices. Furthermore, the vertices in V_0 are partitioned into $\sqrt{n}$ sets $V'_1, V'_2, \ldots, V'_{\sqrt{n}}$ where each set has $\sqrt{n}$ vertices. For any $1 \leq i \leq \sqrt{n}$, each vertex in V_i is connected to every vertex in V'_i. Finally, we pick a *random perfect matching* M inside V_0. The adjacency list of each vertex in the graph is a random permutation of its neighbor set. This completes the description of how a graph in this family is generated and presented to the algorithm.

The algorithm is given upfront the following information: (i) the partition of the vertices into sets $V_0, V_1, \ldots, V_{\sqrt{n}}, V'_1, V'_2, \ldots, V'_{\sqrt{n}}$, (ii) degrees of all the vertices, and (iii) all edges in the graph except the edges in the matching M. Thus the only task that remains for the algorithm is to discover enough information about the random matching M so as to output a valid $(2\sqrt{n})$ coloring with probability at least $1-o(1)$. This is the task we use to prove our lower bound. The high level strategy is as follows. We first argue that by making $o(n\sqrt{n})$, the algorithm is not able to find more than $o(n)$ edges of the matching M (with constant probability). The algorithm now has made all its queries and hence needs to commit to a coloring of the graph. We then show that no matter what coloring the algorithm chooses at this point, there is a non-trivial probability that one of the edges of M not queried by the algorithm appears inside one color class (i.e., becomes monochromatic), hence invalidating the output of the algorithm.

Lemma 3.6.1. *Any algorithm does at most $n\sqrt{n}/400000$ queries on graphs generated by the distribution above, outputs a valid $(2\sqrt{n})$ coloring with probability at most $3/4$.*

We say that an edge $(u, v) \in M$ has been *discovered* if the set of queries performed thus far uniquely identify the edge (u, v) to be in M. For any discovered edge $(u, v) \in M$, we will say the vertices u and v have been *discovered*; any vertex in V_0 that has not been discovered is called an *undiscovered* vertex.

After t queries have been made by the algorithm, let $U(t) \subseteq V_0$ denote the set of undiscovered vertices in V_0. Let $E(t) \subseteq U(t) \times U(t)$ denote the set of edge slots that have not yet been queried/discovered. For any vertex $w \in V_0$, we say that Q_2-*uncertainty* of w is d if there are at least d edges in $E(t)$ that are incident on w.

Additionally, we say that the state of the algorithm is *unsettled* after t queries if there are at least $\frac{9n}{10}$ vertices in $U(t)$ whose Q_2-uncertainty is at least $|U(t)| - \frac{\sqrt{n}}{5000}$; we will say that the state of the algorithm is *settled* otherwise. The proof of Lemma 3.6.1 has two parts. In the first part, we will show that if the state of the algorithm is unsettled after all the queries have been made, then any $(2\sqrt{n})$ coloring of the graph is invalid with some constant probability. In the second part, we will prove that to make the state of the algorithm settled, the algorithm needs $\Omega(n\sqrt{n})$ queries with a large constant probability.

Lemma 3.6.2. *Suppose we are given a graph G on n vertices such that each vertex in G has at least $n - \sqrt{n}/4000$ neighbors. Then if we pick a perfect matching uniformly at random in G, for any edge e, the probability that e is contained in the perfect matching is at most $\frac{1}{n-\sqrt{n}/1000}$.*

Proof. For any edge (u, v) in the graph, u and v have at least $n - \sqrt{n}/2000$ common neighbors. Consider any perfect matching M that contains the edge (u, v). By the assumption on the degree of vertices in G, there are at least $n/2 - \sqrt{n}/2000$ edges in the perfect matching M such that both end-points of these edges are neighbors of u and v. For each pair of such vertices (a, b), we can then obtain two perfect matching by replacing (u, v) and (a, b) with (u, a) and (v, b) or (u, b) and (v, a). Thus for every matching M containing the edge (u, v), we can generate a unique set of $n - \sqrt{n}/1000$ perfect matchings that do not contain the

edge (u, v). It then follows that the probability that a random perfect matching contains the edge (u, v) is at most $\frac{1}{n-\sqrt{n}/1000}$ □

We now prove that if the state of the algorithm is unsettled after it finishes the queries, then the coloring output by the algorithm is invalid with constant probability.

Lemma 3.6.3. *If the state of the algorithm is unsettled after it finishes the queries, then any $(2\sqrt{n})$ coloring output by the algorithm is invalid with probability at least $3/8 - o(1)$.*

We start by the following key lemma.

Lemma 3.6.4. *Given a graph G on n vertices such that each vertex in G has at least $n - \sqrt{n}/4000$ neighbors. If we randomly pick a perfect matching uniformly, then for each vertex v, there are at most $\sqrt{n}/9$ vertices u such that the probability that (u, v) is contained in the perfect matching is less than $\frac{99}{100n}$.*

Proof. Fix a vertex v, let $P_v(u)$ be the probability that the edge (u, v) is contained in a random perfect matching, it is also the probability density function of the distribution of v's neighbor in a random perfect matching. For any vertex u, by Lemma 3.6.2, $P_v(u) \leq \frac{1}{n-\sqrt{n}/1000} < 1/n + \frac{1}{900n\sqrt{n}}$. Let $\mathcal{U}$ be the uniform distribution over the vertices of G, then the ℓ_1-distance between the two distributions satisfies $\|P_v - \mathcal{U}\|_1 < \frac{1}{900\sqrt{n}}$. So there are $\leq \frac{\sqrt{n}}{9}$ vertices u with $P_v(u) \leq \frac{99}{100n}$. □

Proof of Lemma 3.6.3. Since the state of the algorithm is still unsettled, there are at most $|U(t)| - 9n/10$ vertices whose Q_2-uncertainty is less than $|U(t)| - \sqrt{n}/5000$. We give the algorithm all the edges in M incident on these vertices as well as a few additional edges in M if needed so that the number of undiscovered vertices becomes exactly $4n/5$. Let U' be the set of these undiscovered vertices. After this step, these undiscovered vertices have Q_2-uncertainty at least $4n/5 - \sqrt{n}/5000$.

Fix the output $2\sqrt{n}$ coloring of the graph by the algorithm. Let $S \subseteq U' \times U'$ be the set

of pairs of vertices in U' that have the same color. It is not hard to verify that $|S| \geq \frac{1}{2} \cdot \frac{4n}{5} \cdot (\frac{4n}{5} \cdot \frac{1}{2\sqrt{n}} - 1) \geq 0.15n\sqrt{n}$. For any pair of vertices $(u, v) \in S$, let $X_{u,v}$ be the 0/1 indicator variable that indicates (u, v) is in M. Let $X = \sum_{(u,v)\in S} X_{u,v}$; then $\Pr(X > 0)$ is the probability that the coloring output by the algorithm is invalid. We will use Chebyshev's inequality to prove that this probability is a large enough constant.

By Lemma 3.6.4, for any vertex $v \in U'$, there are at most $\sqrt{n}/9$ vertices u such that $\Pr(X_{u,v}) < \frac{99}{100n}$. So

$$\mathbb{E}[X] \geq (|S| - \left|U'\right| \cdot (\sqrt{n}/9)) \cdot (99/100n) \geq (|S| - 0.1n\sqrt{n}) \cdot (99/100n) \geq 0.2\,|S|\,/n.$$

Now consider any two pairs of vertices u, v and u', v' in S. If these two pairs share a vertex, then at least one of the pairs does not appear as an edge in the matching M, which means that the covariance of $X_{u,v}$ and $X_{u',v'}$ is negative. If they do not share a vertex, then $\Pr\left(X_{u,v} \cdot X_{u',v'} = 1\right)$ is the probability that both pairs are in M. If we pick a random matching conditioned on the event that (u, v) is in the matching, using the same argument as in the proof of Lemma 3.6.2, the probability that (u', v') is picked is at most $\frac{801}{800|U'|}$. So $\Pr\left(X_{u',v'} = 1 | X_{u,v} = 1\right) \leq \frac{801}{800|U'|}$, which means the $\mathrm{Cov}\left[X_{u,v}, X_{u',v'}\right] \leq \Pr(X_{u,v} = 1) \cdot (\frac{801}{800|U'|} - \Pr\left(X_{u',v'} = 1\right)) \leq 0.015/\left|U'\right|^2 \leq 0.025/n^2$. So

$$\mathrm{Var}[X] \leq \mathbb{E}[X] + \sum_{(u,v),(u',v')\in S} 0.02/n^2 \leq \mathbb{E}[X] + 0.02\,|S|^2\,/n^2 \leq \mathbb{E}[X] + 5\,\mathbb{E}[X]^2\,/8.$$

By Chebyshev's inequality, $\Pr(X = 0) \leq \frac{\mathrm{Var}[X]}{\mathbb{E}[X]^2} \leq 5/8 + o(1)$. □

We next prove that to make the state of the algorithm settled, the algorithm needs $\Omega(n\sqrt{n})$ queries with large constant probability.

Lemma 3.6.5. *If the algorithm only makes $n\sqrt{n}/400000$ queries, then with probability 0.9, the state of the algorithm is unsettled.*

Proof. If the algorithm does a Q_1 query, say $Q_1(v,k)$, or does a Q_2 query, say $Q_2(u,v)$, then we say that the vertex v has been queried. During the execution of the algorithm, once a vertex v is queried $\sqrt{n}/5000$ times, we declare both the vertex v and the vertex u to which it is matched in M as *bad* vertices, and any further queries on v or u are called *useless* queries. A query which is not useless is called a *useful* query. After all the queries, if an undiscovered vertex v has not been queried $\sqrt{n}/5000$ times, then vertex v has Q_2-uncertainty at least $U(t) - \sqrt{n}/5000$. So to prove the lemma, we need to prove that the total number of bad vertices and discovered vertices is at most $n/10$. The number of bad vertices is at most $n/20$ since at most two vertices are affected by a single query. On the other hand, any discovered vertex which is not bad must be discovered by a useful query. For a useful Q_1 query, say $Q_1(v,k)$, the probability that v is discovered in this query is at most $\frac{1}{\sqrt{n}+1-\sqrt{n}/5000} \leq \frac{2}{\sqrt{n}}$. For a useful Q_2 query, if the number of discovered vertices is less than $n/10$, then by Lemma 3.6.2, the probability that this query discover an edge in M is at most $\frac{901 \cdot 10}{900 \cdot 9n} < \frac{1.2}{n}$. So for any useful query, the probability that it discovers an edge in M is at most $\frac{2}{\sqrt{n}}$. As such, the expected number of useful queries which discover an edge in M is at most $n/200000$. By Markov, this implies that the number of useful queries that discover an edge M is with probability 0.9 at most $n/20000$. So the number of discovered vertices which are not bad is at most $n/10000$. This in turn implies that the total number of bad vertices and discovered vertices is less than $n/10$ with probability 0.9. □

Proof of Lemma 3.6.1. By Lemma 3.6.3, if the algorithm only makes $n\sqrt{n}/400000$ queries, then with probability 0.9 the state of the algorithm is unsettled. Conditioned on this event, by Lemma 3.6.5, the output of the algorithm is an invalid coloring with probability at least $3/8 - o(1)$. So with probability $3/8 - 0.1 - o(1) \geq 1/4$, the output of any algorithm that makes less than $n\sqrt{n}/400000$ queries is an invalid coloring. □

Theorem 3.9 is directly implied by Lemma 3.6.1 (by modifying constants to allow for $c\sqrt{n}$ coloring as opposed to $2\sqrt{n}$).

CHAPTER 4

Sublinear Algorithms for Traveling Salesman Problem

In this chapter, we consider the metric traveling salesman problem in query model and streaming model. A standard approach to estimating the metric TSP cost is to compute the cost of a minimum spanning tree (MST), and output two times this cost as the estimate of the TSP cost (since any spanning tree can be used to create a spanning simple cycle by at most doubling the cost). The problem of approximating the cost of the minimum spanning tree in sublinear time was first studied in the graph adjacency-list model by Chazelle, Rubinfeld, and Trevisan [83]. The authors gave an $\tilde{O}(dW/\varepsilon^2)$-time algorithm to estimate the MST cost to within a $(1+\varepsilon)$-factor in graphs where average degree is d, and all edge costs are integers in $[1..W]$. For certain parameter regimes this gives a sublinear time algorithm for estimating the MST cost but in general, this run-time need not be sublinear. Subsequently, in an identical setting as ours, Czumaj and Sohler [100] showed that for any $\varepsilon > 0$, there exists an $\tilde{O}(n/\varepsilon^{O(1)})$ time algorithm that returns a $(1+\varepsilon)$-approximate estimate of the MST cost when the input is an n-point metric. This result immediately implies an $\tilde{O}(n/\varepsilon^{O(1)})$ time algorithm to estimate the TSP cost to within a $(2+\varepsilon)$ factor for any $\varepsilon > 0$. However, no $o(n^2)$ query algorithms are known to approximate metric TSP to a factor that is strictly better than 2. On the other hand, there are also no known barriers that rule out existence of $(1+\varepsilon)$-approximate estimation algorithms for metric TSP with $\tilde{O}(n)$ queries for any fixed $\varepsilon > 0$. In this chapter, we make progress on both algorithms and lower bounds for estimating metric TSP cost.

4.1. Main Results

On the algorithmic side, we first consider the *graphic TSP* problem, an important case of metric TSP that has been extensively studied in the classical setting – the metric D corresponds to the shortest path distances in a connected unweighted undirected graph [210, 212, 239]. We give the first $\tilde{O}(n)$ time algorithm for graphic TSP that achieves an approximation factor *strictly better* than 2.

Theorem 4.1. *There is an $\tilde{O}(n)$ time randomized algorithm that estimates the cost of graphic TSP to within a factor of* $(27/14)$.

At a high-level, our algorithm is based on showing the following: if a graph G either lacks a matching of size $\Omega(n)$ or has $\Omega(n)$ biconnected components (blocks), then the optimal TSP cost is not too much better than $2n$. Note that a connected unweighted instance of graphic TSP always contains a TSP tour of cost at most $2n$ since the MST cost is $(n-1)$ on such instances. Conversely, if the graph G has both a large matching and not too many blocks, then we can show that the optimal TSP cost is distinctly better than $2n$. Since we do not know an efficient sublinear algorithm to estimate the number of blocks in a graph G, we work with another quantity that serves as a proxy for this and can be estimated in $\tilde{O}(n)$ time. The main remaining algorithmic challenge then is to estimate sufficiently well the size of a largest matching. This problem is very important by itself, and has received much attention [44, 171, 218, 220, 225, 255]. Our $\tilde{O}(n)$ query results utilize the recent result of Behnezhad [44] who give an algorithm to 2-approximate the size of maximum matching in $\tilde{O}(n)$ time in the pair query model.

Our approach for estimating graphic TSP cost in sublinear time also lends itself to an $\tilde{O}(n)$ space streaming algorithm that can obtain an even better estimate of the cost. To our knowledge, no estimate better than a 2-approximation was known previously. In the streaming model, we assume that the input to graphic TSP is presented as a sequence of edges of the underlying graph G. Any algorithm for this model, clearly also works if instead the entries of the distance matrix are presented in the stream – an entry that is 1 corresponds to an edge of G, and it can be ignored otherwise as a non-edge.

Theorem 4.2. *There is an $O(n)$ space randomized streaming algorithm that estimates the cost of graphic TSP to within a factor of* $(11/6)$ *in insertion-only streams.*

We also consider another well-studied special case of metric TSP, namely, $(1,2)$-TSP where all distances are either 1 or 2 [3, 56, 224]. Throughout the chapter, whenever we refer to

a graph associated with a $(1,2)$-TSP instance, it refers to the graph G induced by edges of distance 1 in our $\{1,2\}$-metric. The cost of $(1,2)$-TSP is close related to the size of maximum matching. Thus, the matching algorithm given by Behnezhad [44] also implies an approximation algorithm for $(1,2)$-TSP.

Theorem 4.3 ([44])**.** *There is an $\tilde{O}(n)$ time randomized algorithm that estimates the cost of $(1,2)$-TSP to within a factor of 1.75.*

We note that it is easy to show that randomization is crucial to getting better than a 2-approximation in sublinear time for both graphic TSP and $(1,2)$-TSP. The algorithms underlying Theorem 4.1, lend themselves to $\tilde{O}(n)$ space single-pass streaming algorithms with identical approximation guarantees. These sublinear time algorithms motivate the natural question if analogously to metric MST, there exist sublinear time algorithms that for any $\varepsilon > 0$, output a $(1+\varepsilon)$-approximate estimate of TSP cost for graphic TSP and $(1,2)$-TSP in $\tilde{O}(n)$ time. We rule out this possibility in a strong sense for both graphic TSP and $(1,2)$-TSP.

Theorem 4.4. *There exists an $\varepsilon_0 > 0$, such that any randomized algorithm that estimates the cost of graphic TSP ($(1,2)$-TSP) to within a $(1+\varepsilon_0)$-factor, necessarily requires $\Omega(n^2)$ queries.*

This lower bound result highlights a sharp separation between the behavior of metric MST and metric TSP problems. At a high-level, our lower bound is inspired by the work of Bogdanov *et al.* [60] who showed that any query algorithm that for any $\varepsilon > 0$ distinguishes between instances of parity equations (mod 2) that are either satisfiable (Yes) or at most $(1/2+\varepsilon)$-satisfiable (No), requires $\Omega(n)$ queries where n denotes the number of variables. However, the query model analyzed in [60] is different from ours. We first show that the lower bound of [60] can be adapted to an $\Omega(n^2)$ lower bound in our model, and then show that instances of parity equations can be converted into instances of graphic TSP (resp. $(1,2)$-TSP) such that for some $\varepsilon_0 > 0$, any $(1+\varepsilon_0)$-approximation algorithm for graphic

TSP (resp. $(1,2)$-TSP), can distinguish between the Yes and No instances of the parity equations, giving us the desired result.

Finally, similar to many classical approximation algorithms for TSP, our sublinear time estimation algorithms utilize subroutines for estimating the size of a maximum matching in the underlying graph. We show that this is not merely an artifact of our approach.

Theorem 4.5. *For any $\varepsilon \in [0, 1/5)$, any algorithm that estimates the cost of an n-vertex instance of graphic TSP or $(1,2)$-TSP to within a $(1+\varepsilon)$-factor, can also be used to estimate the size of a maximum matching in an n-vertex bipartite graph to within an εn additive error, with an identical query complexity, running time, and space usage.*

This connection allows us to translate known lower bounds for matching size estimation in various models to similar lower bounds for metric TSP cost estimation. In particular, using the results of [20], we can show that there exists an ε_0 such that any randomized single-pass dynamic streaming algorithm for either graphic TSP or $(1,2)$-TSP that estimates the cost to within a factor of $(1+\varepsilon_0)$, necessarily requires $\Omega(n^2)$ space.

We conclude by establishing several additional lower bound results that further clarify the query complexity of approximating TSP cost. For instance, we show that if an algorithm can access an instance of graphic TSP by only querying the edges of the graph (via neighbor and pair queries), then any algorithm that approximates the graphic TSP cost to a factor better than 2, necessarily requires $\Omega(n^2)$ queries. This is in sharp contrast to Theorem 4.1, and shows that working with the distance matrix is crucial to obtaining sublinear time algorithms for graphic TSP. We also show that even in the distance matrix representation, the task of *finding a tour* that is $(2-\varepsilon)$-approximate for any $\varepsilon > 0$, requires $\Omega(n^2)$ queries for both graphic TSP and $(1,2)$-TSP.

4.1.1. Other Related Work

We note here that there is an orthogonal line of research that focuses on computing an approximate solution in near-linear time when the input is presented as a weighted undirected

graph, and the metric is defined by shortest path distances on this weighted graph. It is known that in this model, for any $\varepsilon > 0$, there is an $\tilde{O}(m/\varepsilon^2 + n^{1.5}/\varepsilon^3)$ time algorithm that computes a $(3/2 + \varepsilon)$-approximate solution; here n denotes the number of vertices and m denotes the number of edges [85], and that a $(3/2 + \varepsilon)$-approximate estimate of the solution cost can be computed in $\tilde{O}(m/\varepsilon^2)$ time [84]. It is not difficult to show that in this access model, even when the input graph is unweighted (i.e. a graphic TSP instance), any algorithm that outputs better than a 2-approximate estimate of the TSP cost, requires $\Omega(n + m)$ time even when $m = \Omega(n^2)$. Hence this access model does not admit sublinear time algorithms that beat the trivial 2-approximate estimate.

4.2. Approximation for Graphic TSP Cost

In this section, we exploit well-known properties of biconnected graphs and biconnected components in graphs to give an algorithm that achieves a $(2 - \frac{1}{7c_0})$-approximation for graphic TSP if we have an efficient algorithm that approximates the maximum matching size within a factor of c_0. We first relate the cost of the TSP tour in a graph to the costs of the TSP tours in the biconnected components of the graph. Next we show that if the graph does not have a sufficiently big matching, it does not have a TSP tour whose length is much better than $2n$. We also show that if a graph has too many degree 1 vertices, or vertices of degree 2, both whose incident edges are bridges, then it does not have a TSP tour of cost much better than $2n$. We then establish the converse - a graph that has a good matching and not too many bad vertices (namely, vertices of degree 1 or articulation points of degree 2), then it necessarily has a TSP tour of cost much better than $2n$. We design $\tilde{O}(n)$ time test for the second condition, allowing us to approximate the cost of an optimal graphic TSP tour in sublinear time together with some known techniques for testing the first condition. In what follows, we first present some basic concepts and develop some tools that will play a central role in our algorithms.

4.2.1. Preliminaries

An unweighted graph $G = (V, E)$, defines a *graphic metric* in V, where the distance between any two vertices u and v is given by the length of the shortest path between u and v. The *graphic TSP* is the Traveling Salesman Problem defined on such a graphic metric. In this paper our goal is to find a non-trivial approximation to the length of the traveling salesman tour in sublinear time in a model where we are allowed to make *distance queries*. In the distance query model, the algorithm can make a query on a pair of vertices (u, v) and get back the answer $d(u, v)$, the distance between u and v in G.

In a connected graph G, an edge e is a *bridge* if the deletion of e would increase the number of connected components of G. A connected graph with no bridge is called a *2-edge-connected graph*. A maximal 2-edge-connected subgraph of G is called a *2-edge-connected component*. The *bridge-block tree* of a graph is a tree such that the vertex set contains the 2-edge-connected components and the edge set contains the bridges in the graph.

A connected graph G is called *2-vertex-connected* or *biconnected* if when any one vertex is removed, the resulting graph remains connected. In a graph which is not biconnected, a vertex v whose removal increases the number of components is called an *articulation point*. It is easy to prove that any biconnected graph with at least 3 vertices does not have degree 1 vertices. A well-known alternate characterization of biconnectedness is that, a graph G is biconnected if and only if for any two distinct edges, there is a simple cycle that contains them.

A *biconnected component* or *block* in a graph is a maximal biconnected subgraph. Any graph G can be decomposed into blocks such that the intersection of any two blocks is either empty, or a single articulation point. Each articulation point belongs to at least two blocks. If a block is a single edge, then we call this block a *trival block*; otherwise it is a *non-trivial block*. A trival block is also a bridge in the graph. The size of a block is the number of vertices in the block. The following lemma shows the relationship between the number of blocks and

the sum of the sizes of the blocks.

Lemma 4.2.1. *If a connected graph G has n vertices and k blocks, then the sum of the sizes of the blocks is equal to $n + k - 1$.*

Proof. We prove the lemma by induction on the number k of blocks. The base case is when $k = 1$. In this case, G itself is a block of size n.

For the induction step, we have $k > 1$ and thus the graph has at least one articulation point. Suppose v is an arbitrary articulation point in G. Let $V_1, V_2, \ldots, V_j$ be the set of vertices in the connected components of $G \setminus \{v\}$. We have $\sum_{i=1}^{j} |V_i| = n - 1$. Let $G_1, G_2, \ldots, G_j$ be the subgraphs of G induced by $V_1 \cup \{v\}, V_2 \cup \{v\}, \ldots, V_j \cup \{v\}$. For any G_i, let k_i be the number of blocks in G_i, we have $\sum_{i=1}^{j} k_i = k$. By induction hypothesis, the sum of the sizes of blocks in G_i is $|V_i| + 1 + k_i - 1 = |V_i| + k_i$. So the sum of the sizes of blocks in G is $\sum_{i=1}^{j} |V_i| + k_i = n - 1 + k$. □

The block decomposition of a graph has a close relationship with the cost of graphic TSP of the graph.

Lemma 4.2.2 (Lemma 2.1 of [209])**.** *The cost of the graphic TSP of a connected graph $G = (V, E)$ is equal to the sum of the costs of the graphic TSP of all blocks in the graph.*

Together these two lemmas give us a simple lower bound on the cost of the graphic TSP of a graph G (using the fact that the cost of graphic TSP is at least the number of vertices in the graph).

Lemma 4.2.3. *If a graph G has n vertices and k blocks, then the cost of graphic TSP of G is at least $n + k - 1$.*

An *ear* in a graph is a simple cycle or a simple path. An ear which is a path is also called an *open ear* and it has two endpoints, whereas for a cycle, one vertex is designated as the endpoint. An *ear decomposition* of a graph is a partition of a graph into a sequence of

ears such the endpoint(s) of each ear (except for the first) appear on previous ears and the internal points (the points that are not endpoints) are not on previous ears. A graph G is biconnected if and only if G has an ear decomposition such that each ear but the first one is an open ear [248]. An ear is *nontrivial* if it has at least one internal point. The following lemma upper bounds the cost of graphic TSP of a biconnected graph.

Lemma 4.2.4 (Lemma 5.3 of [239], also a corollary of Lemma 3.2 of [210])**.** *Given a 2-vertex-connected graph $G = (V, E)$ and an ear-decomposition of G in which all ears are nontrivial, a graphic TSP tour of cost at most $\frac{4}{3}(|V|-1)+\frac{2}{3}\pi$ can be found in $O(|V|^3)$ time, where π is the number of ears.*

We now prove an important lemma that gives an upper bound on the cost of graphic TSP in a biconnected graph in terms of the size of a matching in the graph.

Lemma 4.2.5. *Suppose G is a biconnected graph with at least $n \geq 3$ vertices. If G has a matching M, then the cost of graphic TSP of G is at most $2n - 2 - \frac{2|M|}{3}$.*

Proof. We first find a spanning biconnected subgraph of G that only contains $2n - 2 - M$ edges, then use Lemma 4.2.4 to bound the cost of graphic TSP.

We construct a spanning biconnected subgraph $G^\star = P_0 \cup P_1 \cup \dots$ recursively: P_0 contains a single edge in M. If $G_{i-1} = P_0 \cup P_1 \cup \dots \cup P_{i-1}$ is a spanning subgraph of G, let $G^\star = G_{i-1}$ and finish the construction. Otherwise we construct P_i as follows. Let e be an edge in M both whose endpoints are not in G_{i-1}. If there is no such edge, then let e be an arbitrary edge such that at least one of its endpoints is not in G_{i-1}. Let e' be an arbitrary edge in G_{i-1}. By the alternate characterization of biconnectedness, there is a simple cycle C_i that contains both e and e'. Let P_i be the path in C_i that contains e and exactly two vertices in G_{i-1}, which are the endpoints of P_i.

Since P_i contains at least one vertex not in G_{i-1}, the construction always terminates. Note that $P_0 \cup P_1$ is a cycle, and each P_i $(i > 1)$ is an open ear of $G^\star$. So, $(P_0 \cup P_1, P_2, \dots)$ is an

open ear decomposition of $G^\star$, which means $G^\star$ is biconnected.

Now we prove that the number of edges in $G^\star$ is at most $2n-2-M$. Let n_i be the number of vertices in $G_i \backslash G_{i-1}$. Let G_{-1} be the empty graph, so that n_0=2. Let p_i be the number of edges in P_i and m_i be the number of edges e in M such that $e \cap G_i \neq \emptyset$ and $e \cap G_{i-1} = \emptyset$. (Here we view an edge as a 2-vertex set.) Note that $m_0 = 1$. Suppose $G^\star = G_k$. Then $\sum_{i=1}^{k} n_i = n$, $\sum_{i=1}^{k} p_i$ is the number of edges in $G^\star$ and $\sum_{i=1}^{k} m_i = |M|$. For any $i > 0$, P_i is an open ear whose internal points are not in G_{i-1}. So $n_i = p_i - 1$. If there is an edge $e \in M$ such that $e \cap G_{i-1} = \emptyset$, then P_i contains both endpoints of an edge in M, which means $m_i \leq n_i - 1$. If all edges in M already have an endpoint in G_{i-1}, $m_i = 0 \leq n_i - 1$. So in both cases, $p_i = n_i + 1 = 2n_i - (n_i - 1) \leq 2n_i - m_i$. Also, $p_0 = 1 = 2n_0 - 2 - m_0$. So the number of edges in $G^\star$ is $\sum_{i=0}^{k} p_i \leq 2n_0 - 2 - m_0 + \sum_{i=1}^{k}(2n_i - m_i) = 2n - 2 - |M|$.

Since $(P_0 \cup P_1, P_2, P_3, \ldots, P_k)$ is an open ear decomposition of $G^\star$, the number of ears in G is k. On the other hand, $\sum_{i=0}^{k} p_i = 1 + \sum_{i=1}^{k}(n_i+1) = n-1+k$, we have $n-1+k \leq 2n-2-|M|$, which means $k \leq n - 1 - |M|$. By Lemma 4.2.4, the cost of graphic TSP of $G^\star$ is at most $\frac{4}{3}(n-1) + \frac{2}{3}k \leq 2(n-1) - \frac{2}{3}|M|$.

Since $G^\star$ is a subgraph of G that contains all the vertices in G, the cost of graphic TSP of G is at most the cost of graphic TSP of $G^\star$, which is at most $2n - 2 - \frac{2}{3}|M|$. □

4.2.2. Approximation Algorithm for Graphic TSP

In this section, we give the algorithm that approximates the cost of graphic TSP of a graph G within a factor of less than 2.

We call a vertex v a *bad* vertex if v has degree 1 or is an articulation point with degree 2.

For any given $\delta > 0$, the graphic TSP algorithm performs the following two steps.

1. Obtain an estimate $\hat{\alpha}n$ of the size of maximum matching αn.

2. Obtain an estimate $\hat{\beta}n$ of the number of bad vertices βn.

The algorithm then output $\min\{2n, (2 - \frac{2}{7}(\hat{\alpha} - 2\hat{\beta}))n\}$.

To perform the second step in $\tilde{O}(n)$ distance queries and time, we randomly sample $O(\frac{1}{\delta^2})$ vertices. For each sampled vertex, we can obtain the degree with n queries. The following lemma shows that we can also check whether a degree 2 vertex is an articulation point using distance queries in $O(n)$ time. Then by the Chernoff bound, we can approximate the number of bad vertices with additive error $O(\delta n)$ with a high constant probability.

Lemma 4.2.6. *Suppose a vertex v in a connected graph G has only two neighbors u and w. The following three conditions are equivalent:*

1. *v is an articulation point.*
2. *The edges (u, v) and (v, w) are both bridges.*
3. *For any vertex $v' \neq v$, $|d(u, v') - d(w, v')| = 2$.*

Proof. We first prove the first two conditions are equivalent. If v is an articulation point, then v is in two different blocks. So edge (u, v) and (v, w) are in different blocks, which means v has degree 1 in both blocks. So both blocks are trivial, which means (u, v) and (v, w) are both bridges. If (u, v) and (v, w) are both bridges, then deleting either (u, v) or (v, w) will disconnect u and w, which means deleting v will also disconnect u and w.

Next we prove that the third condition is equivalent to the first two. Suppose v is an articulation point. Since v has degree 2, the graph $G \setminus \{v\}$ has only two components, one containing u and the other containing w. For any vertex $v' \neq v$, without loss of generality, suppose v' is in the same component as u in $G \setminus \{v\}$. Since (u, v) and (v, w) are both bridges in G, any path between v' and w contains u and v. So $d(v', w) = d(v', u) + 2$.

If v is not an articulation point, then u and w are connected in $G \setminus \{v\}$. Let $(u = v_0, v_1, v_2, \ldots, v_k = w)$ be the shortest path between u and w in $G \setminus \{v\}$. For any vertex v_i on the path, the distance between v_i and u (resp. w) in $G \setminus \{v\}$ is i (resp. $k - i$).

Consider the shortest path between u and v_i in G. If this path does not contain v, then it is the same as the path in $G \setminus \{v\}$. In this case, $d(u, v_i) = i$. If the shortest path contains v, then v must be the second last vertex on the path and w be the third last one. In this case, $d(u, v_i) = k - i + 2$. So $d(u, v_i) = \min\{i, k - i + 2\}$. Similarly, we also have $d(v_i, w) = \min\{i + 2, k - i\}$. Let $v' = v_{\lfloor k/2 \rfloor}$. Since $|i - (k - i)| \leq 1$, we have $i < k - i + 2$ and $k - i < i + 2$, which means $|d(u, v') - d(w, v')| = |i - (k - i)| \leq 2$. □

Next, we prove that if α is small or β is large, the cost of graphic TSP is bounded away from n. The following lemma shows that if the size of maximum matching of a graph is small, then the cost of the graphic TSP is large.

Lemma 4.2.7. *For any $\varepsilon > 0$, if the maximum matching of a graph G has size at most $\frac{(1-\varepsilon)n}{2}$, then the cost of graphic TSP of G is at least $(1 + \varepsilon)n$.*

Proof. Suppose the optimal TSP tour is $(v_0, v_1, \ldots, v_{n-1}, v_n = v_0)$. Since the size of maximum matching in G is at most $\frac{(1-\varepsilon)n}{2}$, there are at most $\frac{(1-\varepsilon)n}{2}$ edges between pairs (v_i, v_{i+1}) where i is even (resp. odd). So there are at least εn pairs of (v_i, v_{i+1}) that have distance at least 2, which means that the optimal cost of TSP tour of G is $\sum_{i=1}^{n-1} d(v_i, v_{i+1}) \geq n + \varepsilon n = (1 + \varepsilon)n$. □

The following lemma shows that if β is large, the cost of graphic TSP is large.

Lemma 4.2.8. *For any $\varepsilon > 0$, if a connected graph G has εn bad vertices, then the cost of graph-TSP of G is at least $(1 + \varepsilon)n - 2$.*

Proof. We first prove by induction on the number of vertices that a graph with k bad vertices has $k - 1$ bridges. The base case is when $n = 2$, the graph has $k = 2$ bad vertices and $1 = k - 1$ bridge.

For the induction step, the graph has n vertices with $n \geq 3$. If G has no degree 1 vertices, then the graph has k articulation points with degree 2. By Lemma 4.2.6, any edge incident

on a degree 2 articulation point is a bridge. So each bad vertex is incident on 2 bridges. On the other hand, a bridge is incident on at most 2 vertices. So there are at least $\frac{2k}{2} = k$ bridges in G. Next, suppose G has degree 1 vertices. Let v be an arbitrary such vertex and let u be its neighbor. Since G is connected and $n \geq 3$, u must has degree at least 2, since otherwise u and v are not connected to other vertices in G. Consider the graph $G \setminus \{v\}$, if u is a bad vertex in G, u has degree 1 in $G \setminus \{v\}$ and is still a bad vertex. So the number of bad vertices in $G \setminus \{v\}$ is $k-1$. By induction hypothesis, $G \setminus \{v\}$ has at least $k-2$ bridges. G has at least $k-1$ bridges since (u, v) is also a bridge.

So G has at least $\varepsilon n - 1$ bridges, and the number of blocks in G is at least $\varepsilon n - 1$. By Lemma 4.2.3, the cost of graph-TSP of G is at least $n + \varepsilon n - 2 = (1 + \varepsilon)n - 2$. □

Finally, the following lemma shows that the cost of graphic TSP is at most $(2 - \frac{2}{7}(\hat{\alpha} - 2\beta))n$.

Lemma 4.2.9. *If a graph has a matching M of size $\alpha' n$ and the graph has βn bad vertices, the cost of graphic TSP of G is at most $(2 - \frac{2}{7}(\alpha' - 2\beta))n$.*

Proof. Let $G_1, G_2, \ldots, G_k$ be the block decomposition of G. Let n_i be the size of G_i. If $|n_i| \geq 3$, by Lemma 4.2.5, the cost of the graphic TSP of G_i is at most $2n_i - 3$ since any non-empty graph has a matching of size at least 1. If $|n_i| = 2$, then the graphic TSP of G_i is exactly $2 = 2n_i - 2$. Suppose G has ℓ non-trivial blocks. Then by Lemma 4.2.2 the cost of graphic TSP of G is at most $\sum_{i=1}^{k}(2n_i - 2) - \ell$, which equals to $2n - 2 - \ell$ by Lemma 4.2.1.

Let m_i be the size of maximum matching in G_i if G_i is a non-trivial block, and let $m_i = 0$ if G_i is a trivial block. By Lemma 4.2.5, the cost of the graphic TSP of G_i is at most $2n_i - 2 - \frac{2m_i}{3}$. For any non-trivial block G_i, $M \cap G_i$ is a matching in G_i. So the size of maximum matching in G_i is at least the number of edges in $M \cap G_i$. So by Lemma 4.2.2 and Lemma 4.2.1, the cost of graphic TSP of G is at most $\sum_{i=1}^{k}(2n_i - 2 - \frac{2}{3}m_i) = 2n - 2 - \frac{2}{3}|M'|$, where M' is the set of edges in M that are not bridges in G. Let B be the number of bridges in G. We have $2n - 2 - \frac{2}{3}|M'| \leq 2n - 2 - \frac{2}{3}(|M| - B)$.

So there are two upper bounds of the graphic TSP of G — $2n-2-\ell$ and $2n-2-\frac{2}{3}(|M|-B)$. Which bound is better depends on the number of bridges B.

If $B \leq (\frac{4}{7}\alpha' + \frac{6}{7}\beta)n$, the cost of graphic TSP of G is at most

$$2n - 2 - \frac{2}{3}(|M| - B) \leq 2n - \frac{2}{3}(\frac{3}{7}\alpha' - \frac{6}{7}\beta)n = (2 - \frac{2}{7}(\alpha' - 2\beta))n$$

If $B > (\frac{4}{7}\alpha' + \frac{6}{7}\beta)n$, consider the bridge-block tree T of G. T has at least B edges and at least $B+1$ vertices. Since T is a tree, there are at least $\frac{B}{2}$ vertices of degree at most 2. For any vertex v_T of degree at most 2 in T, if the vertex v_T represents a single vertex v in G, then v is either a degree 1 vertex or a degree 2 articulation point in G, otherwise v_T represents a 2-edge-connected component of size at least 2 in G. So There are at least $\frac{B}{2} - \beta n \geq (\frac{2}{7}\alpha' - \frac{4}{7}\beta)n$ 2-edge-connected components of size at least 2. Since any 2-edge-connected component of size at least 2 has no bridge, each such component of G contains at least 1 non-trivial block in G, implying that $\ell \geq \frac{2}{7}(\alpha' - 2\beta)n$. So the cost of graphic TSP of G is at most $2n - 2 - \ell \leq (2 - \frac{2}{7}(\alpha' - 2\beta))n$. □

We summarize the ideas in this section and prove the following lemma.

Lemma 4.2.10. *For any $c_0 > 1$ and $\delta > 0$, suppose $\hat{\alpha} \leq \alpha \leq c_0\hat{\alpha} + \delta$ and $\hat{\beta} - \delta \leq \beta \leq \hat{\beta}$. Then $(2 - \frac{2}{7}(\hat{\alpha} - 2\hat{\beta}))n$ is an approximation of the size of graphic TSP within a factor of $2 - \frac{1}{7c_0} + \delta$.*

Proof. Let $\hat{T} = (2 - \frac{2}{7}(\hat{\alpha} - 2\hat{\beta}))n$. Since $\hat{\beta} \geq \beta$ and $\hat{\alpha} \leq \alpha$, by Lemma 4.2.9, $T \leq \hat{T}$.

Then we prove that $\hat{T} \leq (2 - \frac{1}{7c_0} + \delta)T$. By Lemma 4.2.7 and Lemma 4.2.8, $T \geq \max\{(2 - 2\alpha)n, (1+\beta)n - 2\}$, which means

$$(2 - \frac{1}{7c_0} + \delta)T \geq (2 - \frac{1}{7c_0})\max\{(2-2\alpha)n, (1+\beta)n\} - 4 + \delta n$$

On the other hand, $\hat{T} \leq (2 - \frac{2}{7}(\frac{\alpha}{c_0} - 2\beta))n + \frac{6}{7}\delta n$ since $c_0\hat{\alpha} + \delta \leq \alpha$ and $\hat{\beta} \leq \beta + \delta$. For sufficient

large n, we have $\delta n - 4 \geq \frac{6}{7}\delta n$, so it is sufficient to prove that $\frac{2-\frac{2}{7}(\frac{\alpha}{c_0}-2\beta)}{\max\{2-2\alpha,1+\beta\}} \leq 2 - \frac{1}{7c_0}$ for any $0 \leq \alpha, \beta \leq 1$ and $c_0 \geq 1$.

Let $\gamma = \frac{\alpha}{c_0} - 2\beta$, $1+\beta = 1+(\frac{\alpha}{c_0}-\gamma)/2$, so if we fix γ, $\max\{2-2\alpha, 1+\beta\}$ is minimized when $2-2\alpha = 1+(\frac{\alpha}{c_0}-\gamma)/2$. In this case $\alpha = \frac{(2+\gamma)c_0}{4c_0+1}$ and $\max\{2-2\alpha, 1+\beta\} = \frac{4c_0+2}{4c_0+1} - \frac{2c_0}{4c_0+1}\gamma$. If $\gamma \leq \frac{1}{2c_0}$,

$$\begin{aligned}\frac{2-\frac{2}{7}(\alpha-2\beta)}{\max\{2-2\alpha,1+\beta\}} &\leq \frac{2-\frac{2}{7}\gamma}{\frac{4c_0+2}{4c_0+1} - \frac{2c_0}{4c_0+1}\gamma} = \frac{4c_0+1}{7c_0} - \frac{2-\frac{4c_0+2}{7c_0}}{\frac{4c_0+2}{4c_0+1} - \frac{2c_0}{4c_0+1}\gamma} \\ &\leq \frac{4c_0+1}{7c_0} + 2 - \frac{4c_0+2}{7c_0} = 2 - \frac{1}{7c_0}\end{aligned}$$

If $\gamma > \frac{1}{2c_0}$, $\frac{2-\frac{2}{7}(\alpha-2\beta)}{\max\{2-2\alpha,1+\beta\}} < \frac{2-\frac{1}{7c_0}}{1} = 2 - \frac{1}{7c_0}$ since $\beta \geq 0$. So $\hat{T} \leq (2 - \frac{1}{7c_0} + \delta)T$. □

By Lemma 4.2.10, we immediately have the following theorem.

Theorem 4.6. *For any $\delta > 0$ and $c_0 \geq 1$. Given a graph G with maximum matching size αn, suppose there is an algorithm that uses pair queries, runs in t time, and with probability at least $2/3$, outputs an estimate of the maximum matching size $\hat{\alpha}n$ such that $\hat{\alpha} \leq \alpha \leq c_0\hat{\alpha} + \delta$. Then there is an algorithm that approximates the cost of graphic TSP of G to within a factor of $2 - \frac{1}{7c_0} + \delta$, using distance queries, in $t + \tilde{O}(n/\delta^2)$ time with probability at least $3/5$.*

Proof. We first use the algorithm in the assumption to obtain an estimate $\hat{\alpha}n$ of the size of maximum matching αn. The following analysis is based on the event that this algorithm is run successfully, which has probability $2/3$.

We then sample $N = \frac{100}{\delta^2}$ vertices. For each sampled vertex v, we first query the distance between v and every vertex in G to obtain the degree of v. If v has degree 2, suppose u and w are the neighbors of v. We query the distance from u and w to every vertex in G. By Lemma 4.2.6, v is an articulation point if and only if there is no vertex v' such that $|d(u,v') - d(w,v')| \leq 1$. So we can check if v is a bad vertex with $O(n)$ distance queries and time. Suppose there are βn bad vertices in G and $(\hat{\beta} - \delta/2)N$ sampled vertices are bad. By

Chernoff bound, the probability that $\left|\beta - \hat{\beta} + \delta/2\right| > \delta/2$ is at most $2e^{\frac{\delta^2 N^2}{16}} < 1/15$. We analyze the performance based on the event that $\beta \leq \hat{\beta} \leq \beta + \delta$.

By Lemma 4.2.10, $(2 - \frac{2}{7}(\hat{\alpha} - 2\hat{\beta}))$ is a $(2 - \frac{1}{7c_0} + \delta)$ approximation of the size of graphic TSP of G. The probability of failure is at most $1/3 + 1/15 = 2/5$. □

Proof of Theorem 4.1: Behnezhad [44] gives an algorithm for matching size estimation that only uses *pair queries* – given a pair of vertices, is there an edge between them? Note that any pair query can be simulated by a single query to the distance matrix in a graphic TSP instance.

Theorem 4.7 ([44])**.** *For any $\varepsilon > 0$, there is an algorithm that uses pair queries, runs in $\tilde{O}(n/\varepsilon^2)$ time, and with probability $2/3$, outputs an estimate of the size of a maximal matching within an additive error εn.*

Substituting the above result in Theorem 4.6 and using the fact that a maximum matching has size at most twice the size of a maximal matching (setting $c_0 = 2$, and $\delta = \varepsilon$), we obtain Theorem 4.1.

4.2.3. An $O(n)$ Space $(\frac{11}{6})$-Approximate Streaming Algorithm for Graphic TSP

We show here that our approach for obtaining a sublinear-time algorithm for graphic TSP can be extended to the insertion-only streaming model to obtain for any $\varepsilon > 0$, an $(\frac{11}{6} + \varepsilon)$-approximate estimate of the graphic TSP cost using $O(n/\varepsilon^2)$ space, proving Theorem 4.2. In the streaming model, we assume that the input to graphic TSP is presented as a sequence of edges of the underlying graph G. Any algorithm for this model, clearly also works if instead the entries of the distance matrix are presented in the stream instead – an entry that is 1 corresponds to an edge of G, and it can be ignored otherwise as a non-edge.

Given a stream containing edges of a graph $G(V, E)$, our algorithm performs the following two tasks in parallel:

- Find a maximal matching M in G – let αn denote its size.

- Estimate the number of bridges in the maximal matching M, say βn, to within an additive error of εn.

The algorithm outputs $(2 - \frac{2}{3}(\alpha - \beta))n$ as the estimated cost of graphic TSP of G.

In an insertion-only stream, it is easy to compute a maximal matching M using $O(n)$ space: we start with M initialized to an empty set, and add a new edge (u, v) into the matching M iff neither u nor v are already in M. It is also easy to check if an edge e is a bridge in insertion-only stream with $O(n)$ space. We can do this by maintaining a disjoint-set data structure. Whenever an edge arrives (other than e), we merge the connected components of its endpoints. If there is only one component remaining at the end of the stream, then e is not a bridge, and otherwise, e is a bridge.

To estimate the number of bridges in the maximal matching, we sample $N = 100/\varepsilon^2$ edges in the matching, and run in parallel N tests where each test determines whether or not the sampled edge is a bridge. We use $O(n/\varepsilon^2)$ space in total since we sample $N = O(1/\varepsilon^2)$ edges. Suppose there are $\bar{\beta}$ sampled edges are bridges, then by Chernoff bound, $\hat{\beta}n = \frac{\bar{\beta}|M|}{N}$ is an approximation of βn to within additive error εn with probability at least $9/10$.

As stated, this gives us a two-pass algorithm: the first pass for computing the matching M, and the second pass for estimating the number of bridges in M. However, we can do both these tasks in parallel in a single pass as follows: at the beginning of the stream, we start the process of finding connected components of graph G. Whenever an edge e is added to M, if $|M| < N$, then we create a new instance I_e of the connectivity problem that ignores the edge e. This clearly allows us to test whether or not e is a bridge. Once $|M| > N$, then whenever an edge e is added to M, with probability $\frac{N}{|M|}$, we drop uniformly at random an existing instance, say $I_{e'}$ of connectivity, and create a new instance I_e of connectivity that only ignores edge e (we insert back the edge e' into I_e). Since there are at most N instances of connectivity that are running in parallel, the algorithm uses $O(nN) = O(n/\varepsilon^2)$ space.

We now prove that the algorithm gives a good approximation of the cost of graphic TSP.

Lemma 4.2.11. *If a graph G has a maximal matching M of size αn, and there are βn edges in M that are bridges in G, then the cost of graphic TSP in G is at most $(2 - \frac{2}{3}(\alpha - \beta))n$, and at least $\frac{6}{11}(2 - \frac{2}{3}(\alpha - \beta))n$.*

Proof. Since there are at least $(\alpha - \beta)n$ edges in the matching M that are not a bridge, by Lemma 4.2.2 and Lemma 4.2.5, the cost of graphic TSP of G is at most $(2 - \frac{2}{3}(\alpha - \beta))n$.

On the other hand, since M is a maximal matching of G, the size of maximum matching of G is at most $2\alpha n$. By Lemma 4.2.7, the cost of graphic TSP is at least $(2 - 4\alpha)n$. Graph G also contains at least βn bridges, so by Lemma 4.2.3, the cost of graphic TSP is also at least $(1 + \beta)n$.

To prove the lemma, it is sufficient to prove that for any $0 \leq \beta \leq \alpha \leq 1$, we have $2 - \frac{2}{3}(\alpha - \beta) \leq \frac{11}{6}\max\{1 + \beta, 2 - 4\alpha\}$. Let $\gamma = \alpha - \beta$. $1 + \beta = 1 + \alpha - \gamma$. So $\max\{1 + \beta, 2 - 4\alpha\} \geq 2 - 4(\frac{1}{5}(1 + \gamma)) = \frac{6}{5} - \frac{4}{5}\gamma$. If $\gamma \leq \frac{1}{4}$, $\frac{2 - \frac{2}{3}(\alpha - \beta)}{\max\{1+\beta, 2-4\alpha\}} \leq \frac{2 - \frac{2}{3}\gamma}{\frac{6}{5} - \frac{4}{5}\gamma} = \frac{5}{6} + \frac{5}{6 - 4\gamma} = \frac{11}{6}$. If $\gamma > \frac{1}{4}$, $2 - \frac{2}{3}(\alpha - \beta) < \frac{11}{6}$, while $\max\{1 + \beta, 2 - 4\alpha\} \geq 1$ since $\beta > 0$. □

By Lemma 4.2.11, the expression $(2 - \frac{2}{3}(\alpha - \beta))n$ gives us an 11/6-approximate estimate to the cost of graphic TSP of G. Since we can exactly compute α and approximate β with additive error ε in a single-pass streaming algorithm that uses $O(n/\varepsilon^2)$ space, we have the following theorem:

Theorem 4.8. *For any $\varepsilon > 0$, there is a single-pass randomized streaming algorithm that estimates the cost of graphic TSP of G to within a factor of $(\frac{11}{6} + \varepsilon)$, in an insertion-only stream, using $O(n/\varepsilon^2)$ space with probability at least 9/10.*

4.3. An $\Omega(n^2)$ Query Lower Bound for Approximation Schemes

In this section, we prove that there exists an $\varepsilon_0 > 0$, such that any query algorithm for graphic or $(1, 2)$-TSP that returns a $(1 + \varepsilon_0)$-approximate estimate of optimal cost, requires $\Omega(n^2)$ queries. In order to prove this, we design a new query model for the 3SAT problem and show an $\Omega(n^2)$ query lower bound for 3SAT in this model. We then use a reduction from

3SAT to $(1,2)$-TSP in [224] to prove the lower bound for $(1,2)$-TSP; with some additional changes, we also get an identical lower bound for graphic TSP.

The idea of proving query lower bound for APX-hard problems by reduction from 3SAT is similar to the idea used in [60], and we follow their general approach. However, in [60], the authors study lower bounds for problems in sparse graphs and hence the query model uses only neighbor queries. So in their query model, the lower bound for 3SAT is $\Omega(n)$. In order to prove an $\Omega(n^2)$ query lower bound in the pair query model, we need to design a new query model for 3SAT.

In the 3SAT problem, we are given a 3CNF instance on n variables, and the goal is to estimate the largest fraction of clauses that can be satisfied by any assignment. The algorithm is allowed to perform only one kind of query: is a variable x present in a clause c? If the answer is yes, then the algorithm is given the full information about all variables that appear in the clause c. The proof of the next theorem is deferred to Section 4.3.3.

Theorem 4.9. *For any $\varepsilon > 0$, any algorithm that with probability at least $2/3$ distinguishes between satisfiable 3CNF instances and 3CNF instances where at most $(7/8+\varepsilon)$ fraction of clauses can be satisfied, needs $\Omega(n^2)$ queries.*

4.3.1. Reduction from 3SAT to $(1,2)$-TSP

We will utilize an additional property of the hard instances of 3SAT in Theorem 4.9, namely, each variable occurs the same constant number of times where the constant only depends on ε. We denote the number of variables by n, the number of clauses by m, and the number of occurrences of each variable by k; thus $m = kn/3$.

We use the reduction in [224] to reduce a 3SAT instance to a $(1,2)$-TSP instance. In this reduction, there is a gadget for each variable and for each clause. Each of these gadgets has size at most $L = \Theta(k^2)$. Thus the $(1,2)$-TSP contains N vertices where $N \leq L(n+m) = \frac{L(k+3)n}{3}$. Let G_{x_j} be the gadget of variable x_j and G_{c_i} be the gadget of clause c_i. There is a ground graph which is the same for each 3SAT instance. Each variable gadget is

connected with the gadgets for clauses that contain that variable. The reduction satisfies the following property. If the 3SAT instance is satisfiable, then the $(1,2)$-TSP instance contains a Hamilton cycle supported only on the weight 1 edges. On the other hand, if at most $m - \ell$ clauses can be satisfied in the 3SAT instance, the $(1,2)$-TSP cost is at least $N + \lceil \ell/2 \rceil$. Thus there is a constant factor separation between the optimal $(1,2)$-TSP cost in the two cases. However, what remains to be shown is that any query algorithm for $(1,2)$-TSP can also be directly simulated on the underlying 3SAT instance with a similar number of queries. The theorem below now follows by establishing this simulation.

Theorem 4.10. *There is a constant ε_0 such that any algorithm that approximates the $(1,2)$-TSP cost to within a factor of $(1+\varepsilon_0)$ needs $\Omega(n^2)$ queries.*

Proof. We consider the following stronger queries for $(1,2)$-TSP: for any query (u,v), if u is in a vertex gadget G_{x_j} and v is in a clause gadget G_{c_i} (or vice versa) and x_j occurs in c_i in the 3SAT instance, then the algorithm is given all the edges incident on G_{c_i}. Otherwise the algorithm just learns if the there is an edge between u and v.

Let $\varepsilon = 1/16$, and let the values of k, L and N correspond to this choice for ε according to the redution in Section 4.3.1. Let $\varepsilon_0 = \frac{k}{32(k+3)L}$. Consider the $(1,2)$-TSP instance reduced from the 3SAT instance generated by the hard distribution in Theorem 4.9 with $\varepsilon = 1/16$. If the 3SAT instance is perfectly satisfiable, then the $(1,2)$-TSP instance has a Hamilton cycle of cost N. If the 3SAT instance satisfies at most $(15/16)$-fraction of clauses, then each Hamilton cycle in the $(1,2)$-TSP instance has cost at least

$$N + (1/8 - \varepsilon)m/2 = N + (1/8 - \varepsilon)kn/6 \geq (1 + \frac{(1/8-\varepsilon)k}{2(k+3)L})N = (1+\varepsilon_0)N$$

For any query (u,v) in the $(1,2)$-TSP instance, we can simulate it by at most one query in the corresponding 3SAT instance as follows: if u is in a vertex gadget G_{x_j} and v is in a clause gadget G_{c_i} (or vice versa), then we make a query of x_j and c_i in the 3SAT instance.

If the 3SAT query returns YES and the full information of c_i, then we return all the edges incident on G_{c_i} according to the reduction rule and the full information of c_i. If the 3SAT query returns NO or (u, v) are not in a vertex gadget and a clause gadget respectively, we return YES if (u, v) is an edge in the ground graph and NO otherwise.

By Theorem 4.9, any algorithm that distinguishes a perfectly satisfiable 3SAT instance from an instance where at most (15/16)-fraction of the clauses can be satisfied needs $\Omega(n^2)$ queries. So any algorithm that distinguishes a $(1, 2)$-TSP instance containing a Hamilton cycle of length N from an instance that has minimum Hamilton cycle of cost $(1 + \varepsilon_0)N$ needs $\Omega(n^2)$ queries.

□

4.3.2. $\Omega(n^2)$ Lower Bound for Graphic TSP

We can reduce an instance of $(1, 2)$-TSP to an instance of graphic TSP by adding a new vertex that is adjacent to all other vertices. By doing so, any pair of vertices in the new graph has a distance at most 2. On the other hand, the cost of graphic TSP in the new graph differs by at most 1 from the cost of $(1, 2)$-TSP in the old graph. So the $\Omega(n^2)$ query lower bound for $(1, 2)$-TSP also holds for the graphic TSP problem.

4.3.3. An $\Omega(n^2)$ Query Lower Bound for the 3SAT Problem

We first prove a lower bound of E3LIN2 problem. E3LIN2 is the problem of deciding the satisfiability of a system of linear equations modulo 2, with three variables per equation.

We consider the following query model: the algorithm can query if an equation contains a variable. If the answer is **YES**, then the algorithm is also given all the variables and the right-hand side of the equation.

Theorem 4.11. *For any $\varepsilon > 0$, any algorithm that distinguishes between a perfectly satisfiable E3LIN2 instance and an instance that satisfies at most $(1/2 + \varepsilon)$-fraction of equations needs $\Omega(n^2)$ queries with probability at least $2/3$.*

We start by defining the hard distribution. The distribution is similar to the one in [60], but the query model and therefore the proof are different. Every hard instance has n variables $x_1, x_2, \ldots x_n$ and $m = kn$ equations $e_1, e_2, \ldots, e_m$ for some positive integer k. We construct the following two distributions of E3LIN2.

- The distribution $\mathcal{D}_{NO}$ is the distribution of NO-instance, and is generated as follows: We first generate a random permuation $\sigma : [1, 3m] \to [1, 3m]$. For each $1 \leq i \leq m$, we assign equation e_i the variables $y_1^i = x_{\lceil \frac{\sigma(3i-2)}{3k} \rceil}$, $y_2^i = x_{\lceil \frac{\sigma(3i-1)}{3k} \rceil}$ and $y_3^i = x_{\lceil \frac{\sigma(3i)}{3k} \rceil}$. The equation e_i is $y_1^i + y_2^i + y_3^i = \pm z_i$ where z_i is choosen to be $+1$ or -1 uniformly randomly.

- The distribution $\mathcal{D}_{YES}$ is the distribution of YES-instance, and is generated as follows: We first assign the variables to each equation with the same process as $\mathcal{D}_{NO}$. Then we randomly choose an assignment of vaiables, say $A^\star$. Finally, for each equation e_i, we set $y_1^i + y_2^i + y_3^i = z_i$ where z_i equals the sum of $y_1^i + y_2^i + y_3^i$ according to assignment $A^\star$.

Our final distribution generates an instance from the NO-distribution with probability $1/2$ and an instance from the YES-distribution with probability $1/2$.

If the instance is generated by $\mathcal{D}_{YES}$, then it is satisfied by the assignment $A^\star$. The following lemma proves that if the instance is generated by $\mathcal{D}_{NO}$, then with high probability, the at most $(1/2 + \varepsilon)$-fraction of the equations can be satisfied.

Lemma 4.3.1. *For any $\varepsilon > 0$, there exists a positive integer k, such that if an instance of E3LIN2 is randomly chosen from $\mathcal{D}_{NO}$ with n variables and $m = kn$ equations, then with probability $9/10$, at most $(1/2 + \varepsilon)$-fraction of the equations can be satisfied.*

Proof. Let $k = 8/\varepsilon^2$ and so $m = \frac{8n}{\varepsilon^2}$. Fix an assignment A. For each equation e_i, the probability that A satisfies e_i is $1/2$. Since in distribution $\mathcal{D}_{NO}$, the right hand side of the equations are sampled independently, the event that A satisfies any equation is independent

of the event of A satisfying any subset of the other equations. By the Chernoff bound, the probability that A satisfies at least $(1/2+\varepsilon)$-fraction of equations is at most $e^{-\frac{\varepsilon^2(m/2)}{4}} \leq e^{-n}$. Taking the union bound over all possible assignments A, the probability that there exists an assignment that satisfies at least $(1/2+\varepsilon)$-fraction of equations is at most $2^n \cdot e^{-n} < 1/10$. □

Now we prove that it is hard to distinguish between the YES and NO instances of this distributions. Define a bipartite graph G_σ associated with the random permutation σ as follows: there are $3m$ vertices on each side of G_σ, there is an edge between the i^{th} vertex on the left and the j^{th} vertex on the right if and only if $\sigma_i = j$. Since σ is chosen uniformly at random, G_σ is a randomly chosen perfect matching. Associate variable x_i with the $(3k(i-1)+1)^{th}$ to the $(3ki)^{th}$ vertices on the left and associate equation e_j with the $(3j-2)^{th}$ to the $(3j)^{th}$ vertex on the right. A variable occurs in an equation if and only if there is an edge between the vertices associate with the variable and the equation.

Fix an algorithm $\mathcal{A}$, let $\mathcal{E}^{\mathcal{A}}_{YES}$ and $\mathcal{E}^{\mathcal{A}}_{NO}$ be the set of equations given to $\mathcal{A}$ after all the queries to an instance generated by $\mathcal{D}_{YES}$ and $\mathcal{D}_{NO}$. Denote the knowledge graph $G^{\mathcal{A}}$ as the subgraph of G_σ induced by the equations given to $\mathcal{A}$ and the variables that occur in these equations. The following lemma shows that if an algorithm only discover a small fraction of equations, then the set of equations discovered by the algorithm has the same distribution in the YES and NO cases with some high constant probability.

Lemma 4.3.2. *For any $k > 0$, there exists a constant δ_0 such that: if $G^{\mathcal{A}}$ contains at most $3\delta_0 n$ edges, then the distributions of $\mathcal{E}^{\mathcal{A}}_{YES}$ and $\mathcal{E}^{\mathcal{A}}_{NO}$ are identical with probability at least 9/10.*

The proof of Lemma 4.3.2 is similar to the proof of Theorem 8 in [60]. We prove that the left hand side of the equations in $\mathcal{E}^{\mathcal{A}}_{YES}$ and $\mathcal{E}^{\mathcal{A}}_{NO}$ are independent, and thus the distribution of the right hand side are identical.

Proof. We first prove that there is a constant δ_0 such that with probability at least 9/10,

any set of equations of size $\delta n \leq \delta_0 n$ contains more than $\frac{3}{2}\delta n$ variables. Fix a set of variables V of size $\frac{3}{2}\delta n$. For any equation e, the probability that it contains only the variables in V is $\frac{4.5k\delta n}{3kn} \cdot \frac{4.5k\delta n-1}{3kn-1} \cdot \frac{4.5k\delta n-2}{3kn-2} \leq 4\delta^3$. For any equation e and any set of equations $\mathcal{E}$ that does not contain e, the events that e only contains variables in V and the equations in $\mathcal{E}$ only contain variable in V are negatively correlated. So for any set of equations of size δn, the probability that these equations only contain variables in V is at most $(4\delta^3)^{\delta n} = 4^{\delta n}\delta^{3\delta n}$. Taking the union bound over all possible set of equations of size δn, the probability that one of them only contains variables in V is at most $4^{\delta n}\delta^{3\delta n} \cdot \binom{kn}{\delta n} \leq 4^{\delta n}\delta^{3\delta n} \cdot (ek/\delta)^{\delta n} = (4ek)^{\delta n}\delta^{2\delta n}$. We now take the union bound over all sets of variables of size $\frac{3}{2}\delta n$; the probability that there exists a set of equations of size δn which only contains $\frac{3}{2}\delta n$ variables is at most $(4ek)^{\delta n}\delta^{2\delta n} \cdot \binom{n}{1.5\delta n} \leq (4ek)^{\delta n}\delta^{2\delta n} \cdot (\frac{2e}{3\delta})^{1.5\delta n} \leq (3e^{2.5}k)^{\delta n}\delta^{0.5\delta n} = (40k\sqrt{\delta})^{\delta n} \leq (40k\sqrt{\delta_0})^{\delta n}$. Let $\delta_0 < \frac{1}{11(40k)^2}$, and taking union bound over all possible sizes i ranging from 1 to $\delta_0 n$, the probability that any set of equations of size $i \leq \delta_0 n$ contains more than $\frac{3}{2}i$ variables is at least $1 - \sum_{i=1}^{\delta_0 n}(\frac{1}{11})^i \geq 9/10$.

So with probability at least 9/10, any set of equations with size $i \leq \delta_0 n$ contains more than $\frac{3}{2}i$ variables, which means there is at least one variable that occurs at most once in these equations by the pigeonhole principle. We prove that under this event, the distribution of $\mathcal{E}^{\mathcal{A}}_{YES}$ and $\mathcal{E}^{\mathcal{A}}_{NO}$ are identical if $G^{\mathcal{A}}$ contains at most $3\delta_0 n$ edges.

Notice that the left hand side of of $\mathcal{E}^{\mathcal{A}}_{YES}$ and $\mathcal{E}^{\mathcal{A}}_{NO}$ are always identical, we only need to prove that the distributions of the right hand side are identical when $G^{\mathcal{A}}$ has at most $3\delta_0 n$ edges. In this case there are at most $\delta_0 n$ equations in $\mathcal{E}^{\mathcal{A}}_{YES}$ since each equation is associated with 3 vertices. Let the right hand sides of $\mathcal{E}^{\mathcal{A}}_{YES}$ and $\mathcal{E}^{\mathcal{A}}_{NO}$ be vectors b_{YES} and b_{NO} respectively. We prove the distributions of b_{YES} and b_{NO} are identical by induction on the size of b_{YES} (which is also the number of equations in $\mathcal{E}^{\mathcal{A}}_{YES}$).

The base case is when there is no equation in $\mathcal{E}^{\mathcal{A}}_{YES}$ at all (which means the algorithm does not discover any equation). In this case, both b_{YES} and b_{NO} are empty vectors.

In the induction step, $|b_{YES}| = |b_{NO}| > 0$. Since the number of equations is at most $\delta_0 n$, there exists a variable v that only occurs once. Without loss of generality, suppose it occurs in the last equation. Let b'_{YES} and b'_{NO} be the vector obtained by deleting the last entry of b_{YES} and b_{NO} respectively. By induction hypothesis, the distributions of b'_{YES} and b'_{NO} are identical. Moreover, v only occurs in the last equation and only occurs once, the distribution of the last entry of b_{YES} is uniform, independent of the other entries, so is the last entry of d_{NO}. So the distributions of b_{YES} and b_{NO} are identical. □

Next we prove that in order to discover a constant fraction of equations, we need $\Omega(n^2)$ queries.

Lemma 4.3.3. *For any $\delta_0 > 0$, there exists a $\delta_1 > 0$ such that: for any algorithm that makes at most $\delta_1 n^2$ queries, $G^{\mathcal{A}}$ contains at most $3\delta_0 n$ edges with probability $9/10$.*

The proof of Lemma 4.3.3 is similar to the proof of Theorem 5.2 in [18] and we will prove it later.

Proof of Theorem 4.11. For any $\varepsilon > 0$, let k, δ_0, δ_1 be the constant defined in Lemma 4.3.1, Lemma 4.3.2, Lemma 4.3.3 respectively. Consider two instance I_{YES} and I_{NO} generated as follows: we generate the instance I_{YES} by distribution $\mathcal{D}_{YES}$, then let the left hand side of I_{NO} be the same as the left hand side of I_{YES}, generate the right hand side of I_{NO} uniformly independently for each equation. Since the process of generating the left hand side is the same for $\mathcal{D}_{YES}$ and $\mathcal{D}_{NO}$, the distribution of I_{NO} is indeed $\mathcal{D}_{NO}$. By Lemma 4.3.1, with probability $9/10$, the I_{NO} satisfies at most $(1/2+\varepsilon)$-fraction of equations. By Lemma 4.3.3, if an algorithm makes at most $\delta_1 n^2$ queries, then it discovers at most $\delta_0 n$ equations with probability $9/10$. Base on this event, by Lemma 4.3.2, the equations discovered by the algorithm has the same probability of being generated by $\mathcal{D}_{YES}$ and by $\mathcal{D}_{NO}$. By the union bound, with probability at most $7/10$, I_{NO} is an instance that satisfies at most $(1/2+\varepsilon)$-fraction of the equations and the algorithm cannot distinguish between I_{YES} and I_{NO}. □

We use the following standard reduction from Equation to 3SAT in [151]. Given a set of equations $\mathcal{E}$, we construct a 3CNF formula $\Phi = F(\mathcal{E})$ as follows: For any equation $X_i + X_j + X_k = 1$ in $\mathcal{E}$, we add four clauses $(X_i \vee X_j \vee X_k)$, $(X_i \vee \bar{X}_j \vee \bar{X}_k)$, $(\bar{X}_i \vee X_j \vee \bar{X}_k)$ and $(\bar{X}_i \vee \bar{X}_j \vee X_k)$ into Φ; for any equation $X_i + X_j + X_k = 0$ in $\mathcal{E}$, we add four clauses $(\bar{X}_i \vee X_j \vee X_k)$, $(X_i \vee \bar{X}_j \vee X_k)$, $(X_i \vee X_j \vee \bar{X}_k)$ and $(\bar{X}_i \vee \bar{X}_j \vee \bar{X}_k)$ into Φ. It is clear that if an assigment satisfies an equation in $\mathcal{E}$, then it also satisfies all of the four corresponding clauses in Φ. Otherwise it satisfies three of the four corresponding clauses. So we have the following lemma.

Lemma 4.3.4. *For any $\gamma \in (0, 1]$, given a set of equations $\mathcal{E}$ and its corresponding 3CNF formula $\Phi = F(\mathcal{E})$, for any assignment A, A satisfies γ-fraction of equations in $\mathcal{E}$ if and only if A satisfies $(3/4 + \gamma/4)$-fraction of clauses in Φ.*

Proof of Theorem 4.9. For any 3CNF formula generated from a E3LIN2 instance $\mathcal{E}$ we consider a stronger type of query model for 3SAT. For any query between a variable and a clause, if the variable occurs in the clause, then the algorithm is not only given the entire clause, but also the other 3 clauses corresponding to the same equation in $\mathcal{E}$. The new query is equivalent to the query in E3LIN2. □

4.3.4. Proof of Lemma 4.3.3

The proof of Lemma 4.3.3 is similar to the proof of Theorem 5.2 in [18]. However, the argument from [18] cannot be used in a black-box manner. So, here we present a complete proof. The following lemma from [18] is useful.

Lemma 4.3.5 (Lemma 5.4 of [18])**.** *Let $G'(L' \cup R', E')$ be an arbitrary bipartite graph such that $|L'| = |R'| = N$, and each vertex in G' has degree at least $2N/3$. Then for any edge $e = (u, v) \in E'$, the probability that e is contained in a perfect matching chosen uniformly at random in G' is at most $3/N$.*

Denote the vertex sets in bipartite graph G_σ as L and R. We have $|L| = |R| = 3kn$. Suppose

whenever a query finds a variable inside an equation, the algorithm is not only given the equation, but also edges incident on the vertices associated with the equation in G_σ. Then, $G^{\mathcal{A}}$ contains exactly those edges that are given to algorithm in response to the queries.

The query process can be viewed as the task of finding $\delta_0 n$ edges in G_σ by the following queries between a variable x_i and an equation e_j: query if there is at least one edge between U and V where $U \subset L$ is the set of vertices associated with x_i and $V \subset R$ is the set of vertices associated with e_j. If so, the algorithm is given all edges incident on the vertices in V. To prove the lemma, we only need to prove that finding $3\delta_0 n$ edges in G_σ in this model needs $\Omega(n^2)$ queries.

For simplicity, we consider the following query model instead: a query asks if there is an edge between a pair of vertices u and v. If so, the algorithm is given all three edges incident on the vertices associated with the same equation as v. Any original query can be simulated by $3k \cdot 3 = 9k$ new queries. So it is sufficient to prove that we need $\Omega(n^2)$ queries in the new model.

We say that an edge (u, v) in G_σ has been *discovered* if the edge is given to the algorithm. After t queries have been made by the algorithm, let $L_U(t) \subseteq L$ and $R_U(t) \subseteq R$ denote the set of undiscovered vertices in L and R respectively. Let $E(t) \subseteq L_U(t) \times R_U(t)$ denote the set of edge slots that have not yet been queried/discovered. Note that by our process for generating G_σ, the undiscovered edges correspond to a random perfect matching between $L_U(t)$ and $R_U(t)$ that is entirely supported on $E(t)$.

We will analyze the performance of any algorithm by partitioning the queries into phases. The first query by the algorithm starts the first phase, and a phase ends as soon as three edges in G_σ have been discovered. Let Z_i be a random variable that denotes the number of queries performed in phase i of the algorithm. Thus we wish to analyze $\mathbb{E}\left[\sum_{i=1}^{\delta_0 n} Z_i\right]$.

For any vertex $w \in (L_U(t) \cup R_U(t))$, we say that the *uncertainty* of w is d if there are at least d edge slots in $E(t)$ that are incident on w.

At time t, we say a vertex $w \in (L_U(t) \cup R_U(t))$ is *bad* if the uncertainty of w is less than $2.5kn$. Note that if at some time t none of the vertices in $(L_U(t) \cup R_U(t))$ are bad then in the next $nk/2$ time steps, the degree of any vertex in $L_U(t) \cup R_U(t)$ in $E(t)$ remains above $2kn$ if there is no successful query. Thus by Lemma 4.3.5, the probability that any query made during the first $nk/2$ queries in the phase succeeds (in discovery of a new edge in G_σ) is at most $3/(3kn) = 1/(kn)$.

We say a phase is *good* if at the start of the phase, there are no bad vertices, and the phase is *bad* otherwise.

Proposition 4.3.6. *The expected length of a good phase is at least $nk/4$.*

Proof. If at the start of the phase i, no vertex is bad, then for the next $nk/2$ time steps, the probability of success for any query is at most $1/(kn)$. Thus the expected number of successes (discovery of a new edge in G_σ) in the first $nk/2$ time steps in a phase is at most $1/2$. By Markov's inequality, it then follows that with probability at least $1/2$, there are no successes among the first $nk/2$ queries in a phase. Thus the expected length of the phase is $\geq nk/4$. □

Note that if all phases were good, then it immediately follows that the expected number of queries to discover $3\delta_0 n$ edges is $\Omega(n^2)$. To complete the proof, it remains to show that most phases are good. For ease of analysis, we will give the algorithm additional information for free and show that it still needs $\Omega(n^2)$ queries in expectation even to discover the first $3\delta_0 n$ edges in G_σ.

Whenever the algorithm starts a bad phase, we immediately reveal to the algorithm an undiscovered edge (u, v) in G_σ (as well as other two edges incident on the vertices associated with the same equation as v) that is incident on an arbitrarily chosen bad vertex. Thus each bad phase is guaranteed to consume a bad vertex (i.e., make the bad vertex discovered and hence remove it from further consideration). On the other hand, to create a bad vertex w,

one of the following two events needs to occur: the number of discovered edges in G_σ plus the number of queries is at least $3kn - 2.5kn = kn/2$.

Since we are restricting ourselves to analyzing the discovery of first $\delta_0 n$ edges in G_σ, any vertex w that becomes bad requires at least $kn/2$ queries incident on it. Thus to create K bad vertices in the first $\delta_0 n$ phases, we need to perform at least $(K \cdot (kn/2))/2$ queries; here the division by 2 accounts for the fact that each query reduces uncertainty for two vertices. It now follows that if the algorithm encounters at least $\delta_0 n/2$ bad phases among the first δ_0 phases, then $K \geq \delta_0 n/2$ and hence it must have already performed $\delta_0 kn^2/8$ queries. Otherwise, at least $\delta_0 n/2$ phases among the first δ_0 phases are good, implying that the expected number of queries is at least $(\delta_0 n/2) \cdot (nk/4) = \Omega(n^2)$. This completes the proof of Lemma 4.3.3 with Markov's inequality.

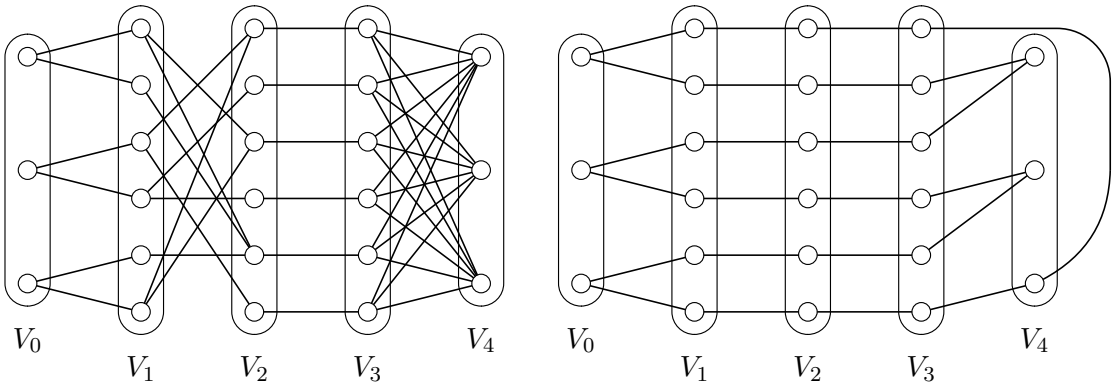
4.4. A Reduction from Matching Size to TSP Cost Estimation

In this section, we give a reduction from the problem of estimating the maximum matching size in a bipartite graph to the problem of estimating the optimal $(1, 2)$-TSP cost. An essentially identical reduction works for graphic TSP cost using the idea described in Section 4.3.2.

We will denote the size of the largest matching in a graph G by $\alpha(G)$. Given a bipartite graph $G(V, E)$ with n vertices on each side, we construct an instance $G'(V', E')$ of the $(1, 2)$-TSP problem on $4n$ vertices such that the optimal TSP cost on G' is $5n - \alpha(G)$. Thus for any $\varepsilon \in [0, 1/5)$, any algorithm that can estimate $(1, 2)$-TSP cost to within a $(1 + \varepsilon)$-factor, also gives us an estimate of the matching size in G to within an additive error of $5\varepsilon n$.

We will now describe our construction of the graph G'. For clarity of exposition, we will describe G' as the graph that contains edges of cost 1 – all other edges have cost 2. Suppose the vertex set V of G consists of the bipartition $V_1 = \{v_1^1, v_2^1, \ldots, v_n^1\}$ and $V_2 = \{v_1^2, v_2^2, \ldots, v_n^2\}$. We construct the graph G' as follows: we start with the graph G, then add three sets of vertices V_0, V_3 and V_4, such that $V_0 = \{v_1^0, v_2^0, \ldots, v_{n/2}^0\}$ with $n/2$ vertices, $V_3 = \{v_1^3, v_2^3, \ldots, v_n^3\}$

with n vertices, and $V_4 = \{v_1^4, v_2^4, \ldots, v_{n/2}^4\}$ with $n/2$ vertices. The graph G' will only have edges between V_j and V_{j+1} ($j = \{0, 1, 2, 3\}$). We will denote the set of edges between V_j and V_{j+1} as $E_{j,j+1}$. For any vertex $v_i^0 \in V_0$, it connects to v_{2i-1}^1 and v_{2i}^1 in V_1. $E_{1,2}$ has the same edges as the edges in G. Each vertex $v_i^2 \in V_2$ is connected to vertex v_i^3 in V_3, that is, vertices in V_2 and V_3 induce a perfect matching (identity matching). Finally, each vertex in V_3 is connected to all the vertices in V_4. See Figure 4.1(a) for an illustration.



(a) The illustration of G'.

(b) The illustration of tour T, where V_2 and V_3 are arranged with order $(v_{f(1)}^2, \ldots, v_{f(6)}^2)$ and $(v_{f(1)}^3, \ldots, v_{f(6)}^3)$.

Figure 4.1: Illustration of the reduction for $n = 6$.

The lemmas below establish a relationship between matching size in G and $(1, 2)$-TSP cost in G'.

Lemma 4.4.1. *Let M be any matching in G. Then there is a $(1, 2)$-TSP tour T in G' of cost at most $5n - |M|$.*

Proof. Let $f : [n] \to [n]$ be any bijection from $[n]$ to $[n]$ such that whenever a vertex v_i^1 is matched to a vertex v_j^2 in M, then $f(i) = j$. Consider the following $(1, 2)$-TSP tour T: each vertex $v_i^0 \in V_0$ connects to v_{2i-1}^1 and v_{2i}^1 in T; each vertex $v_i^1 \in V_1$ connects to $v_{\lceil (i+1)/2 \rceil}^0$ and $v_{f(i)}^2$ in T. For any $v_{f(i)}^2 \in V_2$, it connects to v_i^1 and $v_{f(i)}^3$ in T. For any vertex $v_{f(i)}^3 \in V_3$, if $i > 1$, it connects to $v_{f(i)}^2$ and $v_{\lceil i/2 \rceil}^4$ in T; if $i = 1$, it connects to $v_{f(i)}^2$ and $v_{n/2}^4$ in T. See Figure 4.1(b) as an illustration. T is clearly a TSP-tour.

All edges in T are also edges in G' except for possibly some edges between V_1 and V_2. If v_i^1 is matched in M, then $(v_i^1, v_{f(i)}^2)$ is an edge in G', otherwise it is not in G' and thus has weight 2. So T only has $n - |M|$ weight 2 edges, which means T has cost at most $4n + n - |M| = 5n - |M|$. □

Lemma 4.4.2. *For any $(1,2)$-TSP tour T in G, T has cost at least $5n - \alpha(G)$.*

To prove Lemma 4.4.2, we first prove an auxiliary claim.

Claim 4.4.3. *Suppose $G = (V_1, V_2, E)$ is a bipartite graph which has maximum size $\alpha(G)$. For any 2-degree subgraph H of G, if there are at most X vertices in V_1 has degree 2 in H, then there are at most $\alpha(G) + X$ vertices in V_2 which have degree at least 1 in H. Similarly, if there are at most X vertices in V_2 has degree 2 in H, then there are at most $\alpha(G) + X$ vertices in V_1 which have degree at least 1 in H.*

Proof. If there are at most X vertices in V_1 has degree 2 in H. We construct H' by deleting an arbitrary edge on each degree 2 vertex in V_1, then construct H'' by deleing an arbitrary edge on each degree 2 vertex in V_2. Since H'' does not have degree two vertex, it is a matching of G. So the number of degree 1 vertices in V_2 in H'' is at most $\alpha(G)$. On the other hand, any vertex in V_2 which has degree at least 1 in H' also has degree 1 in H''. So there are at most $\alpha(G)$ vertices of degree at least 1 in V_2 in H'. Furthermore, since there are only X vertices of degree 2 in V_1 in H, we delete at most X edges in H when constructing H'. So H' has at most X more isolate vertices in V_2 than in H, which means H has at most $\alpha(G) + X$ vertices with degree at least 1 in V_2.

The second part of the claim follows via a similar argument as the first part of the claim. □

Proof of Lemma 4.4.2. Let a_{01} be the number of edges in $T \cap E_{0,1}$, $a_{2,3}$ be the number of edges in $T \cap E_{3,4}$. Let G^X be the intersection graph of G and T. Since the vertices in V_0 only connect to the vertices in V_1 in G', and any vertex in T has degree 2, there are at least $n - a_{01}$ edges incident on V_0 in T are not an edge in G'. On the other hand, since

any vertex in V_1 is incident on at at most 1 edge in $E_{0,1}$, there are at least a_{01} vertices in V_1 is connected to a vertex in V_0 in T, which means there are at most $n - a_{01}$ vertices in V_1 has degree 2 in G^X. By Claim 4.4.3, there are at most $n - a_{01} + \alpha(G)$ vertices in V_2 has edge in G^X. For any isolate vertex in V_2 in T, it has only one edge in G' connecting to V_3, so this vertex must incident on an edge in T which is not in G'. So there are at least $n - (n - a_{01} + \alpha(G)) = \alpha(G) - a_{01}$ edges incident on V_2 in T which is not in G'.

There are $2n$ edges incident on V_3 in T, but among them, there are only a_{23} edges between V_2 and V_3 which is also in G', and there are at most n edges between V_3 and V_4 in T since each vertex has degree only 2. So there are at least $2n - n - a_{23} = n - a_{23}$ edges incident on V_3 which is not in G'. On the other hand, since any vertex in V_2 is incident on at at most 1 edge in $E_{2,3}$, there are at least a_{23} vertices in V_2 is connected to a vertex in V_3 in T, which means there are at most $n - a_{23}$ vertices in V_2 has degree 2 in G^X. By Claim 4.4.3, there are at most $n - a_{23} + \alpha(G)$ vertices in V_1 has edge in G^X. For any isolate vertex in V_1 in T, it has only one edge in G' connecting to V_0, so this vertex must incident on an edge in T which is not in G'. So there are at least $n - (n - a_{23} + \alpha(G)) = a_{23} - \alpha(G)$ edges incident on V_1 in T which is not in G'.

Since any edge has two endpoints, the number of edges in T but not in G' is at least $((n - a_{01}) + (a_{01} - \alpha(G)) + (n - a_{23}) + (a_{23} - \alpha(G)))/2 = n - \alpha(G)$, which means T has cost at least $4n + n - \alpha(G) = 5n - \alpha(G)$. □

Corollary 4.4.4. *For any $\varepsilon \in [0, 1/5)$, any algorithm that can estimate $(1,2)$-TSP cost to within a $(1+\varepsilon)$-factor, can be used to estimate the size of a largest matching in a bipartite graph G on $2n$ vertices to within an additive error of $5\varepsilon n$.*

Proof. We use the reduction above to construct a $(1,2)$-TSP instance G' on $4n$ vertices. By Lemma 4.4.1 and Lemma 4.4.2, the optimal TSP cost for G' is $5n - \alpha(G)$. We now run the $(1+\varepsilon)$-approximation algorithm for $(1,2)$-TSP on graph G' (note that the reduction can be simulated in each of neighbor query model, pair query model, and the streaming model

without altering the asymptotic number of queries used). Suppose the output is C which satisfies $(1-\varepsilon)(5n-\alpha(G)) \leq C \leq (1+\varepsilon)(5n-\alpha(G))$, which means $5n-\alpha(G)-5\varepsilon n < C < 5n-\alpha(G)+5\varepsilon n$. Let $\hat{\alpha} = 5n - C$, we have $\alpha(G) - 5\varepsilon n < \hat{\alpha} < \alpha(G) + 5\varepsilon n$. □

4.5. Additional Lower Bound Results for Approximating Graphic and $(1,2)$-TSP Cost

In this section, we prove several additional lower bounds on approximating the costs of graphic TSP and $(1,2)$-TSP. Many of these results involve constructing a simple distribution on graphs where some graphs in the support of the distribution have TSP tours of cost close to n while others have cost close to $2n$. We show that no deterministic algorithm can distinguish between these two types of instances, and then invoke Yao's principle [253] to prove lower bounds for randomized algorithms. When the graphs in the distribution have diameter 2, the graphic TSP instances are also instances of the $(1,2)$ TSP problem. Using this approach we show an $\Omega(n)$ lower bound for both metric TSP and $(1,2)$-TSP costs in our query model.

In the standard graph query model allowing both pair queries and neighbor queries, we show a stronger lower bound of $\Omega(\varepsilon^2 n^2)$ for randomized algorithms that estimate the cost of graphic TSP to within a factor of $(2-\varepsilon)$. This shows that the distance query model is strictly more powerful for estimating graphic TSP cost.

Using Dirac's theorem about the existence of Hamilton Cycles in very dense graphs, we show an $\Omega(n^2)$ lower bound for deterministic algorithms to get any approximation better than 2. For the problem of finding a $(2-\varepsilon)$-approximate tour, rather than just estimating its cost, we show an $\Omega(\varepsilon n^2)$ lower bound for both graphic TSP and $(1,2)$-TSP.

Finally, we show a space lower bound of $\Omega(\varepsilon n)$ for approximating Graphic TSP to within $2-\varepsilon$ in the streaming model.

4.5.1. An $\Omega(n)$ Query Lower Bound for $(2-\varepsilon)$-approximating $(1,2)$-TSP and Graphic TSP Cost

In this subsection, we show that in our query model, any randomized algorithm that approximates the cost of minimum graphic TSP or $(1,2)$-TSP to within a factor of $2-\varepsilon$ for any ε, we need $\Omega(n)$ queries. As stated above, it suffices to create a distribution over $(n+1)$-vertex graphs such that any deterministic algorithm requires $\Omega(n)$ queries to check if the cost of minimum $(1,2)$-TSP or graphic TSP is $n+1$ or $2n$ on this distribution.

The distribution is generated as follows: we start with a "star" graph whose vertices set is $\{v_0, v_1, v_2, \ldots, v_n\}$ where v_0 is connected to all other vertices. Then we pick a random permutation π over $[n]$. With probability half, we connect $v_{\pi(i)}$ and $v_{\pi(i+1)}$, for $1 \leq i \leq n-1$. In this case, the resulting graph is the wheel graph. With probability half we do not join successive vertices in π, and the resulting graph is a star graph. Since v_0 is connected to all other vertices, any two vertices have distance 1 or 2. So graphic TSP and $(1,2)$-TSP are the same in this distribution.

Lemma 4.5.1. *A wheel graph admits a TSP tour of cost $n+1$ while any TSP tour in a star graph has cost at least $2n$.*

Proof. In a wheel graph, the tour $(v_0, v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)}, v_0)$ has cost $n+1$ since all edges are weight 1. For any tour in a star graph, only the edges incident on v_0 have weight 1. So the cost of the tour is at least $2 + 2(n-1) = 2n$. □

Lemma 4.5.2. *If an algorithm only makes $n/3$ queries, then with probability at least $1/3$, the answer to all these queries is the same in a wheel graph and a star graph.*

Proof. For any query (v_i, v_j), if one of v_i or v_j is v_0, then the answer is 1 in both cases. If none of v_i or v_j is v_0, then the answer is 2 if the graph is the star graph. If the graph is a wheel graph, then the answer is 1 only if i and j are adjacent to each other in π, which has probability at most $2/n$. By union bound, with probability at least $1/3$, the answers of all

these queries are the same in both cases. □

By Lemma 4.5.1 and Lemma 4.5.2, we have the following lower bound for graphic TSP and $(1,2)$-TSP problem in the distance query model.

Theorem 4.12. *For any $\varepsilon > 0$, in the distance query model, any algorithm that with probability at least $2/3$ approximates the cost of $(1,2)$-TSP or graphic TSP within a factor of $(2-\varepsilon)$ requires $\Omega(n)$ queries.*

4.5.2. An $\Omega(\varepsilon^2 n^2)$ Query Lower Bound for $(2-2\varepsilon)$-approximating Graphic TSP in Standard Graph Query Model

In this subsection, if an algorithm for graphic TSP is given only access to the underlying graph G via standard graph queries, namely, pair queries, degree queries and neighbor queries, then any randomized algorithm for approximating graphic TSP cost to within a factor of $2-\varepsilon$ for any $\varepsilon > 0$, requires $\Omega(\varepsilon^2 n^2)$ queries. Again by Yao's principle, it suffices to create a distribution over n-vertex graphs such that any deterministic algorithm requires $\Omega(\varepsilon^2 n^2)$ queries to distinguish between graphs where the cost of graphic TSP is n and graphs where the graphic TSP cost is at least $(2-2\varepsilon)n - 1$.

We start with a graph G with three parts: a path P with $(1-2\varepsilon)n$ vertices, and two cliques C_1 and C_2 of size εn. Let $u_1, u_2, \ldots, u_{\varepsilon n}$ be the vertices in C_1, and $v_1, v_2, \ldots, v_{\varepsilon n}$ be the vertices in C_2 . Connect all vertices in C_1 to an endpoint of P, and connect all vertices in C_2 to the other endpoint of P. For any vertex u_i (resp. v_i), we say the j^{th} neighbor of u_i (resp. v_i) is u_j (resp. v_j) for any $j \neq i$, and the i^{th} neighbor is the endpoint of P. For any vertex in P, we pick an arbitrary order of its neighbors. With probability half, we change the graph to create a YES case as follows: we pick two different indices i and j from $[\varepsilon n]$ randomly. We change the j^{th} neighbor of u_i and v_i to be v_j and u_j respectively and the i^{th} neighbor of u_j and v_j to be v_i and u_i respectively. Otherwise, we do not change the graph and say we are in NO case.

Lemma 4.5.3. *If we are in the YES case, then the cost of graphic TSP is n. Otherwise, the*

cost of the graphic TSP is at least $(2-2\varepsilon)n-1$.

Proof. If we are in the YES case, consider the tour that starts at u_i, and goes through the vertices in C_1 in arbitrary order (but not visiting u_j immediately after u_i), then goes to the endpoint of P that connects to all vertices in C_1, goes through the path P, then visits the vertices in C_2 in arbitrary order ending with v_j (but not visiting v_i right before v_j), and finally goes back to u_i. All edges in this tour have weight 1. So the cost of this tour is n.

If we are in the NO case, then all the edges in the path are bridges. So by Lemma 4.2.3, the cost of graphic TSP is at least $n+(1-2\varepsilon)n-1=(2-2\varepsilon)n-1$. □

Lemma 4.5.4. *If an algorithm only makes* $\varepsilon^2 n^2/4$ *queries, then with probability at least* $1/3$*, the answer to these queries are the same in the* YES *and* NO *cases.*

Proof. The degree of the vertices are the same in both cases. So, any neighbor query has the same answer. For any pair query or neighbor query, all queries on the vertices in P also have the same answer. For any query on the vertices in the cliques, we say a query is querying a pair of indices (k,ℓ) if it is a pair query between two vertices with indices k and ℓ, or it is a neighbor query that queries the k^{th} (resp. ℓ^{th}) neighbor of u_ℓ or v_ℓ (resp. u_k or v_k). A pair query or a neighbor query has different answers in YES and NO cases only when it is querying the indices i and j that we picked when generating the YES case. Since i and j are chosen randomly, the probability that a query is querying i and j is $\frac{2}{\varepsilon n(\varepsilon n-1)}$. If the algorithm only make $\varepsilon^2 n^2/4$ queries, the probability that there is a query with different answers in YES case and NO case is at most $\frac{\varepsilon^2 n^2}{2\varepsilon n(\varepsilon n-1)} < 2/3$ by union bound. □

By Lemma 4.5.3 and Lemma 4.5.4, we have the following lower bound for graphic TSP problem in the standard query model.

Theorem 4.13. *For any* $\varepsilon>0$*, if an algorithm approximates the cost of graphic TSP within a factor of* $(2-\varepsilon)$ *with probability at least* $2/3$ *using only degree queries, neighbor queries*

and pair queries, then it requires $\Omega(\varepsilon^2 n^2)$ *queries.*

4.5.3. An $\Omega(n^2)$ Query Lower Bound for Deterministic Algorithms for $(1,2)$-TSP and Graphic TSP

In this subsection, we prove that in our stronger, distance query model, any deterministic algorithm that approximates cost of graphic TSP or $(1,2)$-TSP within a factor of $(2-\varepsilon)$ needs $\Omega(\varepsilon n^2)$ queries.

We first consider the $(1,2)$-TSP problem. We prove that for any $\varepsilon n^2/5$ queries, even if all the answers are that the distance is 2, the graph could still have a TSP of cost $n+\varepsilon n$.

Consider the graph H whose edge set is pairs of vertices that have not been queried. Since there are only $\varepsilon n^2/5$ queries, there are at least $(1-\varepsilon)n$ vertices that have been queried at most $2n/5 < (1-\varepsilon)n/2-1$ times. These vertices has degree at least $(n-1)-((1-\varepsilon)n/2-1) = (1+\varepsilon)n/2$ in H. Let V_0 be an arbitrary set that contains exactly $(1-\varepsilon)n$ of these vertices. The subgraph of H induced by V_0 has minimum degree at least $(1+\varepsilon)n/2-\varepsilon n = (1-\varepsilon)n/2 = |V_0|/2$. By the following well-known theorem due to Dirac about the existence of Hamilton cycles in dense graphs, there is a Hamilton cycle in the subgraph of H induced by V_0.

Lemma 4.5.5 (Dirac [103])**.** *Any n-vertex graph G where each vertex has degree at least $n/2$ has a Hamilton cycle.*

So G has a path of length $(1-\varepsilon)n$ that only contains weight one edges, any TSP tour obtained by expanding this path has length at most $(1-\varepsilon)n+2\varepsilon n = (1+\varepsilon)n$. Thus it is possible that G contains a tour of cost $(1+\varepsilon)n$ after $\varepsilon n^2/5$ queries.

For graphic TSP problem, we use the same trick as in Section 4.3, adding a vertex that connects to all other vertices. This results in the same lower bound for graphic TSP as for $(1,2)$-TSP.

Theorem 4.14. *Any deterministic algorithm that approximates the cost of graphic TSP or $(1,2)$-TSP to within a factor of $(2-\varepsilon)$ using distance queries needs $\Omega(\varepsilon n^2)$ queries.*

4.5.4. An $\Omega(\varepsilon n^2)$ Lower Bound for Finding a $(2-\varepsilon)$-Approximate $(1,2)$-TSP or Graphic TSP Tour

While our focus in this paper has been on estimating the cost of $(1,2)$-TSP or graphic TSP to within a factor that is strictly better than 2, we show here that if the goal were to output an approximate $(1,2)$-TSP tour or graphic TSP tour (not just an estimate of its cost), then even with randomization, any algorithm requires $\Omega(\varepsilon n^2)$ distance queries to output a $(2-\varepsilon)$-approximate solution for any $\varepsilon > 0$. We start by showing this lower bound for $(1,2)$-TSP.

We create a distribution over n-vertex graphs with $(1,2)$-TSP cost $(1+o(1))n$ such that with a large constant probability, any deterministic algorithm requires $\Omega(\varepsilon n^2)$ queries to output a tour that contains at least $3\varepsilon n$ weight-1 edges .

We generate the graph G with n vertices $\{v_1, v_2, \ldots, v_n\}$ as follows: we first generate a random permutation $\pi : [n] \to [n]$. For any $i \neq j$, if $\pi(i) = j$, then v_i and v_j are connected in G.

By construction of G, it consists of vertex disjoint cycles, and each cycle in G corresponds to a cycle in permutation π. Since the expected number of cycles in a random permutation is equal to the n^{th} harmonic number, which is $O(\log n)$ [134], G has a cycle cover with $O(\log n)$ cycles in expectation. By Markov's inequality, the number of cycles in G is $o(n)$ with probability $1 - o(1)$. If we break these cycles into paths and link them in arbitrary order, we obtain a tour of cost at most $n + o(n)$. So the cost of $(1,2)$-TSP of G is $(1+o(1))n$ with probability $1 - o(1)$.

Next, we prove that any algorithm needs $\Omega(n)$ queries to find εn edges. Construct a graph H that only contains a perfect matching such that the i^{th} vertex on the left is matched to the j^{th} vertex on the right if and only if $\pi(i) = j$.

Consider the problem of finding the edges in H by pair queries. Each pair query in G can be simulated by at most 2 pair queries in H. Furthermore, any tour in G corresponding to a

perfect matching between the vertices in H. So to prove the lower bound in $(1,2)$-TSP, we only need to prove that any algorithm that output a perfect matching between the vertices in H contains at most $3\varepsilon n$ edges in H.

The following lemma follows from the arguments in [18] (also similar to the arguments in Section 4.3.4) about the lower bound for finding edges in a random perfect matching.

Lemma 4.5.6 (Section 5.2 in [18]). *Any algorithm needs $\Omega(\varepsilon n^2)$ queries to find εn edges in a random perfect matching with sufficiently large constant probability.*

Finally, we prove that if an algorithm only find εn edges in H, then any output matching contains $\varepsilon n + o(1)$ edges in H with large constant probability. Suppose the algorithm only makes $\varepsilon(1-\varepsilon)(1-2\varepsilon)n^2/3$ queries, then there are at most $\varepsilon(1-\varepsilon)n$ vertices on the left (resp. right) being queried at least $(1-\varepsilon)n/4$ times. Let V_0 be the set of vertices v in H such that the edge incident on v is not found by the algorithm and both v and its neighbor are not queried $(1-2\varepsilon)n/3$ times. V_0 contains at least $(1-\varepsilon)^2 n > (1-2\varepsilon)n$ vertices. Let H_0 be the subgraph of H induced by V_0. In H_0, each vertex v has $\frac{3}{4}|V_0|$ vertices u on the other side such that the algorithm does not query the pair (u,v).

By Lemma 4.3.5, each pair of vertices in V_0 contains an edge with probability $O(\frac{1}{n})$. So for any perfect matching between the vertices in H, any edge incident on a vertices in V_0 is also in H with probability only $O(\frac{1}{n})$. So there are $o(n)$ edges incident on the vertices in V_0 that are also edges in H with probability $1-o(1)$ by Markov's inequality. So the perfect matching contains at most $2\varepsilon n + o(n) < 3\varepsilon n$ edges in H with probability $1-o(1)$, which implies the same lower bound for $(1,2)$-TSP.

For graphic TSP problem, we use the same trick as in Section 4.3 of adding a vertex that connects to all vertices to prove the same lower bound as for $(1,2)$-TSP.

Theorem 4.15. *Any algorithm that output a graphic TSP or $(1,2)$-TSP tour within a factor of $(2-\varepsilon)$ using distance query with with sufficiently large constant probability needs $\Omega(\varepsilon n^2)$*

queries.

4.5.5. An $\Omega(\varepsilon n)$ Lower Bound for $(2-\varepsilon)$ Approximation of Graphic TSP in the Streaming Model

In this subsection, we prove that any single-pass streaming algorithm that approximates the cost of graphic TSP in insertion-only streams to within a factor of $(2-\varepsilon)$ with probability at least $2/3$ requires $\Omega(\varepsilon n)$ space.

To prove the lower bound for single-pass streaming algorithm, it is sufficient to prove the lower bound in the one-way communication model. The graphic TSP problem in the communication model is the two-player communication problem in which the edge set E of a graph $G(V, E)$ is partitioned between Alice and Bob, and their goal is to approximate the cost of the graphic TSP of G.

We prove the lower bound by a reduction from the Index problem, In Index, Alice is given a bit-string $x \in \{0,1\}^N$, Bob is given an index $k^\star \in [N]$, and the goal is for Alice to send a message to Bob so that Bob outputs $x_{k^\star}$. It is well-known that any one-way communication protocal that solves Index with probability $2/3$ requires $\Omega(N)$ bits of communication [189].

We use the Index problem with size $N = \varepsilon n/4$. We will construct a graph G such that the cost of graphic TSP is at most $n + 2N$ if $x_{k^\star} = 1$ and at least $2n - N - 1$ if $x_{k^\star} = 0$. Since $N = \varepsilon n/4$, in order to approximate the cost of graphic TSP within a factor of $(2-\varepsilon)$, Alice and Bob need to be able to check if the cost of graphic TSP is larger than $2n - 2N$ or less than $n + 2N$.

Reduction:

Given an instance of Index with size $N = \varepsilon n/4$:

1. Alice and Bob construct the following graph $G(V, E)$ with no communication: The vertex set V is a union of three set P, U, W, where $P = \{v_1, v_2, \ldots, v_{n-2N}\}$, $U = \{u_1, u_2, \ldots, u_N\}$ and $W = \{w_1, w_2, \ldots, w_N\}$. In Alice's graph, all vertices in P form a path, whose endpoints are v_1 and v_{n-2N}, for any $i \in [N]$, there is an edge between u_i

and w_i. Furthermore, v_1 connects to all u_i such that $x_i = 1$ in her input in the Index instance. In Bob's graph, v_1 connects to all w_i for $i \neq k^\star$ and $w_{k^\star}$ connects to v_{n-2N} instead.

2. Alice and Bob then approximate the cost of graphic TSP of the graph G using the best protocol. Bob outputs $x_{k^\star} = 0$ if the cost of graphic TSP is larger than $2n - 2N$ and outputs $x_{k^\star} = 1$ otherwise.

The communication cost of this protocol is at most as large as the communication complexity of the protocol used to solve graphic TSP. Now we prove the correctness of the reduction.

Lemma 4.5.7. *If $x_{k^\star} = 1$, then the cost of graphic TSP of G is at most $n+2N$. If $x_{k^\star} = 0$, then the cost of graphic TSP of G is at least $2n - N - 1$.*

Proof. If $x_{k^\star} = 1$, consider the tour that first visits the path in P from v_1 to v_{2n-2N}, then visits $w_{k^\star}, u_{k^\star}$, then visits w_i, u_i for each $i \neq k^\star$ in arbitrary order, and finally goes back to v_1. Since $x_{k^\star} = 1$, $u_{k^\star}$ connects to v_1. Also for each $i \neq k^\star$, $w_{k^\star}$ connects to v_1, so for any i and $j \neq k^\star$, u_i and w_j have distance at most 3. Since any other edge in the tour has weight 1, the cost of the tour is at most $n + (3-1) \cdot N = n + 2N$.

If $x_{k^\star} = 0$, both $u_{k^\star}$ and $w_{k^\star}$ do not connect to v_1. So $u_{k^\star}$, $w_{k^\star}$ and v_{n-2N} form a block in G. For any $i \neq k^\star$, both u_i and w_i do not connect to v_{n-2N}. So u_i, w_i and v_1 forms a block in G. Furthermore, all edges in the path are bridges in G. By Lemma 4.2.3, the cost of graphic TSP of G is at least $n + N + (n - 2N) - 1 = 2n - N - 1$. □

Theorem 4.16. *For any $\varepsilon > 0$, any single-pass streaming algorithm that is able to approximate the cost of graphic TSP of an input graph G within a factor of $2 - \varepsilon$ in insertion-only streams with probability at least $2/3$ requires $\Omega(\varepsilon n)$ space.*

Proof. Let Π be any 1/3-error one-way protocol that approximates graphic TSP within a factor of $(2 - \varepsilon)$. By Lemma 4.5.7, we obtain a protocol for Index that errs with probability

at most $1/3$ and has communication cost at most equal to cost of Π. By the $\Omega(N)$ lower bound on the one-way communication complexity of Index, we obtain that communication cost of Π must be $\Omega(N) = \Omega(\varepsilon n)$. The theorem now follows from this argument, as one-way communication complexity lower bounds the space complexity of single pass streaming algorithms. $\square$

CHAPTER 5

Near-linear Size Hypergraph Cut Sparsifier

In this chapter, we consider the problem of cut sparsification for hypergraphs. A hypergraph $H(V, E)$ consists of a vertex set V and a set E of hyperedges where each edge $e \in E$ is a subset of vertices. Kogan and Krauthgamer [185] initiated a study of this basic question and showed that given any weighted hypergraph H, there is an $O(mn^2)$ time algorithm to find a $(1 \pm \varepsilon)$-approximate cut sparsifier of H of size $\tilde{O}(\frac{nr}{\varepsilon^2})$ where r denotes the rank of the hypergraph. Similar to the case of graphs, the *size* of a hypergraph sparsifier refers to the number of edges in the sparsifier. Since r can be as large as n, in general, this gives a hypergraph cut sparsifier of size $\tilde{O}(n^2/\varepsilon^2)$, which is a factor of n larger than the Benczúr-Karger bound for graphs. Chekuri and Xu [88] designed a more efficient algorithm for building a hypergraph sparsifier. They gave a near-linear time algorithm in the total representation size (sum of the sizes of all hyperedges) to construct a hypergraph sparsifier of size $\tilde{O}(nr^2/\varepsilon^2)$ in hypergraphs of rank r, thus speeding up the run-time obtained in the work of Kogan and Krauthgamer [185] by at least a factor of n, but at the expense of an increased sparsifier size. It has remained an open question if the Benczúr-Karger bound is also achievable on hypergraphs, that is, do there exist hypergraph sparsifiers with $\tilde{O}(n/\varepsilon^2)$ edges? In this work, we resolve this question in the affirmative by giving a new polynomial-time algorithm for creating hypergraph sparsifiers of size $\tilde{O}(n/\varepsilon^2)$.

5.1. Main Results

We prove that any hypergraph has a $(1\pm\varepsilon)$-approximate cut sparsifier with $\tilde{O}(n)$ hyperedges.

Theorem 5.1. *Given a weighted hypergraph H, for any $0 < \varepsilon < 1$, there exists a randomized algorithm that constructs a $(1 \pm \varepsilon)$-approximate cut sparsifier of H of size $O(\frac{n \log n}{\varepsilon^2})$ in $\tilde{O}(mn + n^{10}/\varepsilon^7)$ time with high probability; here n denotes the number of vertices and m denotes the number of edges in the hypergraph.*

It is worth to note that the space bound $\tilde{O}(n^2)$ (each hyperedge may have size $\Theta(n)$) is also

the best possible to within a logarithmic factor due to a recent work [169].

As the number of hyperedges in a hypergarph could be expoential of the number of vertices n. We also consider the hypergraph cut sparsifier problem in sublinear models. First, we note that Theorem 5.1 also yields a $\tilde{O}(n^2/\varepsilon^2)$ space streaming algorithm for building a hypergraph sparsifier in a single-pass over an insertion-only stream. This can be done using a black-box technique for transforming cut sparsification algorithms into streaming algorithms whose space requirement is only slightly more than the sparsifier size (see Section 2.2 of [202]):

Lemma 5.1.1 ([202]). *Given an algorithm that finds a $(1\pm\varepsilon)$-approximate cut sparsifier of a hypergraph of size at most $f(n,\varepsilon)$ with high probability, there exists a single-pass insertion-only streaming algorithm to compute a $(1\pm\varepsilon)$-approximate cut sparsifier of size $2\log(m/n)\cdot f(n,\frac{\varepsilon}{2\log(m/n)})$ that stores at most $2\log^2(m/n)\cdot f(n,\frac{\varepsilon}{2\log(m/n)})$ hyperedges at any given time with high probability.*

Corollary 5.1.2. *For any $0<\varepsilon<1$, there exists a randomized insertion only streaming algorithm that constructs $(1\pm\varepsilon)$-approximate cut sparsifier of H of size $O(\frac{n\log n\log^3(m/n)}{\varepsilon^2})$ with high probability and stores only $O(\frac{n\log n\log^4(m/n)}{\varepsilon^2})$ hyperedges, and hence uses $O(\frac{n^2\log n\log^4(m/n)}{\varepsilon^2})$ space in the worst-case.*

The above result improves upon the $\tilde{O}(n^3/\varepsilon^2)$ space streaming algorithm in [185] for building hypergraph sparsifiers in insertion-only streams. We note here that for hypergraphs of *constant* rank, an $\tilde{O}(n/\varepsilon^2)$ space streaming algorithm is known [139] in dynamic streams where both insertion and deletion of hyperedges is allowed.

We also consider the prove in query model. Since we are working with hypergraphs, we first need to consider what type of queries are allowed. The most basic requirement is to have the ability to efficiently evaluate the size or weight of any cut in a given hypergraph. We assume here access to a *cut value oracle*, denoted as O_{value}, which takes as input a cut $C=(S,\bar{S})$, returns the size of the cut $|\delta_H(S)|$. This is akin to the standard assumption in submodular function minimization, namely, the algorithm has an oracle access to the value

of the submodular function on any set S since the cut function is a submodular function. However, as it turns out, it is easy to show that the access to a cut value oracle is provably not sufficient to construct a sparsifier, regardless of the time allowed as this oracle can not differentiate between hypergraphs where all edges have size 2 from hypergraphs where all edges have size 3[1]. So we also need a mechanism for accessing edges of the underlying graph. We thus introduce a second oracle, referred to as the *cut edge oracle*, denoted as O_{edge}, which takes as input a cut $C = (S, \bar{S})$, returns a random edge crossing the cut. Given access to both these oracles, we are indeed able to solve the problem of hypergraph sparsification in polynomial time in n.

Theorem 5.2. *Suppose we are given an unweighted hypergraph $H = (V, E)$ that can be accessed using the oracles O_{value} and O_{edge}. Then for any $0 < \varepsilon < 1$, a $(1 \pm \varepsilon)$-approximate sparsifier with $\tilde{O}(n/\varepsilon^2)$ hyperedges can be constructed in $O(n^{10}/\varepsilon^7)$ time, independent of the number of hyperedges.*

We complement the algorithmic result above by showing that just like the oracle O_{value} alone is not sufficient to achieve the result above, the oracle O_{edge} alone is also not sufficient to create a poly(n) size hypergraph sparsifier in poly(n) time.

Theorem 5.3. *There is no polynomial time randomized algorithm that can use O_{edge} queries alone to construct a $(1 \pm \varepsilon)$-approximate sparsifier of an underlying hypergraph H with probability better than $o(1)$.*

One may wonder if the oracle O_{edge} can be replaced with another access oracle that is used in sublinear algorithms for standard graphs, namely, ability to access the i_{th} neighbor of a vertex v for any integer i that is at most the degree of v. It is easy to see that this is essentially same as the ability to access a random edge incident on a vertex v. We can generalize this idea to the setting of hypergraphs as follows. A neighbor query oracle in a

[1]For instance, the cut value oracle can not distinguish between a copy of K_4 and the hypergraph that contains all possible hyperedges of size 3 on 4 vertices. Note that this does not rule out the possibility of efficiently constructing a data structure/sketch that can be used to answer cut queries. Our focus in this chapter, however, is on constructing sparsifiers, namely, sparse subgraphs of the original graph that preserve all cuts.

hypergraph takes as input a set $S \subseteq V$, and returns a random edge that contains all vertices in S if there is such an edge, and returns NIL if there is no edge. We say that a neighbor query is a *single vertex neighbor query* if $|S| = 1$, and it is a *vertex pair neighbor query* if $|S| = 2$. We denote the oracles that answer a single vertex neighbor query and a vertex pair neighbor query as O^1_{nbr} and O^2_{nbr} respectively. We next show that the oracle O_{edge} can be replaced with the oracle O^2_{nbr}, to obtain an alternate poly(n) time implementation of the result in Theorem 5.2.

Theorem 5.4. *Given an unweighted hypergraph $H = (V, E)$, suppose the algorithm can access the hypergraph using O_{value} and O^2_{nbr}, then for any $0 < \varepsilon < 1$, a $(1 \pm \varepsilon)$-approximate sparsifier with $\tilde{O}(n/\varepsilon^2)$ hyperedges can be constructed in $O(n^10/\varepsilon^7)$ time in n, independent of the number of hyperedges.*

In contrast to the result above, we show any algorithm that has access only to oracles O_{value} and O^1_{nbr}, requires exponentially many queries in the worst-case to construct a poly(n) size sparsifier.

Theorem 5.5. *There is no polynomial time randomized algorithm that can use O_{value} and O^1_{nbr} queries alone to construct a $(1 \pm \varepsilon)$-approximate sparsifier of an underlying hypergraph H with probability better than $o(1)$.*

5.2. Our Techniques

In this section, we breifly summerize the high-level ideas of Theorem 5.1 and Theorem 5.2.

5.2.1. High-level Ideas of Proving Theorem 5.1

First, We briefly describe the high-level idea behind the proof of Theorem 5.1. In the work of Benczúr and Karger [50], a graph sparsifier is constructed by sampling the edges with probabilities according to their *strengths*, a notion that captures the importance of an edge. Informally speaking, any edge that is among a small number of edges crossing some cut will have a high strength while any edge that does not participate in any small cuts will have a low strength. Once edges are sampled in this manner, a second key element in showing

that the (appropriately weighted) sampled graph approximately preserves *every cut* in the original graph, is to establish a cut counting bound which shows that there can not be too many cuts that are within a given factor of the minimum cut size in the graph. This allows use of a union bound over all cuts to show that every cut is well-approximated. Kogan and Krauthgamer [185] extend this elegant approach to constructing hypergraph sparsifiers. Similar to [50], they construct a hypergraph sparsifier by sampling hyperedges according to their strengths. A key point of divergence occurs in the second element, namely, the cut counting bound. As it turns out, number of cuts that are within a given factor of the minimum cut size, can be exponentially larger in the setting of hypergraphs[2]. To compensate for this increase in the number of cuts, their algorithm samples edges at roughly r times higher rate, resulting in a sparsifier of $\tilde{O}(nr)$ for hypergraphs of rank r. This size bound is essentially best possible by a direct execution of the Benczúr-Karger framework.

Our proof of Theorem 5.1 follows the high-level idea of creating a suitable probability distribution over hyperedges, and then sampling them in accordance with this distribution. However, we construct our hyperdge sampling distribution by analyzing the interaction among hyperdeges at a finer granularity. In particular, we start by constructing an auxiliary graph G where for each hyperedge e in H, we add a clique F_e whose vertex set is the same as the vertex set of the hyperedge e. The probability of sampling a hyperedge e in H is now determined by the strengths of the edges in the clique F_e. However, for this "sparsification-preserving coupling" between the graphs G and H to work, we can not directly use the graph G but instead need to create a *non-uniform* weight assignment to the edges in G that roughly ensures that the edges in F_e have similar strengths in G. In particular, for any hyperedge e, the edges in F_e may get assigned weights that now range from 0 to the weight

[2] As a simple example (derived from an example in [185]), consider a n-vertex hypergraph that contains a single hyperedge of size n with weight 1, as well as a clique on the n vertices such that each clique edge has weight $1/n^2$. It is easy to see that the weight of a minimum cut in this graph is $1 + (n-1)/n^2 \approx 1$. On the other hand, all possible $2^n - 1$ non-trivial partitions of the n vertices gives us a cut of size at most $3/2$. This is an exponential increase compared to the graph setting where it is known that the number of cuts that are at most twice as big as the minimum cut is bounded by $O(n^4)$ [174]. Note the $2^n - 1$ cuts created above not only correspond to distinct vertex partitions, but also have a distinct set of edges crossing them. Interestingly, the maximum number of distinct minimum cuts is the same in both graphs and hypergraphs, see, for instance, the work of Ghaffari, Karger, and Panigrahi [127].

of the hyperedge e. This weight assignment scheme, referred to as a *balanced assignment*, and an algorithm to compute it efficiently, are the key technical insights in our work. We note that the strategy of building sparsifiers of a hypergraph by the auxiliary graph G is also used in [29] where the authors use this strategy to construct spectral hypergraph sparsifier. Unlike our scheme, however, the work in [29] assigns uniform weights to the edges in F_e.

We conclude our overview by summarizing the three main technical steps involved in obtaining Theorem 5.1 by executing the high-level idea and described above. In the first step, we assign weights to the edges in G so that the edges in each clique F_e have similar strengths. In general, this task might be impossible, but we get around this by working with a weaker condition, namely, we only require that all edges in F_e that receive a positive weight have similar strengths. We design an iterative algorithm to achieve this goal, and show that it converges in polynomial time. In the second step, we prove that the hypergraph sparsifier constructed by sampling each hyperedge e according to the strengths of edges in F_e is indeed a good sparsifier for our input hypergraph. The proof of the second step follows the framework in [50] at a high-level but a key challenge is to couple together the performance of a sparsifier in H with the performance of a sparsifier in G. Together these two steps give us a polynomial-time algorithm for constructing a hypergraph sparsifier of size $\tilde{O}(n/\varepsilon^2)$. However, the running time of the resulting algorithm is quadratic in terms of m, the number of hyperedges. Since in a hypergraph, the number of edges m can be exponentially larger than n, in the third step, we present a way to speed up the algorithm so that the run-time has only a linear dependence on m.

5.2.2. High-level Ideas of Proving Theorem 5.2

At a high-level, graph and hypergraph sparsification algorithms work by estimating the importance of each edge in preserving cut sizes, and then sampling edges with probability proportional to their importance and assigning them an appropriately scaled weight. The main technical challenge in proving Theorem 5.2 is that the cut value oracle on the original graph cannot be used to estimate cut sizes in the vertex-induced subgraphs of the original

graph – a step that is implicit in determining importance of edges in preserving the cut structure. This issue does not arise in normal graphs where each edge contains 2 vertices, and the cut value oracle on the original graph indeed suffices to recover cut values in any induced subgraph. But once we consider hypergraphs with even edges of size 3, it is easy to show that the cut value oracle on the original graph can not distinguish between induced subgraphs that have minimum cut value 0 and induced subgraphs where the minimum cut value is polynomially large. We get around this issue by introducing for any subset of vertices X, a weaker notion of *pseudo cut size* for approximating cut sizes in the subgraph induced by X. The new cut size function remains submodular, and we show that it suffices to approximate the importance of each edge to within a factor n of its true importance. We then use the O_{edge} oracle to sample edges in accordance with their approximate importance. The resulting sparsifier H' has $\text{poly}(n)$ edges which we further sparsify to $\tilde{O}(n/\varepsilon^2)$ edges in $\text{poly}(n)$ time by applying Theorem 5.1 to H'.

5.3. Previous Results for Cut Sparsifier

In this section, we review some concepts and results that can be found in previous works on cut sparsifiers in standard graphs and hypergraphs, which also play important roles in our algorithm.

Definition 5.3.1. *Given a weighted graph G, a* k-strong component *of G is a maximal induced subgraph of G that has minimum cut at least k.*

Lemma 5.3.2 ([50])**.** *Given a weighted graph $G = (V, F, w)$ and some real number k, the k-strong components of G partition V. Given another real number $k' \geq k$, the k'-strong components of G are a refinement of the partition of k-strong components of G.*

Definition 5.3.3. *Given a weighted graph $G = (V, F, w)$ and an edge $f \in F$, the* strength *of f, denoted by k_f, in G is the maximum value of k such that f is contained in a k-strong component of G.*

Alternatively, the strength of an edge $f \in F$ is the largest minimum cut size among all

induced subgraphs $G[X]$ that contain f, where X ranges over all subsets of V. The following two claims give some properties of strength of edges in a graph.

Claim 5.3.4 (Corollary 4.9 in [51]). *Given a weighted graph G on n vertices, there are at most $n-1$ distinct values of edge strengths.*

Claim 5.3.5 (Lemma 4.11 in [51]). *For any weighted graph $G = (V, F, w)$ on n vertices, $\sum_{f \in F} \frac{w(f)}{k_f} \leq n - 1$.*

We can compute the strength of every edge in G by computing the global min-cut of $(n-1)$ induced subgraphs of G [51]. For the completeness of the argument, we prove the following lemma in Section 5.3.2.

Lemma 5.3.6. *Given a weighted graph G with n vertices and m edges. There is an algorithm that computes the strength of each edge in $\tilde{O}(mn)$ time with high probability.*

The following cut counting lemma due to Karger [174] gives an upper bound on the number of "small cuts" in a graph.

Lemma 5.3.7 (Corollary 8.2 in [174]). *Given a weighted graph $G = (V, F, w)$ with minimum cut size c, for all integers $\alpha \geq 1$, the number of cuts of the graph with weight at most αc is at most $|V|^{2\alpha}$. We will refer to such cuts as α-*cuts *throughout the paper.*

5.3.1. Strength Based Sampling Framework

In this section, we briefly review Benczúr and Karger's algorithm for graph sparsifiers [50, 51].

Given a graph $G = (V, F, w)$, they construct a sparsifier $\hat{G}$ as follows: for each edge $f \in F$, we include f in $\hat{G}$ with probability $p_f = \tilde{O}(\frac{w(f)}{k_f})$ (i.e. its weight over its strength). Every edge f that gets sampled is assigned a weight of $\hat{w}(f) = \frac{w(f)}{p_f}$ in $\hat{G}$. By Claim 5.3.5, the expected size of the sparsifier is $\tilde{O}(n)$. For any cut $C = (S, \bar{S})$ in the graph, the expected size of $\hat{w}(\delta_{\hat{G}}(S))$ is equal to $w(\delta_G(S))$. We need to give an upper bound of the probability that $\left|\hat{w}(\delta_{\hat{G}}(S)) - \mathbb{E}\left[\hat{w}(\delta_{\hat{G}}(S))\right]\right| > \varepsilon\, \mathbb{E}\left[\hat{w}(\delta_{\hat{G}}(S))\right]$. By concentration bounds, the larger the size of C, the lower probability that $\hat{w}(\delta_{\hat{G}}(S))$ is far from its expectation. By Lemma 5.3.7,

if a graph has minimum cut size c, for any integer α, the number of cuts of size at most αc is at most $n^{2\alpha}$. So we can group the cuts in different sizes based on this α value, take a union bound within each group, and then take a union bound over all groups to prove that with high probability, every cut in $\hat{G}$ has size close to its expectation. This gives a $(1 \pm \varepsilon)$-approximate cut sparsifier.

Recently, Kogan and Krauthgamer [185] generalized this approach to hypergraphs by defining an analogue of edge strengths for hyperedges. Most of the analysis for standard graphs also holds in the case of hypergraphs. The main difference is that in hypergraphs, the cut counting bound (Lemma 5.3.7) is no longer true. Instead, the authors prove that if the minimum cut size of a hypergraph is c, the number of cuts with size at most αc is $O(2^{\alpha r} n^{2\alpha})$ for any integer α, where r is the maximum cardinality of the edges in the hypergraph (see the footnote on page 2 for an example showing that an exponential dependence on r is necessary even for constant α). This increase in the number of α-cuts in turn requires edges to be oversampled at a rate that is $O(r)$ times higher, giving a hypergraph sparsifier of size $\tilde{O}(nr)$.

Theorem 5.6 ([185])**.** *Let H be a hypergraph with rank r, and let $\varepsilon > 0$ be an error parameter. Consider the hypergraph H' obtained by sampling each hyperedge e in H independently with probability $p_e = \min\{1, \frac{3((d+2)\log n + r)}{k_e \varepsilon^2}\}$, giving it weight $1/p_e$ if included. Then with probability at least $1 - O(n^{-d})$*

1. *the hypergraph H' has $O(\frac{n}{\varepsilon^2}(r + \log n))$ edges, and*
2. *H' is a $(1 \pm \varepsilon)$-approximate cut sparsifier of H.*

In fact, if for each edge e, the sampling proability p_e is at least $\frac{3((d+2)\log n + r)}{k_e \varepsilon^2}$, then the resulting graph is still a $(1 \pm \varepsilon)$-approximate cut sparsifier.

Lemma 5.3.8. *Let H be a hypergraph with rank r, and let $\varepsilon > 0$ be an error parameter. Consider the hypergraph H' obtained by sampling each hyperedge e in H independently with probability $p_e \geq \min\{1, \frac{3((d+2)\log n + r)}{k_e \varepsilon^2}\}$, giving it weight $1/p_e$ if included. Then with*

probability at least $1 - O(n^{-d})$*,* H' *is a* $(1 \pm \varepsilon)$*-approximate cut sparsifier of* H*.*

5.3.2. Computing Exact Edge Strengths

In this section, we give for completeness an algorithm that computes the *exact* strength of each edge in a graph and prove Lemma 5.3.6. Our algorithm will use as a subroutine the following global min-cut result of Karger [176]:

Theorem 5.7 ([176])**.** *Given a weighted graph* G *with* n *vertices and* m *edges, there is a randomized algorithm that finds the minimum cut in* $\tilde{O}(m)$ *time with high probability.*

The algorithm for computing exact edge strengths works as follows. We start by finding a minimum cut in the input graph G, and removing the edges in the minimum cut. We then repeat this process in each connected components, until the graph becomes an empty graph. Now for each edge in the graph, we output the strength of this edge as the largest min-cut value among all connected components containing this edge, that are encountered during the execution of the algorithm.

Lemma (Restatement of Lemma 5.3.6)**.** *Given a weighted graph* G *with* n *vertices and* m *edges, there is a randomized algorithm that computes the strength of each edge exactly in* $\tilde{O}(mn)$ *time with high probability.*

Proof. The above algorithm requires $(n-1)$ computations of global min-cut. Thus by Theorem 5.7, the total running time is $\tilde{O}(mn)$. We now prove that it correctly outputs exact edge strengths. We fix an edge e, let $\bar{k}_e$ denote the strength that our algorithm outputs for the edge e. It is clear that $\bar{k}_e \leq k_e$ since by the definition of $\bar{k}_e$, there is a subgraph of G which contains e and has min-cut size $\bar{k}_e$. To show that $\bar{k}_e \geq k_e$ also holds, consider the induced subgraph $G[X]$ which contains the edge e and has min-cut k_e. During the execution of our algorithm, let $G[\bar{X}]$ be the last connected component encountered which fully contains X. By our choice of $G[\bar{X}]$, the min-cut of $G[\bar{X}]$ must also cut through $G[X]$, which means that the cut size in this step is at least the min-cut size of $G[X]$, which is k_e. Thus by the definition of $\bar{k}_e$, the value of $\bar{k}_e$ is at least the size of min-cut of $G[\bar{X}]$ since

$G[\bar{X}]$ contains $G[X]$ which contains e. So $\bar{k}_e \geq k_e$. □

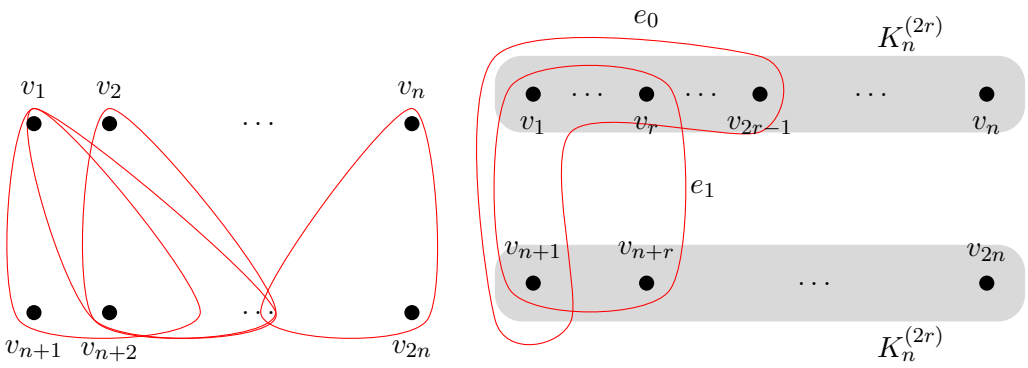
5.4. Construction of Near-linear Size Hypergraph Cut Sparsifiers

Similar to the previous works on graph/hypergraph sparsification, for each edge e in the hypergraph H, we will assign a probability p_e of sampling the edge in the sparsifier $\hat{H}$. If e is sampled, we give it weight $\frac{w_e}{p_e}$ in the sparsifier. However, unlike [185], our probabilities are not decided by the strength of the edge e in H. Instead, we derive these probabilities from edge strengths in an auxiliary standard graph G, where for each hyperedge e in H, we create a clique over the vertices of e in G such that the total weight of these clique edges is w_e. The hyperedge sampling probability p_e is derived from the strengths of the edges in the associate clique in G.

To prove that the sparsifier $\hat{H}$ is valid, we compare $\hat{H}$ to the Benczúr-Karger sparsifier $\hat{G}$ of G. For any cut C, it is not hard to see that the total weight of C in H is at least as large as the size of C in G. Consider the cut size in $\hat{H}$ as the sum of several random variables (each one representing an edge/hyperedge across the cut). By concentration bounds, the higher the probability mass of these random variables, the greater is the concentration of their sum, which means the variance of the size of C in $\hat{H}$ is at most its variance in $\hat{G}$. So we can use the cut-counting bound for standard graphs on $\hat{G}$ to analyze the concentration of the hypergraph sparsifier $\hat{H}$.

The approach of analyzing the performance of a hypergraph sparsifier through an auxiliary standard graph is also used in [29]. The authors use it to build a spectral sparsifier of a hypergraph. For a hyperedge e in H, like [29], a natural way of assigning its weight is to distribute its weight uniformly among all corresponding edges in G. However, this may cause the strengths of these edges in G to be very different. Two natural ways of assigning p_e are to let p_e to be decided by the maximum inverse strength of these edges or the average inverse strength. We can prove that deriving probabilities from the maximum inverse strength gives us small variance in cut sizes, while deriving probabilities from the average inverse strength results in a small number of sampled edges. However, the first approach may cause the

number of sampled edges to be too large and the second approach cannot guarantee that the variance of the cut sizes in $\hat{H}$ is small enough. The two examples below illustrate this.



(a) Example 5.1 (b) Example 5.2

Figure 5.1: Illustrations of Example 5.1 and Example 5.2. $K_n^{(2r)}$ refers to a copy of the complete $2r$-uniform hypergraph.

Example 5.1. *Consider the following hypergraph with $2n$ vertices $v_1, v_2, \ldots v_{2n}$: for any $1 \leq i \leq n$, we have all $\binom{n}{r-1}$ edges of size r containing v_i and $r-1$ vertices in $\{v_{n+1}, v_{n+2}, \ldots, v_{2n}\}$. Suppose we were to distribute the weight of each hyperedge uniformly in the auxiliary graph G, each edge in G has weight $1/\binom{r}{2} = O(1/r^2)$. For any $1 \leq i \leq n$, the weighted degree of v_i in the graph G is $O(1/r) \cdot \binom{n}{r-1}$, which means for each hyperedge, some of the edges in the associated clique in G have strength $O(1/r) \cdot \binom{n}{r-1}$. Hence if the hyperedges are sampled according to the minimum strength of the corresponding edges in G, each hyperedge will be sampled with probability $\frac{\Omega(r)}{\binom{n}{r-1}}$, and the expected number of edges in the sparsifier will be $\Omega(nr)$ since there are $n \cdot \binom{n}{r-1}$ hyperedges.*

Example 5.2. *Consider the following hypergraph with $2n$ vertices and hyperedge size $2r \leq \frac{n}{2}$: let $V = V_1 \cup V_2$ where $V_1 = \{v_1, \ldots, v_n\}$ and $V_2 = \{v_{n+1}, \ldots, v_{2n}\}$. The graph contains one hyperedge $e_0 = \{v_1, \ldots, v_{2r-1}, v_{n+1}\}$, and one hyperedge $e_1 = \{v_1, \ldots, v_r, v_{n+1}, \ldots, v_{n+r}\}$. There are also $\binom{n}{2r}$ hyperedges in V_1 and $\binom{n}{2r}$ hyperedges in V_2. Suppose we distribute the weight of each hyperedge uniformly in the auxiliary graph G. The cut size of $C = (V_1, V_2)$ is $\Theta(1)$ in G since there are $r^2 + 2r - 1$ edges of weight $1/\binom{2r}{2}$ crossing C. On the other hand, the induced subgraphs $G[V_1]$ and $G[V_2]$ both has minimum cut size $\Omega(2^r)$. So for any*

edge in G crossing the cut C, its strength is $\Theta(1)$, and other edges in G have strength $\Omega(2^r)$. Let F_0 be set of edges in G corresponding to e_0. About $1/r$ fraction of the edges in F_0 have strength $\Theta(1)$ while the others have strength $\Omega(2^r)$. Both $\binom{r}{2}/(\sum_{f\in F_0} k_f)$ (inverse of average) and $(\sum_{f\in F_0} \frac{1}{k_f})/\binom{r}{2}$ (average of inverse) are $O(1/r)$. However, the cut C has size 2 in the hypergraph, which means in order to build a $(1\pm\varepsilon)$-approximate cut sparsifier with $\varepsilon < 1/2$, the edge e_0 must be included.

To solve this problem, we give an algorithm that assigns the weights of edges in G such that for each hyperedge e, the strength of all corresponding edges in G whose weight is positive is close to the smallest strength edge in the clique (we will formally define this idea in the next sub-section). In this case, the maximum inverse strength is quite close to the average inverse strength, so if p_e is decided by the smallest strength (i.e. the largest inverse strength) in the clique, both the size of the sparsifier and the variance of the cuts have the properties we desire.

5.4.1. Construction of the Cut Sparsifier

In this section, we formalize the ideas introduced in the previous section. To simplify the analysis, we first consider *unweighted* hypergraphs, and then give a simple reduction from the weighted case to the unweighted case. Later, in Section 5.7, we present a more sophisticated approach for handling weighted hypergraphs that gives us our final algorithm whose run-time has only a linear dependence on m.

Let $H = (V, E)$ be an unweighted multi-hypergraph with $|V| = n$ and $|E| = m$. Our goal is to create a $(1\pm\varepsilon)$-approximate cut sparsifier, given any $\varepsilon \in (0, 1]$. That is, we want to create a weighted hypergraph $\hat{H} = (V, \hat{E}, \hat{w})$ where $\hat{E} \subseteq E$ such that with high probability, for all cuts $C = (S, \bar{S})$ of V,

$$\left|\hat{w}(\delta_{\hat{H}}(S)) - |\delta_H(S)|\right| \leq \varepsilon\,|\delta_H(S)|\,.$$

In other words, the graph $\hat{H}$ preserves all cuts up to a factor of $(1\pm\varepsilon)$. We will sample the

graph $\hat{H}$ by computing a probability p_e for each edge $e \in E$. Each edge $e \in E$ is included in $\hat{H}$ with probability p_e, and if included, it is given a weight of $\hat{w}(e) := 1/p_e$.

Given a hyperedge $e \in E$, define $F_e := \{\{u, v\} : u, v \in e, u \neq v\}$ as the clique on the vertex set of e. Let $F := \bigcup_{e \in E} F_e$ be the *multiset union* of all such cliques. Given a weight function $w^F : F \to \mathbb{R}_{\geq 0}$, we define $G = (V, F, w^F)$ as the weighted multigraph induced by w^F. Given any subset $F_{sub} \subseteq F$, define $F^+_{sub} = \{f \in F_{sub} : w^F(f) > 0\}$ to be subset of F_{sub} containing only positive weight edges.

For all hyperedges $e \in E$, define $\kappa_e := \min_{f \in F_e} k_f$ to be the minimum strength over *all* edges in its associated clique, and $\kappa^{\max}_e := \max_{f \in F^+_e} k_f$ to be the maximum strength over *all positive-weighted* edges in its associated clique.

Definition 5.4.1. *Let $\gamma \geq 1$ be some parameter. The weight function $w^F : F \to \mathbb{R}_{\geq 0}$ is called a* γ-balanced weight assignment *if it satisfies the following two conditions for all $e \in E$ in the hypergraph H:*

(1) $\sum_{f \in F_e} w^F(f) = 1$

(2) $\kappa^{\max}_e / \kappa_e \leq \gamma$.

The next theorem, whose proof appears in Section 5.5, shows that there exists a γ-balanced weight assignment for any $\gamma \geq 2$. We say two hyperedges are *distinct* if the vertex sets of these two hyperedges are not the same.

Theorem 5.8. *Suppose we are given a hypergraph with n vertices and m hyperedges such that there are at most $\bar{m}$ distinct hyperedges. Then for any integer $\gamma \geq 2$, there is an algorithm that runs in $\tilde{O}(m\bar{m}n^4)$ time and finds a γ-balanced weight assignment.*

In fact, with a more careful analysis, we can prove the statement of Theorem 5.8 is true for any real number $\gamma > 1$. Together with Bolzano-Weierstrass theorem and some standard analysis, we can prove the existance of a balanced weight function even for $\gamma = 1$. See Section 5.5.2 for more details.

Given such a weight assignment, the theorem below, whose proof appears in Section 5.6, shows that sampling with probabilities proportional to $1/\kappa_e$ gives a good sparsifier:

Theorem 5.9. *Let $\varepsilon \in (0,1]$ and let d be any integer constant. Suppose w^F is a γ-balanced weight assignment of H. Consider a random subgraph $\hat{H}$ of H where each edge $e \in E$ is sampled with probability $p_e := \min(1, \frac{8(d+6)\gamma^2 \log n}{0.38\varepsilon^2 \kappa_e})$ and is given weight $1/p_e$ if sampled. Let $\hat{w}$ be this weight function on the sampled edges. Then with probability at least $1 - O(n^{-d})$, for every cut $C = (S, \bar{S})$,*

$$\left|\hat{w}(\delta_{\hat{H}}(S)) - |\delta_H(S)|\right| \leq 2\varepsilon \left|\delta_H(S)\right|.$$

Furthermore, the expected number of edges in $\hat{H}$ is $O(\frac{\gamma^3 n \log n}{\varepsilon^2})$.

Setting $\gamma = 2$, for any unweighted hypergraph $H = (V, E)$, by Theorem 5.8, there exists an algorithm that finds a γ-balanced weight assignment. Thus by Theorem 5.9, we can create a $(1 \pm \varepsilon)$-approximate cut sparsifier of H of size $O(\frac{n \log n}{\varepsilon^2})$ with high probability.

The corollary below gives a simple reduction from the weighted case to the unweighted case.

Corollary 5.4.2. *Given a weighted hypergraph $H = (V, E, w)$, suppose W is the ratio of the largest edge weight to the smallest edge weight in H. Then for any $\varepsilon \in (0,1]$, there exists an algorithm that constructs an $(1 \pm \varepsilon)$-approximate sparsifier of H with size $O(\frac{n \log n}{\varepsilon^2})$ in $\tilde{O}(Wm^2n^4)$ time with high probability.*

Proof. Without loss of generality, assume that $1/\epsilon$ is an integer, and also that the weights w are between $3/\epsilon$ and $3W/\epsilon$. For every edge $e \in E$, we add $\lfloor w(e) \rfloor$ copies of e to a multiset E'. Since $w(e) \geq 3/\epsilon$, the number of copies of e in E' is $(w(e) \pm 1)$, which is within the range $(1 \pm \epsilon/3) \cdot w(e)$. Let $\hat{H}$ be a $(1 \pm \epsilon/3)$-approximate cut sparsifier of $H' = (V, E')$ computed using Theorem 5.8 and Theorem 5.9. Then the weight of a cut in $\hat{H}$ is within a $(1 \pm \epsilon/3)^2$ factor (which is within the range $(1 \pm \epsilon)$) of its weight in H. In H', there are at most Wm hyperedges and there are at most m hyperedges are distinct with each other. By

Theorem 5.8, the running time is $\tilde{O}(Wm^2n^4)$. □

We prove Theorem 5.8 in Section 5.5 and Theorem 5.9 in Section 5.6. In Section 5.7, we speed up our algorithm so that the running time is linear in m and eliminate the dependance of W, and thus prove Theorem 5.1.

5.5. Finding a γ-balanced Assignment

In this section, we prove Theorem 5.8, which shows that given an unweighted hypergraph $H = (V, E)$ with n vertices and m hyperedges, and for any integer $\gamma \geq 2$, we can find a γ-balanced assignment in polynomial time. Although we only consider the case when γ is an integer for convenience, the argument can be easily generalized to the case when γ is not an integer.

We find a γ-balanced assignment using an iterative algorithm. We start with the uniform weight assignment. In each step, say e is an unbalanced hyperedge (i.e. e violates condition (2) of Definition 5.4.1) where f_1 and f_2 are the two edges in F_e that "witness" e being unbalanced, i.e. f_1 has positive weight and $k_{f_1} > \gamma k_{f_2}$. We move weight from f_1 to f_2. Informally (we will prove this later), the strength of f_1 can only decrease and the strength of f_2 can only increase as a result of this weight transition. There are two possible events that may happen if we keep moving weight from f_1 to f_2: either the strength of f_1 finally moves within a γ factor of f_2; or we end up moving all the weight of f_1 to f_2, but k_{f_1} is still larger than γk_{f_2}. In either case, f_1 and f_2 are no longer a pair of "witnesses" to e being unbalanced. We repeat this weight transfer until no unbalanced hyperedge remains.

Before we formally describe the algorithm, we first prove a lemma that shows how edge strengths in a graph change when we change the weight of an edge.

Lemma 5.5.1. *Let $G = (V, E, w)$ be a weighted graph, and let $G' = (V, E, w')$ be the weighted graph obtained from G by increasing the weight of some edge f by δ. For any edge f', denote by $k_{f'}$ and $k'_{f'}$ the strengths of f' in G and G' respectively. Then for any edge f',*

1. $k_{f'} \leq k'_{f'} \leq k_{f'} + \delta$

2. *If* $k'_{f'} > k_{f'}$, *then* $k_{f'} \geq k_f$ *and* $k'_{f'} \leq k'_f$

Proof. Let f' be an edge, and let $G[X_{f'}]$ be the induced subgraph of G that contains f' and has minimum cut size $k_{f'}$. Since we only increase the weight of an edge f, the minimum cut size of $G'[X_{f'}]$ is at least $k_{f'}$, which means $k_{f'} \leq k'_{f'}$. On the other hand, since the weight of f is increased by δ, the minimum cut size of any induced subgraph is increased by at most δ. So $k'_{f'} \leq k_{f'} + \delta$.

Next, we prove the second part of the lemma. Let f' be an edge, and suppose $k'_{f'} > k_{f'}$. Let $G'[X'_{f'}]$ be the induced subgraph of G' that contains f' and has minimum cut size $k'_{f'}$. Since $k'_{f'} > k_{f'}$, the minimum cut size of $G[X'_{f'}]$ is strictly less than $k'_{f'}$, which means f is a part of some minimum cut of $G[X'_{f'}]$. In particular, this implies that f is in $X'_{f'}$, so k'_f is at least the minimum cut size of $G'[X'_{f'}]$, which is $k'_{f'}$.

On the other hand, let $G[X_f]$ be the induced subgraph of G that contains f and has minimum cut size k_f. Consider the subgraph $G[X'_{f'} \cup X_f]$. Let $C = (S, \bar{S})$ be a minimum cut of this induced subgraph, and let c be the size of C. Since this subgraph contains f', by definition of strength, c is at most $k_{f'}$. Note that $X'_{f'}$ and X_f have nonempty intersection (they both contain the edge f). Therefore any cut of $X'_{f'} \cup X_f$ must either cut through X_f, or cut through $X'_{f'}$ but not X_f. In the case that C cuts through $X'_{f'}$ but not X_f, C does not cut through f, so it has size at most c in $G'[X'_{f'}]$ (since the weight of all edges crossing C stays the same). This implies that the minimum cut of $G'[X'_{f'}]$ is at most c, which means that $k'_{f'} \leq c \leq k_{f'}$, contradicting our assumption. So it must be the case that C cuts through the vertex set X_f, which means c is at least the minimum cut size of $G[X_f]$, and therefore $k_f \leq c \leq k_{f'}$. □

Our algorithm will maintain the invariant that all weights in the current weight assignment graph are integer multiples of some fixed $\delta > 0$, and the magnitude of each weight update will

be exactly δ. In such a graph, Lemma 5.5.1 immediately implies that changing (increasing or decreasing) the weight of some edge f by δ can only change the strength of an edge f' if f and f' have the same strength both before and after the change.

5.5.1. The Algorithm

Now we describe the algorithm to find a γ-balanced assignment. Let $\delta = \frac{1}{n^2}$. First we assign the initial weights $w^{init} : F \to \mathbb{R}_{\geq 0}$ with the following constraint: the weight of each edge in G is an integer multiple of δ and is at least 2δ. We can always do so because each hyperedge in H has weight 1, which is an integer multiple of δ, and the number of edges in the clique associated with a hyperedge is at most $\binom{n}{2}$, which is less than $\frac{1}{2\delta}$. These initial weights give us a set of initial edge strengths k_f^{init} of the weighted graph $G^{init} = (V, F, w^{init})$. Define $K_0 := \min_{f \in F} k_f^{init}$, and define ℓ to be the smallest integer such that $K_0 \cdot \gamma^\ell$ is larger than $\max_{f \in F} k_f^{init}$. For each integer $0 \leq i \leq \ell$, define $K_i = K_0 \cdot \gamma^i$. Note that since the weights of all edges are integer multiples of δ, the strength of each edge is also an integer multiple of δ, which means K_0 is an integer multiple of δ. Since γ is an integer, all K_i is also integer multiples of δ. We partition the interval $I = [K_0, K_\ell]$ into subintervals $I_0, I_1, I_2, \ldots, I_\ell$, where $I_j := (K_{i-1}, K_i]$ for $i > 0$, and $I_0 = \{K_0\}$. Note that $\max_{f \in F} k_f^{init}$ is at most the total weight of the edges and K_0 is at least 2δ, so ℓ is at most $\log_\gamma(n^2 m) = O(\log m)$. We fix this partition for the rest of this section.

We use this partition $I_0, I_1, I_2, \ldots, I_\ell$ to determine how to iteratively modify these weights. Given a real number $x \in I$, we define $\text{ind}(x)$ to be the integer j such that $x \in I_j$. Given a weight function $w^F : F \to \mathbb{R}_{\geq 0}$ and the corresponding edge strengths $k : F \to \mathbb{R}_{\geq 0}$, we say that a hyperedge $e \in E$ is *bad* in $G = (V, F, w^F)$ if there exist some $f, f' \in F_e$ such that $w^F(f') > 0$ and $k_f < K_{\text{ind}(k_{f'})-1}$. It is clear that if a hyperedge is not bad, then it is γ-balanced. We note that in general, as we update the weights, k_f and $k_{f'}$ might not be contained in I (so $\text{ind}(k'_f)$ might not be defined), but as it will turn out that we will maintain the invariant that all the edge strengths are always contained in I. We expand this definition to $\text{ind}(e) := \text{ind}(\max_{f \in F_e^+} k_f)$. Note that a hyperedge e is bad if and only if

$\kappa_e < K_{\mathrm{ind}(e)-1}$.

We run the following algorithm: while there exist bad hyperedges, we find a bad hyperedge e with the maximum $\mathrm{ind}(e)$. Let $f, f' \in F_e$ be a pair that such $w^F(f') > 0$ and $k_f < K_{\mathrm{ind}(k_{f'})-1}$. We move δ weight from f' to f.

EdgeBalancing: An algorithm that eliminates all bad hyperedges

1. $w = w^{init}$
2. While there exists some bad hyperedge
3. Let e be the one with maximum $\mathrm{ind}(e)$
4. Let $f_{\min} := \arg\min_{f \in F_e} k_f$ and $f_{\max} := \arg\max_{f \in F_e^+} k_f$
5. Let $k_{\min}$ and $k_{\max}$ to be the strengths of $f_{\min}$ and $f_{\max}$, respectively
6. Increase $w(f_{\min})$ by δ and decrease $w(f_{\max})$ by δ
7. Return w

Note that throughout the execution of the algorithm, the weight of each edge is an integer multiple of δ, so the strength of each edge throughout the running of the algorithm is also an integer multiple of δ. To prove the correctness of the algorithm, we first prove an important invariant that is maintained by the algorithm.

Claim 5.5.2. *Let i equal the value of* $\mathrm{ind}(e)$ *at some iteration of the while loop. For any edge f whose strength increased as a result of transferring the weights (Line 6),* $\mathrm{ind}(k_f) < i$ *after executing the transfer of weights. Also, no edge f has strength less than K_0 after executing the transfer of weights.*

Proof. Fix some iteration of the while loop, and let $i = \mathrm{ind}(e)$. By definition of $\mathrm{ind}(e)$ and $f_{\max}$, we have $\mathrm{ind}(k_{\max}) = \mathrm{ind}(e)$. On the other hand, since e is a bad hyperedge, we have $k_{\min} < K_{i-1}$, which means $k_{\min} \leq K_{i-1} - \delta$ since $k_{\min}$ is an integer multiple of δ. By the first half of Lemma 5.5.1, $k_{f_{\min}}$ is increased by at most δ, which implies that after the weight transfer, $k_{f_{\min}} \leq K_{i-1}$. By the second half of Lemma 5.5.1, for any edge f such that k_f

increases, $k_f \leq k_{f_{\min}} \leq K_{i-1}$, so $\text{ind}(k_f) \leq i-1$ after the weight transfer. This concludes the first part of the claim.

Now we prove the second part of the claim inductively. Suppose that before we change the weights, no edge has strength less than K_0. Since $K_0 \leq k_{\min} < K_{i-1}$, $i \geq 2$, so $k_{\max} \geq K_1 + \delta$. By the first half of Lemma 5.5.1, $k_{f_{\max}} \geq K_1$ after the weight transfer. By the second half of Lemma 5.5.1, for any edge f such that k_f decreases, $k_f \geq k_{f_{\max}} \geq K_1 > K_0$ after the weight change. So the second invariant still holds and this concludes the second part of the claim. □

Claim 5.5.2 essentially proves that the interval $I = [K_0, K_\ell]$ (which was defined using the initial graph G^{init}) is the correct range of strengths to focus on. Algorithm 1 gives a γ-balanced assignment if it terminates since there would be no bad hyperedges. Therefore, to prove Theorem 5.8, it is sufficient to prove that the running time of Algorithm 1 is $\tilde{O}(m\bar{m}n^4)$. We call the t^{th} iteration of the while loop as iteration t. The following claim is another important invariant of Algorithm 1.

Claim 5.5.3. *For any integer i, we define iteration t_i as the earliest iteration that the bad hyperedge e in the while loop has* $\text{ind}(e) \leq i$. *Then after iteration t_i, the total weight of edges that have strength larger than K_{i-1} is non-increasing.*

Proof. For any $t \geq 1$, we denote e^t as the bad hyperedge in line 3 during iteration t. We say a hyperedge e' is *very bad* if $\kappa_{e'} < K_{\text{ind}(e')-1} - \delta$. We first prove that at any iteration starting from t_i, no hyperedge e' with $\text{ind}(e') > i$ is very bad. We prove it by contradiction. Suppose the statement is not true, and let $\bar{t} \geq t_i - 1$ be the first iteration such that after iteration $\bar{t}$, a hyperedge $\bar{e}$ is very bad. At the beginning of iteration t_i, by the definition of e^{t_i}, no hyperedge e' with $\text{ind}(e') > i$ is bad, and hence no such hyperedge is very bad. So $\bar{t} \geq t_i$, which means at the beginning of iteration $\bar{t}$, no hyperedge e' with $\text{ind}(e') > i$ is very bad. There are two possible reasons that would cause $\bar{e}$ to become very bad: either $\text{ind}(\bar{e})$ is increased or $\kappa_{\bar{e}}$ is decreased during the weight transfer in iteration $\bar{t}$.

Suppose $\text{ind}(\bar{e})$ increases during the weight transfer in iteration $\bar{t}$, and let $f \in F_{\bar{e}}^{+}$ be the edge that $\text{ind}(f)$ increases. By Lemma 5.5.1, k_f increases by at most δ during iteration $\bar{t}$. On the other hand, since k_f is always an integer multiple of δ, $k_f = K_{\text{ind}(f)}$ at the beginning of iteration $\bar{t}$. Let $\hat{f} \in F_{e^{\bar{t}}}$ be the edge whose weight is increased during iteration $\bar{t}$. By Lemma 5.5.1, $k_{\hat{f}} = k_f = K_{\text{ind}(k_f)}$ since $k_{\hat{f}}$ is an integer multiple of δ. So at the beginning of iteration $\bar{t}$, $\text{ind}(e^{\bar{t}}) \geq \text{ind}(f) + 2$ since $e^{\bar{t}}$ is bad. This means

$$k_{\hat{f}} = K_{\text{ind}(f)} < K_{\text{ind}(f)+1} - \delta \leq K_{\text{ind}(e^{\bar{t}})-1} - \delta$$

where the first inequality is because $K_{\text{ind}(f)+1} - K_{\text{ind}(f)} \geq K_1 - K_0 = (\gamma - 1)K_0 \geq 2\delta$. So $e^{\bar{t}}$ is very bad at the beginning of iteration $\bar{t}$, which contradicts the minimality of $\bar{t}$.

Now consider the other possibility - $\kappa_{\bar{e}}$ decreases while $\text{ind}(\bar{e})$ does not increase during weight transfer in iteration $\bar{t}$. By Lemma 5.5.1, $\kappa_{\bar{e}}$ is decreased by at most δ during iteration $\bar{t}$, which means that at the beginning of iteration $\bar{t}$, $\kappa_{\bar{e}} \leq K_{\text{ind}(\bar{e})-1} - \delta$. So $\bar{e}$ is a bad hyperedge. On the other hand, by Lemma 5.5.1, $\kappa_{e^{\bar{t}}}^{\max} = \kappa_{\bar{e}}$, so $\text{ind}(e^{\bar{t}}) < \text{ind}(\bar{e})$, which contradicts that $e^{\bar{t}}$ is the bad hyperedge which has the maximum index at the beginning of iteration $\bar{t}$.

So at any time after iteration t_i, there is no very bad hyperedge e' with $\text{ind}(e') > i$.

Since the algorithm only moves the weight from a high strength edge to a low strength edge, there is only one way that the total weight of the edges that has strength larger than K_{i-1} increases: the strength of some edges increase from less than or equal to K_{i-1} to larger than K_{i-1}. At the beginning of any iteration t' after t_i, by Claim 5.5.2, if $\text{ind}(e^{t'}) \leq i$, any edge f whose strength increases has $k_f \leq K_{i-1}$. On the other hand, if $\text{ind}(e^{t'}) > i$, $e^{t'}$ is not very bad, which means $\kappa_{e^{t'}} \geq K_i - \delta > K_{i-1}$. So any edge f whose strength increases already has $k_f > K_{i-1}$ at the beginning of iteration t'. So the total weight of edges that has strength larger than K_{i-1} is non-increasing. □

Claim 5.5.4. *Algorithm 1 iterates in the while loop $\tilde{O}(mn^2)$ times.*

Proof. Throughout the running of the algorithm, for any $1 \leq j \leq \ell - 1$, we define a nonnegative potential function W_j as follows: before iteration t_j, W_j is always equal to m; after iteration t_j, W_j equals the total weight of edges that have strength larger than K_j. Since the total weight of all edges is m, by Claim 5.5.3, all W_j's are non-increasing throughout the running of the algorithm. On the other hand, for each iteration, suppose the bad hyperedge e has $\text{ind}(e) = i$. Note that this iteration cannot be before t_i. In this iteration, we transfer δ amount of weight from an edge whose strength is larger than K_{i-1} to an edge whose strength is less than K_{i-1}. Furthermore, the edge whose weight increases does not have strength larger than K_{i-1} after the weight change. So W_{i-1} is decreased by at least δ. Thus, in each iteration, no W_j increases, and W_i is decreased by at least δ, which means there are at most $m * \ell/\delta = \tilde{O}(mn^2)$ iterations since $\ell = O(\log m)$. □

By Claim 5.5.2 and Claim 5.5.4, Algorithm 1 correctly outputs a γ-balanced weight assignment within a polynomial number of iterations.

Proof of Theorem 5.8. The multi-graph G contains $O(mn^2)$ edges, so computing the initial weight assignment takes $O(mn^2)$ time.

In each iteration of the while loop, we need to compute the strength of all edges in G and find the bad hyperedges with maximum index. Note that if two edges share the same endpoints, their strengths are the same, so to compute the strength of the edges, we only need to compute the strength on a weighted complete graph $\bar{G}$ where for each pair of vertices (u, v), the weight of edge (u, v) is the sum of weights of edges whose endpoints are u and v in G. By Lemma 5.3.6, we need $\tilde{O}(n^3)$ time to compute the strength of all edges in $\bar{G}$ since there are $\binom{n}{2}$ edges in $\bar{G}$. Updating the weight of edges in $\bar{G}$ only takes $O(1)$ time.

Once the strengths of all edges in $\bar{G}$ has been computed, it takes $O(mn^2)$ time to check for each hyperedge if it is bad or not. However, if there are at most $\bar{m}$ distinct hyperedges, we can do it in $O(\bar{m}n^2)$ time in the following way: we group the hyperedges with the same vertex sets. For each group, we store the total weight in each edge slot, together with the identity

of the hyperedges which have positive weight in each edge slot. To find a bad hyperedge with the maximum index in one group, we only need to consider the edge slot that has the maximum strength with positive weight, and check if the hyperedge that has weight in this slot is bad. In each iteration, it takes $O(\bar{m}n^2)$ time to find the maximum strength positive weight edge slot in each group and takes constant time to update the information in each edge slot.

Thus overall, each iteration takes $\tilde{O}(\bar{m}n^2+n^3) = \tilde{O}(\bar{m}n^2)$ time. So by Claim 5.5.4, Algorithm 1 runs in $\tilde{O}(m\bar{m}n^4)$ time. □

5.5.2. Proof of Existence of 1-balanced Assignment

In this section, we prove that there exists a 1-balanced weight assignment $G = (V, F, w)$ for every hypergraph $H = (V, E)$. To do this, we first prove that the conclusion of Theorem 5.8 holds for all $\gamma > 1$ (as opposed to $\gamma \geq 2$). Equivalently, we prove that Theorem 5.8 holds for $\gamma = 1+1/i$ every positive integer i. The only change needed in the algorithm is to use $\delta = \frac{1}{n^2 i}$ instead of $\delta = \frac{1}{n^2}$, and to ensure that K_0 is at least $2i\delta$ instead of 2δ. The rest of the proofs are completely analogous, with the only modification being that $(\gamma - 1)K_0 \geq 2\delta$ no longer follows from the fact that $\gamma \geq 2$, but simply from the fact that K_0 is at least $2i\delta = \frac{2\delta}{\gamma-1}$. Note that the number of iterations (and hence the running time) of the algorithm is increased by a factor of i^2, since δ and ℓ are decreased and increased by a factor of i respectively.

For the rest of this section, it will be convenient to represent a weight assignment $w : F \to \mathbb{R}_{\geq 0}$ as a vector in $\mathbb{R}_{\geq 0}^{|F|}$. Additionally, given a vector $w \in \mathbb{R}_{\geq 0}^{|F|}$, we use $k_f(w)$, $\kappa_e(w)$, and $\kappa_e^{\max}(w)$, and $F_e^+(w)$ to denote the value of these quantities in the weight assignment represented by w.

Let $\{w_i \in \mathbb{R}_{\geq 0}^{|F|}\}$ be a sequence of vectors such that w_i represents a $1 + 1/i$-balanced weight assignment. We invoke the Bolzano-Weierstrass Theorem on this sequence:

Theorem 5.10 (Bolzano-Weierstrass Theorem)**.** *Every bounded sequence of vectors in* $\mathbb{R}^n$ *has a convergent subsequence.*

Denote this convergent subsequence by $\{w'_i \in \mathbb{R}^{|F|}\}$, and let w be the limit of this subsequence. We will use a limiting argument to show that w is 1-balanced. First we note that the strength of an edge k_f is a (1-Lipschitz) continuous function of the weight assignment. This follows immediately from the first half of Lemma 5.5.1. Therefore $\lim k_f(w'_i) = k_f(\lim w'_i) = k_f(w)$. Since min is also a continuous function, this implies that

$$\lim \kappa_e(w'_i) = \lim \min_{f \in F_e} k_f(w'_i) = \min_{f \in F_e} \lim k_f(w'_i) = \min_{f \in F_e} k_f(w) = \kappa_e(w) \tag{5.1}$$

We would like to be able to make a similar statement about $\lim \kappa_e^{\max}(w'_i)$, but it is not true in general because $\kappa_e^{\max}$ is not a continuous function of the weight assignment vector. Instead, we observe that for i large enough, the set $F_e^+(w'_i)$ is a superset of $F_e^+(w)$, since the weight of any edge in $F_e^+(w)$ must eventually become positive in the sequence $\{w'_i\}$. So

$$\begin{aligned} \lim \kappa_e^{\max}(w'_i) &= \lim \max_{f \in F_e^+(w'_i)} k_f(w'_i) \geq \lim \max_{f \in F_e^+(w)} k_f(w'_i) \\ &= \max_{f \in F_e^+(w)} \lim k_f(w'_i) = \max_{f \in F_e^+(w)} k_f(w) = \kappa_e^{\max}(w) \end{aligned} \tag{5.2}$$

Here the inequality used the fact that for for large i, $F_e^+(w'_i) \supseteq F_e^+(w)$, and second equality used that max is a continuous function. Combining Eq (5.1) and Eq (5.2),

$$\kappa_e^{\max}(w) \leq \lim \kappa_e^{\max}(w'_i) \leq \lim((1 + 1/i) \cdot \kappa_e(w'_i)) = 1 \cdot \kappa_e(w),$$

where the second inequality holds because w'_i is $1+1/i$-balanced. Therefore, w is 1-balanced, as desired.

5.6. Constructing a Cut Sparsifier from a γ-balanced Assigment

In this section, we prove Theorem 5.9, which shows that given a γ-balanced assigment w^F, we can construct a $(1 \pm \varepsilon)$-approximate cut sparsifier that contains $O(\frac{\gamma^3 n \log n}{\varepsilon^2})$ edges.

Let $\rho = \frac{8(d+6)\gamma^2 \log n}{0.38\varepsilon^2}$, we sample each hyperedge e in H with probability $p_e = \min\{1, \frac{\rho}{\kappa_e}\}$. If

an edge e is sampled, it will have weight $\hat{w}_e = \frac{1}{p_e}$ in $\hat{H}$. We first show the expected number of edges in the sparsifier $\hat{H}$ is small.

Claim 5.6.1. *The expected number of edges in the sparsifier $\hat{H}$ is $O(\frac{\gamma^3 n \log n}{\varepsilon^2})$.*

Proof. The expected number of edges in the sparsifier is

$$\sum_{e \in E} p_e \leq \rho \sum_{e \in E} \frac{1}{\kappa_e} = \rho \sum_{e \in E} \sum_{f \in F_e} \frac{w^F(f)}{\kappa_e} = \rho \sum_{e \in E} \sum_{f \in F_e} \frac{w^F(f)}{k_f} \frac{k_f}{\kappa_e}$$
$$\leq \rho\gamma \sum_{e \in E} \sum_{f \in F_e} \frac{w^F(f)}{k_f} = \rho\gamma \sum_{f \in F} \frac{w^F(f)}{k_f} \leq \rho\gamma(n-1)$$

For the second-to-last inequality, we used that for every $f \in F_e$ such that $w^F(f) > 0$, $k_f \leq \kappa_e^{\max} \leq \gamma \kappa_e$ due to Definition 5.4.1. The last inequality is due to Claim 5.3.5, which asserts that $\sum_{f \in F} \frac{w^F(f)}{k_f} \leq n - 1$. By definition of ρ, this is $O(\gamma^3 n \log n / \varepsilon^2)$. □

In the rest of this section, we prove that $\hat{H}$ is indeed a good sparsifier. This proof is inspired by the framework of [50], who partition the edges into classes based on strength, and analyze the performance of each class separately. Before we start, as an additional piece of notation, given any subset of hyperedges $E' \subseteq E$, we define $\hat{E}'$ to be the subset of edges of E' that were sampled in the sparsifier.

We first group the edges by their strengths. For each integer i, let $F_{\geq i} := \{f \in F^+ : k_f \geq \rho \cdot 2^i\}$ be the multiset of positive-weight edges with strength at least $\rho \cdot 2^i$. Let $E_{\geq i} := \{e \in E : \kappa_e \geq \rho \cdot 2^i\}$ be the set of hyperedges with minimum strength at least $\rho \cdot 2^i$, and let $E_{\geq i}^{\max} := \{e \in E : \kappa_e^{\max} \geq \rho \cdot 2^i\}$ be the set of hyperedges with maximum strength at least $\rho \cdot 2^i$. Note that $E_{\geq i} \subseteq E_{\geq i}^{\max}$.

Let $E_i := E_{\geq i} \setminus E_{\geq i+1}$. We will prove an error bound for each E_i separately. To prove this error bound, we define and analyze some slightly modified graphs. We first define some modified weights $w_i^F : F_{\geq i} \to \mathbb{R}^+$ and $w_i^E : E_{\geq i} \to \mathbb{R}^+$ in the following way: for an

edge $f \in F$ such that $\rho \cdot 2^j \leq k_f \leq \rho \cdot 2^{j+1}$, $w_i^F(f) := w^F(f) \cdot 2^{i-j}$, and for a hyperedge $e \in E_j$, $w_i^E(e) := 2^{i-j}$. Note that for a hyperedge $e \in E_i$, the weight of e in w_i^E remains 1. Finally, define $G_{\geq i} = (V, F_{\geq i}, w_i^F)$, $H_{\geq i} = (V, E_{\geq i}, w_i^E)$, and $H_{\geq i}^{\max} = (V, E_{\geq i}^{\max}, w_i^E)$ to be the weighted graphs induced by these modified weights.

The following lemma proves that for any i and any cut C, the weight of the edges in $\hat{E}_i$ which cross C is close to its expectation.

Lemma 5.6.2. *Fix some integer $i \geq 0$. With probability at least $1 - 4n^{-(d+1)}$, for all cuts $C = (S, \bar{S})$ of V, we have that*

$$\left|\hat{w}(\delta_{\hat{E}_i}(S)) - |\delta_{E_i}(S)|\right| \leq \frac{\varepsilon}{\gamma} \cdot w_i^E(\delta_{E_{\geq i}^{\max}}(S))$$

Note that this lemma is not claiming that $\hat{E}_i$ is a good sparsifier of E_i - the error term $\frac{\varepsilon}{\gamma} w_i^E(\delta_{E_{\geq i}^{\max}}(S))$ can be much larger than $\varepsilon\,|\delta_{E_i}(S)|$. We postpone the proof of Lemma 5.6.2 and first show why Lemma 5.6.2 completes the proof Theorem 5.9.

Proof of Theorem 5.9. In order to obtain concentration over all edges, we wish to take a union bound over every value of i such that E_i is not empty. By Claim 5.3.4, there are at most $n - 1$ such values of i.

By Lemma 5.6.2, taking a union bound over these values of i, we get that with probability at least $1 - 4n^{-d}$, for all cuts $C = (S, \bar{S})$ of V and for all i,

$$\left|\hat{w}(\delta_{\hat{E}_i}(S)) - |\delta_{E_i}(S)|\right| \leq \frac{\varepsilon}{\gamma} \cdot w_i^E(\delta_{E_{\geq i}^{\max}}(S)) \leq \frac{\varepsilon}{\gamma} \cdot \sum_{j \geq i - \log \gamma} 2^{i-j} \left|\delta_{E_j}(S)\right|$$

where the last inequality is because $E_{\geq i}^{\max} \subseteq E_{\geq i - \log \gamma}$ (since $\kappa_e^{\max}/\kappa_e \leq \gamma$). Note that for all hyperedges e that do not belong to any E_i, $\kappa_e \leq \rho$, so $p_e = 1$. That is, the contribution of these hyperedges to the error is 0. We sum the errors over edges in E_i for $i \geq 0$ to obtain

that the total error is at most

$$\begin{aligned}\sum_{i\geq 0}\left|\hat{w}(\delta_{\hat{E}_i}(S)) - |\delta_{E_i}(S)|\right| &\leq \frac{\varepsilon}{\gamma}\sum_{i\geq 0}\sum_{j\geq i-\log\gamma} 2^{i-j}\left|\delta_{E_j}(S)\right| \\ &= \frac{\varepsilon}{\gamma}\sum_{j\geq 0}\left(\left|\delta_{E_j}(S)\right|\cdot\sum_{i\leq j+\log\gamma} 2^{i-j}\right) \\ &\leq 2\varepsilon\sum_{j\geq 0}\left|\delta_{E_j}(S)\right|\end{aligned}$$

Which is at most $2\varepsilon\left|\delta_E(S)\right|$. Here the last inequality is due to $\sum_{i\leq j+\log\gamma} 2^{i-j} \leq \sum_{i=-\lfloor\log\gamma\rfloor}^{\infty} 2^{-i} \leq 2\gamma$. Therefore with probability at least $1-4n^{-d}$, for all cuts $C=(S,\bar{S})$, the size of C in $\hat{H}$ is a $(1\pm 2\varepsilon)$-approximation of the size in H. □

5.6.1. Proof of Lemma 5.6.2

Before proving Lemma 5.6.2, we first make some observations. As stated before, we associate the performance of $\hat{H}$ with the auxiliary standard graph G. The following claim states that for any cut C, the total weight of the edges crossing C in $H^{\max}_{\geq i}$ is at least the total weights of the edges crossing C in $G_{\geq i}$.

Claim 5.6.3. *For any cut $C=(S,\bar{S})$ of V, $w_i^E(\delta_{E^{\max}_{\geq i}}(S)) \geq w_i^F(\delta_{F_{\geq i}}(S))$.*

Proof. Let e be some hyperedge, and let $f\in F_e$. If f is a member of $G_{\geq i}$, then e must be a member of $H^{\max}_{\geq i}$. Therefore if f is cut by C in $G_{\geq i}$, then e must be cut by C in $H^{\max}_{\geq i}$. Thus,

$$\sum_{f\in\delta_{G_{\geq i}}(S)} w_i^F(f) \leq \sum_{e\in\delta_{H^{\max}_{\geq i}}(S)}\sum_{f\in F_e} w_i^F(f) = \sum_{e\in\delta_{H^{\max}_{\geq i}}(S)} w_i^E(e)$$

Here the equality is because for an edge $e\in E_j$, by condition (1) of γ-balanced weight assignments, $\sum_{f\in F_e} w_i^F(f) = \sum_{f\in F_e} w^F(f)\cdot 2^{i-j} = 2^{i-j} = w_i^E(e)$. □

In our analysis, we will independently bound the error incurred by each connected component of $G_{\geq i}$. The following claim states that no hyperedge is split among two different connected

components of $G_{\geq i}$.

Claim 5.6.4. *For any $e \in E_{\geq i}$, the entire vertex set of e belongs to the same connected component in $G_{\geq i}$.*

Proof. Consider an edge $e \in E_{\geq i}$ and let u, v be any two vertices in e. By definition of κ_e, the strength of the edge $(u, v) \in F_e$ is at least κ_e, so there exists some vertex set $X \subseteq V$ such that $u, v \in X$ and the induced subgraph $G[X]$ has min-cut size at least $\kappa_e > 0$.

Therefore u and v are connected by a path P such that each edge on P has positive weight. On the other hand, since $G[X]$ has min-cut size at least κ_e, which is at least $\rho \cdot 2^i$, all edges f in $G[X]$ have $k_f \geq 2^i$. By definition of $F_{\geq i}$, this implies that all edges on P are in $F_{\geq i}$, so u and v are connected in $G_{\geq i}$. □

The following claim is similar to Lemma 3.2 in [50], which states that the min-cut size of each component in $G_{\geq i}$ is at least $\rho \cdot 2^i$, even with regards to the new weight function w_i^F. We give the proof of this claim for completeness.

Claim 5.6.5 (Analog of Lemma 3.2 in [50])**.** *Let A_G be a connected component of $G_{\geq i}$. Then the minimum cut size of A_G is at least $\rho \cdot 2^i$.*

Proof. Let A'_G be the graph with the same vertex set and edge set as A_G, but instead of the modified weights w_i^F, we use the original weights w^F. We first claim that the strength of an edge f in A'_G is the same as its strength in G. To see this, let $X \subseteq V$ be a set of vertices such that $f \subseteq X$ and the induced weighted graph $G[X]$ has min-cut size at least k_f. Let $G[X]^+$ denote the subgraph of $G[X]$ that contains only positive-weight edges. Then every edge in $G[X]^+$ has strength at least $k_f \geq \rho \cdot 2^i$, which implies that $G[X]^+$ is a (induced) subgraph of $F_{\geq i}$. Since $G[X]^+$ is connected (and A'_G is a connected component), $G[X]^+$ is also an induced subgraph of A'_G, providing a certificate that the strength of f in A'_G is k_f.

Next, fix a cut $C = (S, \bar{S})$ of the vertex set of A_G. Let f^* be a maximum strength edge in

$\delta_{A'_G}(S)$. We claim that the total weight of strength k_{f^*} edges in $\delta_{A'_G}(S)$ is at least k_{f^*}. To see this, let $X \subseteq V(A'_G)$ be a set of vertices such that $f^* \subseteq X$ and the min-cut size of $A'_G[X]$ is k_{f^*}. As required, all edges in $A'_G[X]$ have strength at least k_{f^*}, and the total weight of such edges crossing C is at least k_{f^*}. Furthermore, by maximality of f^*, all edges crossing C in $A'_G[X]$ have strength exactly k_{f^*}. Let j be the index such that $\rho \cdot 2^j \leq k_{f^*} \leq \rho \cdot 2^{j+1}$. Now we bound the weight of edges crossing the cut in A_G:

$$w_i^F(\delta_{A_G}(S)) \geq \sum_{f \in \delta_{A_G}(S): k_f = k_{f^*}} w_i^F(f) = \sum_{f \in \delta_{A_G}(S): k_f = k_{f^*}} w^F(f) \cdot 2^{i-j} \geq k_{f^*} \cdot 2^{i-j} \geq \rho \cdot 2^i$$

□

To prove Lemma 5.6.2, for any cut $C = (S, \bar{S})$, we deal with each connected component in $G_{\geq i}$ separately. For each component A_G, we use concentration bound Lemma 2.2.3 together with Claim 5.6.5 to prove that the total weights of the edges crossing C in A_G is preserved within an additive error $O(\max\{w_i^E(\delta_{A_H}(S)), w_i^F(\delta_{A_G}(S))\})$ where A_H is the subhypergraph of $H_{\geq i}$ induced by the vertex set of A_G (it is well defined due to Claim 5.6.4). On the other hand, since $w_i^E(\delta_{E_{\geq i}^{\max}}(S))$ dominates both $w_i^E(\delta_{E_{\geq i}}(S))$ and $w_i^F(\delta_{F_{\geq i}}(S))$ (by Claim 5.6.3), by summing up the weights of the edges crossing C in different components, we are able to prove that for the edges in H_i, the total weights of the edges crossing C is perserved within additive error $O(w_i^E(\delta_{E_{\geq i}^{\max}}(S)))$.

Proof of Lemma 5.6.2. Fix some connected component A_G of $G_{\geq i}$, and let V_A be the vertex set of this component. Let $C = (S, \bar{S})$ be some cut of V_A. For brevity, let $A_H := H_{\geq i}[V_A]$ and $A'_H := H_i[V_A]$ be the subgraphs induced by this component.

In order to apply Lemma 2.2.3, we set the random variables $x_1, \ldots, x_k$ to be the sampled weights of edges in $\delta_{A'_H}(S)$ (so k equals $\left|\delta_{A'_H}(S)\right|$). We set $N := \max\{w_i^E(\delta_{A_H}(S)), w_i^F(\delta_{A_G}(S))\}$. We know that for each edge $e \in A'_H$, $w_i^E(e) = 2^{i-i} = 1$, so $N \geq w_i^E(\delta_{A_H}(S)) \geq \left|\delta_{A'_H}(S)\right|$. Therefore $N \geq k$, and we can indeed apply Lemma 2.2.3.

Let c be the size of the minimum cut of A_G. By Claim 5.6.5, we have $c \geq \rho \cdot 2^i$. Now define $\alpha := \frac{w_i^F(\delta_{A_G}(S))}{c}$. Note that N is at least $w_i^F(\delta_{A_G}(S)) = \alpha c \geq \alpha \cdot \rho \cdot 2^i$.

Also, we have $\min_{e \in \delta_{A'_H}(S)} p_e = \min\{1, \min_{e \in \delta_{A'_H}(S)} \rho/\kappa_e\} \leq \min\{1, \rho/(\rho \cdot 2^{i+1})\} \leq 1/2^{i+1}$. The second-to-last inequality is because for any edge $e \in E_i$, we have that $\kappa_e \leq \rho \cdot 2^{i+1}$, and the last inequality is because $i \geq 0$.

We apply Lemma 2.2.3 and get that

$$\begin{aligned}
\Pr\left(\left|\hat{w}(\delta_{\hat{A}'_H}(S)) - \left|\delta_{A'_H}(S)\right|\right| \geq \frac{\varepsilon}{2\gamma} N\right) &\leq 2\exp\left(-\frac{0.38\varepsilon^2}{4\gamma^2} \cdot \min p_e \cdot N\right) \\
&\leq 2\exp\left(-\frac{0.38\varepsilon^2}{4\gamma^2} \cdot \frac{1}{2^{i+1}} \cdot \alpha \cdot \frac{8(d+6)\gamma^2 \log n}{0.38\varepsilon^2} \cdot 2^i\right) \\
&= 2n^{-(d+6)\alpha}
\end{aligned} \tag{5.3}$$

We now have a concentration bound which gets stronger as α increases.

Apply cut counting bound (Lemma 5.3.7) on the weighted graph A_G, and we use this to apply a union bound over all cuts $C = (S, \bar{S})$ of A_H such that $\alpha c \leq w_F^i(\delta_{A_G}(S)) \leq 2\alpha c$ to conclude that with probability at least $1 - 2n^{2\cdot 2\alpha} \cdot n^{-(d+6)\alpha} = 1 - 2n^{-(d+2)\alpha}$, the event in equation (5.3) does not occur for all of these cuts. We again apply the union bound over all values of $\alpha \geq 1$ that are powers of 2 to obtain that with probability at least $1 - \sum_{j=0}^{\infty} 2n^{-(d+2)\cdot 2^j} \geq 1 - 4n^{-(d+2)}$, for all cuts $C = (S, \bar{S})$ of $V(A_H)$,

$$\begin{aligned}
\left|\hat{w}(\delta_{\hat{A}'_H}(S)) - \left|\delta_{A'_H}(S)\right|\right| &\leq \frac{\varepsilon}{2\gamma} \cdot \max\{w_i^E(\delta_{A_H}(S)), w_i^F(\delta_{A_G}(S))\} \\
&\leq \frac{\varepsilon}{2\gamma} \cdot \left(w_i^E(\delta_{A_H}(S)) + w_i^F(\delta_{A_G}(S))\right)
\end{aligned}$$

Now we apply another union bound over all connected components of $G_{\geq i}$ (of which there are at most n) and sum this error term over all components. Let $C = (S, \bar{S})$ be a cut of the entire vertex set V. By Claim 5.6.4, every hyperedge in $\delta_{H_{\geq i}}(S)$ is cut in exactly one

such connected component. Therefore with probability at least $1 - 4n^{-(d+1)}$, for all cuts $C = (S, \bar{S})$ of V,

$$\left|\hat{w}(\delta_{\hat{E}_i}(S)) - |\delta_{E_i}(S)|\right| \leq \frac{\varepsilon}{2\gamma} \cdot \left(w_i^E(\delta_{E_{\geq i}}(S)) + w_i^F(\delta_{F_{\geq i}}(S))\right)$$

By Claim 5.6.3 and by the fact that $H_{\geq i}$ is a subgraph of $H_{\geq i}^{\max}$, this is at most

$$\frac{\varepsilon}{2\gamma} \cdot \left(2 \cdot w_i^E(\delta_{E_{\geq i}^{\max}}(S))\right) = \frac{\varepsilon}{\gamma} w_i^E(\delta_{E_{\geq i}^{\max}}(S))$$

□

5.7. Speeding Up the Sparsifier Construction

In this section, we complete the proof of Theorem 5.1 by speeding up our algorithm so that its running time reduces to $\tilde{O}(mn + n^{10}/\varepsilon^7)$ from $\tilde{O}(Wm^2n^4)$ (Corollary 5.4.2). Note that even for unweighted case ($W = 1$), this is a significant speed-up in dense hypergraphs.

At a high-level, the idea underlying the speed up is to reduce the general weighted problem to one where both m and W are polynomially bounded in n. The first task is easy to accomplish using previously known results while the second task requires some additional ideas.

Our starting point for reducing the number of edges is the following result by Chekuri and Xu [88] which shows that the number of edges m can be reduced to a polynomial in n in near-linear time:

Lemma 5.7.1 (Corollary 6.3 of [88]). *A $(1 \pm \varepsilon)$-approximate cut sparsifier of a weighted hypergraph H with $O(n^3/\varepsilon^2)$ edges can be found in $O(mn \log^2 n \log m)$ time with high probability.*

After running this algorithm, we obtain a $(1 \pm \varepsilon)$-approximate cut sparsifier of H with only $O(n^3/\varepsilon^2)$ edges. Then we run the algorithm by Kogan and Krauthgamer [185] and get a

cut-sparsifier with $\tilde{O}(n^2/\varepsilon^2)$ edges.

Lemma 5.7.2 ([185]). *A $(1\pm\varepsilon)$-approximate cut sparsifier of a weighted hypergraph H with $\tilde{O}(n^2/\varepsilon^2)$ edges can be found in $O(mn^2+n^3)$ time with high probability.*

Since the number of hyperedges in the sparsifier given by Lemma 5.7.1 is $O(n^3/\varepsilon^2)$, we only need $\tilde{O}(n^5/\varepsilon^2)$ time to run the algorithm in Lemma 5.7.2. Let $\bar{H}=(V,\bar{E},\bar{w})$ be the sparsifier.

It is worth noting that although the number of edges in $\bar{H}$ is polynomial, the ratio of maximum and minimum weight is still unbounded. In fact, even if H is unweighted, the ratio of maximum and minimum weight of $\bar{H}$ still could be as large as 2^n. To solve this problem, we group the edges by their weights. Let $\alpha=\frac{10n^2}{\varepsilon^3}$ and $\bar{E}=E_1\cup E_2\cup\ldots$ where $E_i=\{e\in\bar{E}:\bar{w}(e)\in[w_0\cdot\alpha^{i-1},w_0\cdot\alpha^i)\}$ where w_0 is the minimum weight in $\bar{H}$.

Let $H_i=(V,E_i,\bar{w})$ and $m_i=|E_i|$. By Corollary 5.4.2, we only need $\tilde{O}(\alpha m_i^2 n^4)$ time to build a near-linear size (in n) sparsifier for each of H_i. However, if we combine these sparsifiers together, the size is no longer near-linear.

Note that $\alpha\geq\frac{10\bar{m}}{\varepsilon}$ where $\bar{m}$ is the number of edges in $\bar{H}$. Suppose a cut separates an edge e in H_i, the sum of weights of all edges in $\cup_{j\leq i-2}E_j$ is less than $\varepsilon/10$ fraction of the size of the cut. Therefore, for any i, we can ignore the performance of the sparsifier of H_j for $j\leq i-2$ within the connected components of H_i.

Define $E_{odd}=E_1\cup E_3\cup\ldots$ and $E_{even}=E_2\cup E_4\cup\ldots$. We will independently construct sparsifiers of $H_{odd}=(V,E_{odd},w)$ and $H_{even}=(V,E_{even},w)$ and merge them into a sparsifier for $\bar{H}$.

Lemma 5.7.3. *For any $0<\varepsilon<1$, there is an algorithm that constructs $(1\pm\varepsilon)$-approximate cut sparsifiers for both H_{even} and H_{odd} with size $O(\frac{n\log n}{\varepsilon^2})$ in $\tilde{O}(n^{10}/\varepsilon^7)$ time with high probability.*

Without loss of generality, we focus on H_{even}. The algorithm builds sparsifiers for each of

H_{2i} one by one from higher i to lower i. Let $E_{>2i} = \cup_{j>i} E_{2j}$ and $H_{>2i} = (V, E_{>2i}, \bar{w})$. For each i, we first find all connected components of $H_{>2i}$. Let V_{2i}^C be a vertex set such that each connected component (including isolated vertices) of $H_{>2i}$ is a "supervertex" in V_{2i}^C. Let E_{2i}^C be the hyperedge set such that for each edge $e \in E_{2i}$, E_{2i}^C contains the hyperedge $e' \subseteq V_i^C$ with weight $\bar{w}(e') = \bar{w}(e)$ that contains all vertices in V_i^C such that e contains a vertex in the corresponding connected component. Let $H_{2i}^C = (V_{2i}^C, E_{2i}^C, \bar{w})$.

For each connected component of H_{2i}^C, we build a $(1 \pm \frac{\varepsilon}{2})$-approximate cut sparsifier by the algorithm in Corollary 5.4.2. We take the union of these sparsifiers and get an $\frac{\varepsilon}{2}$-sparsifier $\hat{H}_{2i}^C = (V_{2i}^C, \hat{E}_{2i}^C, \hat{w})$ of H_{2i}^C. Let $\hat{H}_{2i} = (V, \hat{E}_{2i}, \hat{w})$ be the graph "restored" from $\hat{H}_{2i}^C$, i.e. for each edge e in E_{2i}, e is in $\hat{E}_{2i}$ if the corresponding edge e' is in $\hat{H}_{2i}^C$. It also gets the same weight as e' if it is included in $\hat{H}_{2i}$. For any cut $(S, \bar{S})$ of V_{2i} which does not cut any component in $H_{>2i}$, the cut size in $\hat{H}_{2i}$ and $\hat{H}_{2i}^C$ are the same, and the cut size in H_{2i} and H_{2i}^C are the same. In particular, this implies that $\hat{H}_{2i}$ is a good sparsifier of H_{2i} with respect to all cuts that do not cut any component in $H_{>2i}$.

Output $\hat{H}_{even} = \cup_i \hat{H}_{2i}$ as a sparsifier of H_{even}. By Corollary 5.4.2, the running time is

$$\sum_i \tilde{O}(\alpha m_i^2 n^4) = \tilde{O}((\sum_i m_i)^2 \alpha n^4) = \tilde{O}(\alpha \bar{m}^2 n^4) = \tilde{O}(n^{10}/\varepsilon^7)$$

Now we prove $\hat{H}_{even}$ is indeed a good cut sparsifier of H_{even}. From this point, we assume the algorithm in Corollary 5.4.2 is always successful throughout the algorithm (which happens with high probability). We first prove that $\hat{H}_{even}$ is indeed a $(1 \pm \varepsilon)$-approximate cut sparsifier of H_{even}.

Claim 5.7.4. *$\hat{H}_{even}$ is a $(1 \pm \varepsilon)$-approximate cut sparsifier of H_{even}.*

Proof. We first prove that for any i, $\hat{w}(\hat{E}_{2i}) \leq 3\bar{w}(E_{2i})$. Equivalently, we prove that $\hat{w}(\hat{E}_{2i}^C) \leq 3\bar{w}(E_{2i}^C)$. Let $(S', \bar{S}')$ be some cut of $\hat{H}_{2i}^C$ of weight at least $\hat{w}(\hat{E}_{2i}^C)/2$. Such a cut must exist because the expected weight of a random cut of a graph/hypergraph is at least half of

the total weight of the graph. Since $\hat{H}_{2i}^C$ is a $(1 \pm \frac{\varepsilon}{2})$-approximate cut sparsifier of H_{2i}^C, $\hat{w}(\delta_{\hat{H}_{2i}^C}(S')) \leq (1 + \frac{\varepsilon}{2}) \cdot \bar{w}(\delta_{H_{2i}^C}(S')) \leq 1.5 \cdot \bar{w}(E_{2i}^C)$ since $\varepsilon < 1$. Therefore $\hat{w}(\hat{E}_{2i}^C)/2 \leq 1.5 \cdot \bar{w}(E_{2i}^C)$, concluding the proof.

Now fix any cut $C = (S, \bar{S})$ of V. Let i be the largest integer such that $\delta_{E_{2i}}(S) \neq \emptyset$. Since $\alpha \geq \frac{10\bar{m}}{\varepsilon}$, $\bar{w}(\delta_{E_{2i}}(S))$ is at least $(1 - \frac{\varepsilon}{10})$ fraction of $\bar{w}(\delta_{E_{even}}(S))$.

Since C does not cut through any component of $H_{>2i}$, $\hat{w}(\delta_{\hat{H}_{2i}}(S))$ is within $(1 \pm \frac{\varepsilon}{2})$ fraction of $\hat{w}(\delta_{H_{2i}}(S))$, which means

$$\hat{w}(\delta_{\hat{H}_{even}}(S)) \geq \hat{w}(\delta_{\hat{H}_{2i}}(S)) \geq (1 - 0.5\varepsilon)\bar{w}(\delta_{H_{2i}}(S)) \geq (1 - \varepsilon)\bar{w}(\delta_{\bar{H}_{even}}(S))$$

On the other hand, since $\alpha \geq \frac{10\bar{m}}{\varepsilon}$ and $\hat{w}(\hat{E}_{2j}) \leq 3\bar{w}(E_{2j})$ for any j, we have $\hat{w}(\cup_{j<i}\hat{E}_{2j}) < 0.3\varepsilon \cdot \bar{w}(\delta_{H_{even}}(S))$. which means

$$\begin{aligned}
\hat{w}(\delta_{\hat{H}_{even}}(S)) &\leq \hat{w}(\delta_{\hat{H}_{2i}}(S)) + 0.3\varepsilon \cdot \bar{w}(\delta_{E_{even}}(S)) \\
&\leq (1 + 0.5\varepsilon)\bar{w}(\delta_{H_{2i}}(S)) + 0.3\varepsilon \cdot \bar{w}(\delta_{E_{even}}(S)) \\
&\leq (1 + \varepsilon)\bar{w}(\delta_{\bar{H}_{even}}(S))
\end{aligned}$$

□

The next claim shows that $\hat{H}_{even}$ has near linear size.

Claim 5.7.5. *The size of $\hat{H}_{even}$ is $O(\frac{n \log n}{\varepsilon^2})$.*

Proof. For any $i > 0$, let $\Delta_i = |V_{>2i}| - \left|V_{>2(i-1)}\right|$ for all $i > 0$ and let $|V_{>0}|$ be the number of connected components in H_{even}. To prove the claim, it is sufficient to prove that $\left|\hat{E}_{2i}\right| = O(\frac{\Delta_i \log n}{\varepsilon^2})$ for all $i > 0$.

Suppose there are ℓ connected components in H_{2i}^C and their sizes are $n_{i1}, n_{i2}, \ldots, n_{i\ell}$. For any j, if $n_{ij} > 1$, then $2(n_{ij} - 1) \geq n_{ij}$, so the size of the sparsifier of this component

is $O(\frac{(n_{ij}-1)\log n}{\varepsilon^2})$ by Corollary 5.4.2. On the other hand, if $n_{ij} = 1$, the component is an isolated vertex and we do not need to find a sparsifier for this component. So the total size of these sparsifiers is $\left|\hat{E}_{2i}\right| = O(\frac{\sum_{j=1}^{\ell}(n_{ij}-1)\log n}{\varepsilon^2})$.

For each component of H_{2i}^C of size n_{ij}, the vertices in the component will contract to one single vertex in $V_{>2(i-1)}^C$, which means

$$\left|V_{>2(i-1)}^C\right| = \ell = \sum_{j=1}^{\ell}(n_{ij} - (n_{ij}-1)) = \left|V_{>2i}^C\right| - \sum_{j=1}^{\ell}(n_{ij}-1)$$

So $\sum_{j-1}^{\ell}(n_{ij}-1) = \Delta_i$, implying that $\left|\hat{E}_{2i}\right| = O(\frac{\Delta_i \log n}{\epsilon^2})$. □

Lemma 5.7.3 immediately follows from Claim 5.7.4 and Claim 5.7.5. Now we are ready to prove Theorem 5.1.

Proof of Theorem 5.1. We first apply the algorithm in Lemma 5.7.1 and Lemma 5.7.2 to build $\bar{H}$, which runs in time $\tilde{O}(mn+n^5/\varepsilon^2)$. Then we build the graphs H_{even} and H_{odd}, find $(1\pm\varepsilon)$-approximate cut sparsifiers with size $O(\frac{n\log n}{\varepsilon^2})$ for each of them and take the union of these two sparsifiers to get a $(1\pm\varepsilon)$-approximate cut sparsifier $\hat{H}$ of $\bar{H}$. By Lemma 5.7.3, this runs in time $\tilde{O}(n^{10}/\varepsilon^7)$. So we get a $(1\pm O(\varepsilon))$-approximate cut sparsifier $\hat{H}$ of H with size $O(\frac{n\log n}{\varepsilon^2})$, in $\tilde{O}(mn+n^{10}/\varepsilon^7)$ time. □

5.8. Sublinear Time Cut Sparsification with Cut Size and Cut Edge Sampling Queries

We now present an algorithm that, given access to a hypergraph H through cut size queries (oracle O_{value}) and queries to sample a random edge crossing a cut (oracle O_{edge}), outputs a $(1\pm\varepsilon)$-approximate sparsifier with $\tilde{O}(n/\varepsilon^2)$ hyperedges in poly(n) time. At a high-level, our algorithm will first create a poly(n) size sparsifier H_1 by indirectly implementing the algorithm underlying Theorem 5.6. We then use the algorithm in Theorem 5.1 to construct a sparsifier H_2 of H_1 which has $\tilde{O}(n/\varepsilon^2)$ hyperedges. By the definition of cut sparsifier, H_2

is also a cut sparsifier of H. We can thus focus on the construction of the sparsifier H_1.

The primary challenge in simulating the algorithm of Theorem 5.6 is to sample edges according to their strength with a small number of queries. Consider the following recursive algorithm. We start with the graph H, and then at each step, we find the minimum cut of the connected graph, and sample $\Theta((r + \log n)/\varepsilon^2)$ edges from the cut. We then recursively execute this algorithm on each side of the cut. Algorithm StrengthSampling gives an implementation of this idea.

StrengthSampling: Sampling edges with probability proportional to their strength

1. Let $(S, \bar{S})$ be a minimum cut of the induced graph $G[V']$
2. Let c be the number of edges crossing $(S, \bar{S})$ in $G[V']$
3. Sample an integer $N \sim B(c, \frac{10(\log n+r)}{\epsilon^2 c})$
4. Sample N edges from $\delta_{G[V']}(S)$ uniformly at random, and assign each of them a weight of $\frac{\epsilon^2 c}{10(\log n+r)}$
5. Delete all edges in $\delta_{G[V']}(S)$ and recurse on each of the newly created connected components

It is easy to see that this algorithm samples each edge independently, and that the sampling probability is at least that of Kogan-Krauthgamer in Theorem 5.6. The challenge is that unlike cut queries in the normal graph, it is hard to compute the cut size in an induced subgraph of a hypergraph using only cut queries on the original graph, which is crucial as the algorithm proceeds recursively.

We first note that this task is straightforward to do in graphs where each edge has exactly two vertices. For any two disjoint subsets of vertices S, T, the number of edges in $S \times T$ is $\frac{1}{2}(|\delta(S)| + |\delta(T)| - |\delta(S \cup T)|)$.

However, this is far from true in the hypergraph setting. The problem is that there may be some hyperedges that intersect with each of S, T, and $V \setminus (S \cup T)$. These edges are inside

all of $\delta(S)$, $\delta(T)$ and $\delta(S \cup T)$. We have

$$\begin{aligned}
&|\delta(S)| + |\delta(T)| - |\delta(S \cup T)| \\
=&2\left|\{e | e \cap S \neq \emptyset, e \cap T \neq \emptyset, e \cap (\overline{S \cup T}) = \emptyset\}\right| + \left|\{e | e \cap S \neq \emptyset, e \cap T \neq \emptyset, e \cap (\overline{S \cup T}) \neq \emptyset\}\right| \\
=&|\{e | e \cap S \neq \emptyset, e \cap T \neq \emptyset\}| + \left|\{e | e \cap S \neq \emptyset, e \cap T \neq \emptyset, e \cap (\overline{S \cup T}) = \emptyset\}\right|
\end{aligned}$$

Example 5.3. *Consider a hypergraph H that consists of three equal size sets of vertices A, B, C, such that each hyperedge has a non-empty intersection with each of A, B, and C. Then there are no hyperedges in $H[A \cup B]$. But the quantity $\frac{1}{2}(|\delta(A)| + |\delta(B)| - |\delta(A \cup B)|)$ is half the total number of hyperedges which could be exponentially large in n.*

Example 5.4. *Consider the following pair of hypergraphs on 4 vertices, say $\{v_1, v_2, v_3, v_4\}$: the graph H_1 is a (rank 2) clique on 4 vertices while the graph H_2 contains every possible edge of size 3 on these 4 vertices. It is easy to verify that the answer to every cut query is the same on the graphs H_1 and H_2. Now consider the subgraph of these graphs induced by the vertices $X = \{v_1, v_2\}$. In case of H_1, the minimum cut in the induced subgraph is 1 while in H_2, the minimum cut in the graph induced by X is 0. We can amplify this gap to 0 versus $\Omega(n)$ by taking $n/4$ copies of H_1 in one case, and $n/4$ copies of H_2 in the other case, and defining X to be union of arbitrarily chosen pairs of vertices from each copy. This means that O_{value} queries can not be used to estimate cut size in induced subgraphs to any multiplicative factor or to better than a polynomial additive error.*

To get around the challenge highlighted by examples above, we next introduce notions of *pseudo cut size* over a subset of vertices and *pseudo strength* of hyperedges, such that the pseudo cut sizes are easy to compute by cut queries and pseudo strength of any hyperedge is at most a factor n larger than the strength of the hyperedge. We develop these ideas in detail in the next subsection.

5.8.1. Pseudo Cuts and Pseudo Strengths

Given a set of vertices X, we define $\Delta_X(S)$, the *pseudo cut* size of a set $S \subset X$ as $\frac{1}{2}(|\delta(S)| + |\delta(X \setminus S)| - |\delta(X)|)$, and define the *pseudo min cut* over X as a cut $(S, X \setminus S)$ that minimizes $\Delta_X(S)$. Note that $\Delta_X(S)$ is at most the number of edges that intersect both S and $X \setminus S$. The following lemma shows that $\Delta_X(S)$ is a submodular function, so we can compute the pseudo min cut over any vertex set in poly(n) time by Theorem 2.1.

Lemma 5.8.1. *For any vertex set $X \subseteq V$, $\Delta_X(S)$ is a submodular function.*

Proof. Let $f_1(S)$ be the number of edges that intersect both S and $X \setminus S$, and let $f_2(S)$ be the number of edges that intersect both S and $X \setminus S$ but are fully contained in X. By definition, we have $\Delta_X(S) = \frac{1}{2}(f_1(S) + f_2(S))$, so to prove that $\Delta_X(S)$ is a submodular function, it is sufficient to prove that both f_1 and f_2 are submodular.

Since f_2 is the cut function in the induced graph $H[X]$, it is submodular. In fact, f_1 is also the cut function of the hypergraph whose vertex set is X and edge set is $\{e \cap X | e \in E\}$. So f_1 is also a submodular function. □

For any edge e, we define the *pseudo strength* k'_e as the largest pseudo min-cut size among all sets X that contain e, where X ranges over all subsets of V. It is easy to see that for any edge e, k'_e is at least k_e since for any set of vertices X, the minimum cut size of $H[X]$ is at most the pseudo min-cut size of set X. More interestingly, although Example 5.3 showed that the pseudo min-cut size of a set X may be arbitrarily larger than the minimum cut size of $H[X]$, the lemma below shows that the pseudo strength of an edge is at most a factor n larger than its strength.

Lemma 5.8.2. *For any edge e, $k'_e \leq nk_e$.*

Proof. Let X be any set of vertices that contains the edge e and has pseudo min-cut size k'_e in H. To prove the lemma, it is sufficient to prove that the pseudo min-cut size of X in H

is at most nk_e.

Let $Y = V$ and $E_c = \emptyset$. Consider the following iterative process: we find the minimum cut $(S, Y \setminus S)$ in $H[Y]$. If either S or $Y \setminus S$ fully contains the set X, we add all edges crossing the cut into E_c, and set Y to be S or $Y \setminus S$ (whichever fully contains X), and repeat. Otherwise, we stop the process.

After the process terminates, suppose $(S, Y \setminus S)$ is the minimum cut in $H[Y]$. Since the process terminated, $(S, Y \setminus S)$ must partition X. Let $S' = S \cap X$, and consider the pseudo cut $(S', X \setminus S')$. We prove that the number of edges in H that intersect both S' and $X \setminus S'$ (which is an upper bound on $\Delta_X(S')$) is at most nk_e.

First, note that no edge e' such that $e' \not\subseteq Y$ and $e' \notin E_c$ can intersect with the set Y; hence any such edge e' also does not intersect with S' or $X \setminus S'$. Therefore every edge that intersects with both S' and $X \setminus S'$ either belongs to E_c or is completely contained in Y. During the iterative process, the set Y always fully contains e, so by the definition of strength, the minimum cut size of $H[Y]$ is at most k_e. This implies during each step, at most k_e edges are added into E_c. On the other hand, the process repeats at most $n - 2$ times, since each time the size of Y is reduced by at least 1. So $|E_c| \leq (n-2)k_e$. Finally, any edge that is fully contained in Y and intersects with both S' and $X \setminus S'$ crosses the cut $(S, Y \setminus S)$ in $H[Y]$, and the number of such edges is at most the minimum cut size of $H[Y]$, which is at most k_e. So in total, there are at most $|E_c| + k_e \leq (n-1)k_e$ edges that intersect both S' and $X \setminus S'$. □

5.8.2. Sampling the Edges

We are now ready to present an algorithm that uses the cut size queries and cut edge sample queries to sample each edge with probability inversely proportional to its strength. Specifically, we will ensure that each edge e gets sampled with probability at least n^2/k'_e which is at least n/k_e by Lemma 5.8.2. The algorithm is similar to StrengthSampling, but uses pseudo cuts and pseudo strengths instead. To sample the edges, we call PesudoStrengthSampling on

set V.

PesudoStrengthSampling: Sampling edges with probability proportional to their pseudo strength

1. Find the pseudo min-cut $(S, V' \setminus S)$ within the set V'
2. Let c be $|\delta(S)|$, the cut size of $(S, V \setminus S)$
3. Sample an integer $N \sim B(c, \min\{1, \frac{10n^3}{\epsilon^2 c}\})$
4. Keep sampling edges in cut $(S, \bar{S})$ until we get N different hyperedges.
5. Recurse on both S and $V' \setminus S$

We now prove that each edge gets sampled with probability at least as large as the sampling probability in Theorem 5.6. Fix an edge e, let S_1 be the last input set that fully contains e. For any $i \geq 1$, if S_i is not V, we define S_{i+1} to be the input set in the recursion that generates a recursive call of the algorithm on the set S_i. In other word, $(S_i, S_{i+1} \setminus S_i)$ is the pseudo min cut within set S_{i+1}. Let $(S_0, S_1 \setminus S_0)$ be the pseudo min cut within S_1, by definition, $e \cap S_0 \neq \emptyset$ and $e \cap S_1 \setminus S_0 \neq \emptyset$. When the algorithm works on set S_1, e gets sampled with probability $\min\{1, \frac{10n^3}{\varepsilon^2 |\delta(S_0)|}\}$. If e gets sampled with probability 1, then it is clearly as large as the probability in Theorem 5.6. Otherwise we need to prove that $n^3 / |\delta(S_0)| = \Omega((\log n + r)/k_e)$. Since $n = \Omega(\log n + r)$, by Lemma 5.8.2, it is sufficient to prove that $|\delta(S_0)| \leq nk'_e$.

Lemma 5.8.3. $|\delta(S_0)| \leq nk'_e$.

Proof. We partition the edges crossing the cut $(S_0, \overline{S_0})$ into sets $E_1, E_2, \ldots$ such that for any $i \geq 0$, E_i is the set of edges that are fully contained in S_{i+1} but not in S_i. Note that $|\delta(S_0)| = \sum_i |E_i|$. Since the algorithm has at most n levels of recursion, to prove the lemma, it is sufficient to prove $|E_i| \leq k'_e$ for all $i \geq 0$.

For any edge $e' \in E_i$, $e' \cap S_i \neq \emptyset$ since e' crosses the cut $(S_0, \overline{S_0})$ and $S_0 \subseteq S_i$. We also have $e' \cap S_{i+1} \setminus S_i \neq \emptyset$ and $e' \cap \overline{S_{i+1}} = \emptyset$ since e' is fully contained in S_{i+1} but not S_i. So

$|E_i| \leq \Delta_{S_{i+1}}(S_i)$. On the other hand, by definition of pseudo strength, $k'_e \geq \Delta_{S_{i+1}}(S_i)$ since e is fully contained in S_{i+1}. Therefore, $|E_i| \leq k'_e$. □

By Lemma 5.8.3, we proved that each edge e is sampled with probability at least the required probability in Theorem 5.6. Next, we need to assign weights to each sampled edge.

We do this after we finish sampling. For each edge e that gets sampled, we need to know the probability that it gets sampled. Since we sample edges from each cut independently, we only need to know the probability that e gets sampled during each recursive call, and that probability depends only on the size of the cut and whether e crosses the cut. So we can compute the probability that e gets sampled during PesudoStrengthSampling.

To complete the proof of Theorem 5.2, we need to show that the running time of the whole process is polynomial in n.

Proof of Theorem 5.2. During each call to PesudoStrengthSampling, we need $\tilde{O}(n^3)$ queries to cut size query oracle and $\tilde{O}(n^4)$ time to figure out the pseudo min-cut within the set V' by Theorem 2.1 and Lemma 5.8.1. At line 4, we call cut edge sample query $10n^3/\varepsilon^2$ times in expectation. Total number of recursive calls to PesudoStrengthSampling is $O(n)$, since each time, the input set gets partitioned into two sets, and there are n sets in the end. Thus the running time of PesudoStrengthSampling is $\tilde{O}(n^5 + n^4/\varepsilon^2)$.

We sample the edges in $O(n)$ cuts, so when assigning the weights, we only need to query the size of these $O(n)$ cuts and calculate the probability of each sampled edge, which can also be done in $O(n)$ time for each edge. So the running time of assigning the weights is $\tilde{O}(n^5/\varepsilon^2)$.

After sampling the edges and assigning weights, we get a $(1 \pm \varepsilon)$-approximate cut sparsifier H_1 of H with polynomial size in n. Then we run the algorithm in Theorem 5.1 to find a $(1 + \varepsilon)$-approximate cut sparsifier H_2 of H_1 with $\tilde{O}(n/\varepsilon^2)$ number of edges in polynomial time in n. By definition of cut sparsifier, H_2 is a $(1 \pm \varepsilon)^2$-approximate cut sparsifier of H.

Since H_1 contains $\tilde{O}(n^4/\varepsilon^2)$ edges, by Theorem 5.1, the running time is $O(n^{10}/\varepsilon^7)$.

So the total running time is $O(n^{10}/\varepsilon^7)$. □

5.9. Sublinear Time Cut Sparsification with Cut Size and Pair Neighbor Queries

In this section, we show that cut edge queries can be simulated by a poly(n) number of cut size queries (oracle O_{value}) and pair neighbor queries (oracle O^2_{nbr}), establishing that cut size query oracle and pair neighbor query oracle are also sufficient to compute a $(1 \pm \varepsilon)$-approximate cut sparsifier in poly(n) time.

Given a pair of vertices u and v, let $E(\{u, v\})$ be the set of edges that contain both u and v. We first show how to approximate $|E(\{u, v\})|$ to within a factor of $(1 \pm \varepsilon)$ with probability $1-\xi$, for some small ξ. Note that we can compute $2\Delta_{\{u,v\}}(\{u\}) = |E(\{u, v\})| + |E \cap \{u, v\}|$, where $|E \cap \{u, v\}|$ is the number of copies of the edge $\{u, v\}$. We now describe an algorithm to approximate $|E(\{u, v\})|$:

NeighborApproximation: Approximating $|E(\{u, v\})|$

1. Define $k = 12\log(2/\xi)/(\varepsilon^2)$.
2. Call the oracle O^2_{nbr} k times on (u, v), and let $\hat{\alpha}$ be the fraction of returned edges that were $\{u, v\}$.
3. Return $\hat{E}(\{u, v\}) := 2\Delta_{\{u,v\}}(\{u\}) \cdot \frac{1}{1+\hat{\alpha}}$.

Note that this algorithm makes $k = O(\frac{\log(1/\xi)}{\varepsilon^2})$ queries.

Lemma 5.9.1. *With probability at least* $1-\xi$, $\hat{E}(\{u, v\})$ *is an approximation of* $|E(\{u, v\})|$ *to within a factor of* $(1 \pm \varepsilon)$.

Proof. Let $\alpha := \frac{|E\cap\{u,v\}|}{|E(\{u,v\})|}$ be the fraction of hyperedges that are $\{u, v\}$. The algorithm runs a Monte Carlo simulation to approximate α by the ratio $\hat{\alpha}$. In order to prove concentration of $\hat{\alpha}$ around α, let k' be the total number of $\{u, v\}$ edges returned, and observe that k'

is the sum of k independent Bernoulli random variables each having probability equal to α. By Chernoff bound, $\Pr[|k' - \alpha k| > \varepsilon k/2] \leq 2\exp(-\alpha k\varepsilon^2/12\alpha) \leq 2\exp(-k\varepsilon^2/12) = 2\exp(\log(\xi/2)) = \xi$. Therefore with probability at least $1-\xi$, $|\hat{\alpha} - \alpha| \leq \varepsilon/2$. This implies that $\frac{1}{1+\hat{\alpha}} \in \frac{1}{1+\alpha\pm\varepsilon/2} \subseteq \frac{1}{(1\pm\varepsilon/2)(1+\alpha)}$. Finally, we use that $\frac{1}{(1\mp\varepsilon/2)} \subseteq (1 \pm \varepsilon)$ to conclude that $\frac{1}{1+\hat{\alpha}} \in \frac{1\pm\varepsilon}{1+\alpha}$, so

$$\frac{2\Delta_{\{u,v\}}(\{u\})}{1+\hat{\alpha}} \in (1\pm\varepsilon)\frac{2\Delta_{\{u,v\}}(\{u\})}{1+\alpha} = (1\pm\varepsilon)|E(\{u,v\})|.$$

□

We now describe an algorithm to sample a random edge from $\delta(S)$, simulating a response to O_{edge}. We first approximate the size of $E(\{u,v\})$ for each pair of vertices $u \in S$ and $v \in \bar{S}$. Then we sample a pair of u, v with probability proportional to $|E(\{u,v\})|$, sample an edge in $E(\{u,v\})$, and then decide whether we keep it or not with probability proportional to its size. If we decide not to pick the edge, we repeat the whole process again.

NeighborSampling: Sampling an edge in $\delta(S)$

1. For each pair of vertices u, v such that $u \in S$ and $v \in \bar{S}$, call NeighborApproximation with $\xi = 1/n^{20}$, and let $\hat{E}(\{u,v\})$ be the output
2. Sample a pair of vertices $(u,v) \in S \times \bar{S}$ with probability proportional to $\hat{E}(\{u,v\})$
3. Use the oracle O^2_{nbr} to sample an edge e in $E(\{u,v\})$
4. With probability $\frac{1}{|e\cap S|\cdot|e\cap\bar{S}|}$, return e. Otherwise go to Step 2.

Lemma 5.9.2. *With probability at least $1 - 1/n^{-10}$, NeighborSampling samples each edge in $\delta(S)$ gets with probability $\frac{1\pm\varepsilon}{|\delta(S)|}$. The expected running time is $\tilde{O}(n^2/\varepsilon^2)$.*

Proof. We first condition on the $|S| \cdot |\bar{S}| \leq n^2$ events that for each pair u, v with $u \in S$ and $v \in \bar{S}$, the estimate $\hat{E}(\{u,v\})$ was indeed in $(1 \pm \varepsilon) \cdot |E(\{u,v\})|$, which happens with probability at least $1 - n^2\xi > 1 - 1/n^{-10}$. Now fix an edge $e \in \delta(S)$. The probability that

it was sampled at a particular iteration of NeighborSampling is

$$\sum_{u\in S\cap e, v\in \bar{S}\cap e} \frac{\hat{E}(\{u,v\})}{\sum_{(u',v')\in S\times\bar{S}} \hat{E}(\{u',v'\})} \cdot \frac{1}{|E(\{u,v\})|} \cdot \frac{1}{|e\cap S|\cdot|e\cap\bar{S}|}$$

$$\in \frac{1}{\sum_{(u',v')\in S\times\bar{S}} \hat{E}(\{u',v'\})} \sum_{u\in S\cap e, v\in\bar{S}\cap e} \frac{(1\pm\varepsilon)}{|e\cap S|\cdot|e\cap\bar{S}|} = \frac{(1\pm\varepsilon)}{\sum_{(u',v')\in S\times\bar{S}} \hat{E}(\{u',v'\})}$$

That is, the probability of sampling each edge at any given iteration of NeighborSampling is within $(1\pm\varepsilon)$ of every other edge. Therefore the probability of sampling each edge is within a factor of $(1\pm\varepsilon)$ of every other edge.

Step 1 calls NeighborApproximation $O(n^2)$ times, so the running time is $\tilde{O}(n^2/\varepsilon^2)$. At step 4, the probability that we keep the edge and finish the algorithm is at least $1/n^2$, so the expected number of iterations through step 2 to 4 is at most n^2. So the total running time on step 2 to 4 is $\tilde{O}(n^2)$ in expectation. □

Proof of Theorem 5.4. We run PesudoStrengthSampling, but each time it calls O_{edge}, we instead run NeighborSampling twice. With high probability, each time we simulate O_{edge} by NeighborSampling, the probability of any edge in the cut being sampled is within a $(1\pm\varepsilon)$ factor of the uniform distribution. Denote by q'_e be the probability that an edge e is sampled by this algorithm, and let q_e be the probability that the edge e is sampled in PesudoStrengthSampling. We have $q'_e \in 2(1\pm\varepsilon)q_e$, which is larger than p_e the probability of sampling an edge in Theorem 5.6. Also we cannot directly compute q'_e, but we can approximate it to within a factor of $(1\pm\varepsilon)$, which only adds another $(1\pm\varepsilon)$ factor to the approximation achieved by the cut sparsifier.

Since the number of calls to O_{edge} oracle in PesudoStrengthSampling is $\tilde{O}(n^4/\varepsilon^2)$. So we need $\tilde{O}(n^6/\varepsilon^2) = o(n^{10}/\varepsilon^7)$ queries to simulate these calls. So the running time of the algorithm is still $O(n^{10}/\varepsilon^7)$. □

5.10. Lower Bounds

In this section we show that any natural relaxation of the assumptions underlying Theorem 5.2 and Theorem 5.4 rules out poly(n) time sparsification algorithms, proving Theorem 5.3 and Theorem 5.5.

5.10.1. Queries O_{value} and O^1_{nbr} Together are not Sufficient

In this section, we prove that if any randomized algorithm can only access the underlying hypergraph via O_{value} and O^1_{nbr}, it is not possible to find with probability better than $o(1)$ a $(1 \pm \varepsilon)$-approximate cut sparsifier with only poly(n) queries, proving Theorem 5.5. We start by showing a weaker result, as stated in the lemma below, which shows that the failure probability of a poly(n) time algorithm must be at least $1/2 - o(1)$, and then show how to amplify the failure probability to $1 - o(1)$.

Lemma 5.10.1. *There is no polynomial time algorithm that can use O_{value} and O^1_{nbr} queries alone to construct a $(1 \pm \varepsilon)$-approximate sparsifier of an underlying hypergraph H with probability at least $1/2 + \xi$ for any constant $\xi > 0$.*

Proof. Suppose the runtime of the algorithm is bounded by some polynomial $f(n)$. We will construct two graphs $H_1 = (V \cup V', E_1)$ and $H_2 = (V \cup V', E_2)$ with $|V| = |V'| = n$ and the algorithm is shown with probability $1/2$ the graph H_1 and with probability $1/2$ the graph H_2. We will then show that (a) any algorithm that can only access the underlying graph using O_{value} and O^1_{nbr} cannot distinguish between these two graphs with probability at least $1/2 + \xi$ for any constant $\xi > 0$, and (b) there exists a non-empty cut such that H_1 and H_2 do not have any common edges crossing the cut. Together, these properties immediately imply the lemma .

Let $u, v \in V$ and $u', v' \in V'$ be two arbitrary pairs of vertices. Let $E = 2^V \cup 2^{V'}$ be the union of the complete hypergraphs on V and V'. We define E_1 as E along with all possible edges of size two among $\{u, v, u', v'\}$. We define E_2 as E along with all possible edges of size 3 among $\{u, v, u', v'\}$. It is easy to verify that for any cut, the number of edges in E_1

crossing the cut equals the number of edges in E_2 crossing the cut. So any cut size query O_{value} has the same answer in H_1 and H_2, and hence can not distinguish between these two graphs, no matter the number of queries allowed.

The algorithm can additionally make at most $f(n)$ calls to O^1_{nbr}. But since each vertex $w \in V \cup V'$ has at least 2^n edges incident on it, the probability that a uniformly random edge incident on w is not in E is at most $3/2^n$. Using a union bound over all $f(n)$ queries along with the fact that $3f(n)/2^n \leq \xi$ for sufficiently large n, we get that for both hypergraphs, with probability at least $1 - 3f(n)/2^n \geq 1 - \xi$, all sampled edges are in E.

Thus conditioned on the event that all of the sampled edges are in E, the algorithm cannot distinguish between H_1 and H_2. On the other hand, there are no common edges crossing the cut (V, V') in H_1 and H_2, so to output a proper $(1 \pm \varepsilon)$-approximate cut sparsifier, the algorithm must distinguish between H_1 and H_2. Hence the probability that algorithm succeeds is at most $1/2 + \xi$. □

To amplify the failure probability to $1 - o(1)$, we can independently generate $\log n$ instances from the distribution above with each instance containing $n/\log n$ vertices. We now let our underlying graph be a union of these $\log n$ instances. Any algorithm that outputs a $(1 \pm \varepsilon)$-approximate sparsifier, must successfully identify for each of the $\log n$ instances whether it is an instance of H_1 or H_2. Thus the probability of success is at most $(1/2 + o(1))^{\log n} = o(1)$. This completes the proof of Theorem 5.5.

5.10.2. O_{edge} Queries Alone are not Sufficient

In this section, we prove that if the algorithm can access the hypergraph through only O_{edge} queries, it is not possible to find a proper $(1 \pm \varepsilon)$-approximate cut sparsifier with $\text{poly}(n)$ queries with success probability better than $o(1)$, proving Theorem 5.3. As above, we start by showing a weaker result, which shows that the failure probability of a $\text{poly}(n)$ time algorithm must be at least $1/2 - o(1)$, and then show how to amplify the failure probability to $1 - o(1)$.

We first define two distributions of hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ such that for any sequence of the queries the algorithm asks to O_{edge}, the distribution of the answers are almost identical regardless of whether the graph was chosen from $\mathcal{H}_1$ or $\mathcal{H}_2$.

A graph in each of the distributions $\mathcal{H}_1$ and $\mathcal{H}_2$ is generated as follows. There are $n+1$ vertices $v_0, v_1, \ldots, v_n$ and the generated graph will have $2^n - n - 1$ edges. If the graph is generated by $\mathcal{H}_1$, then we randomly choose $2^{n/2}$ subsets of $\{v_1, \ldots, v_n\}$ with size at least 2. If the graph is generated by $\mathcal{H}_2$, then we randomly choose $2^{n/4}$ subsets of $\{v_1, \ldots, v_n\}$ with size at least 2. Then for any subset S of $\{v_1, \ldots, v_n\}$ of size at least 2, if S is chosen in the previous step, then the edge $S \cup \{v_0\}$ is in the graph, otherwise the edge S is in the graph.

The algorithm is presented with probability $1/2$ a graph H generated by $\mathcal{H}_1$, and with probability $1/2$ a graph H generated by $\mathcal{H}_2$, that is, the algorithm sees a graph H generated by the distribution $1/2\mathcal{H}_1 + 1/2\mathcal{H}_2$. Since the cut sizes of $(\{v_0\}, \overline{\{v_0\}})$ in the graph generated by $\mathcal{H}_1$ and $\mathcal{H}_2$ are $2^{n/2}$ and $2^{n/4}$ respectively, any algorithm that outputs a $(1 \pm \varepsilon)$-approximate cut sparsifier with $\varepsilon < 1$ must be able to distinguish between the graphs generated by $\mathcal{H}_1$ and $\mathcal{H}_2$. However, the following lemma shows that unless the algorithm makes exponential number of queries, it cannot distinguish between the graphs generated by $\mathcal{H}_1$ and $\mathcal{H}_2$.

Lemma 5.10.2. *Any algorithm that only makes k O_{edge} queries where $k = \text{Poly}(n)$ cannot determine with probability better than $\frac{1}{2} + \frac{k^2}{2^{n/4}}$ if the underlying graph H is generated from $\mathcal{H}_1$ or $\mathcal{H}_2$.*

Thus any algorithm that makes only $\text{poly}(n)$ O_{edge} queries, fails with probability at least $1/2 - o(1)$. To amplify the failure probability to $1 - o(1)$, we can as before independently generate $\log n$ instances from the distribution above with each instance containing $n/\log n$ vertices. We now let our underlying graph be a union of these $\log n$ instances. Any algorithm that outputs a $(1 \pm \varepsilon)$-approximate sparsifier, must successfully identify for each of the $\log n$ instances whether it was generated from the first distribution or the second. Thus the probability of success is at most $(1/2 + o(1))^{\log n} = o(1)$. This completes the proof of

Theorem 5.3.

5.10.3. Proof of Lemma 5.10.2

Given a sequence of k O_{edge} queries $Q = (C_1, \ldots, C_k)$, we denote by $e_1, \ldots, e_k$ the sequence of random edges that are returned by the query O_{edge}. We first prove that for any i, if we fix the first $i-1$ answers $e_1, \ldots, e_{i-1}$, then the distribution of edge e_i is almost the same irrespective of whether the underlying graph was sampled from $\mathcal{H}_1$ or $\mathcal{H}_2$. In particular, if we denote these two distributions by $\mathcal{D}_1^i$ and $\mathcal{D}_2^i$, respectively, then we will show that the total variation distance $\|\mathcal{D}_1^i - \mathcal{D}_2^i\|_{tvd} \leq \frac{i}{2^{n/4}}$.

Let $C_i = (S_i, \bar{S}_i)$ be the i_{th} cut on which the algorithm makes an O_{edge} query. Without loss of generality, assume that $v_0 \in S_i$. We first consider the case when $S_i \neq \{v_0\}$. In this case, we can sample a random edge by the following two steps: we first sample a random set $S \subseteq \{v_1, \ldots, v_n\}$ that intersects both S_i and $\bar{S}_i$, and we then return S or $S \cup \{v_0\}$ depending on which edge is in the graph. We couple the random process that samples the edge in $\mathcal{D}_1^i$ and $\mathcal{D}_2^i$, so that in the first step, these two processes sample the same set S. If S or $S \cup \{v_0\}$ is among $e_1 \ldots e_{i-1}$, then the output of O_{edge} is fixed, which means the distributions are the same in both cases. If neither S nor $S \cup \{v_0\}$ are among $e_1 \ldots e_{i-1}$ and suppose there are ℓ different edges among $e_1, \ldots, e_{i-1}$ and j of them contain v_0, then the probability that the oracle returns $S \cup \{v_0\}$ is $\frac{2^{n/2}-j}{2^n-n-1-\ell}$ when the graph is sampled from $\mathcal{H}_1$, and $\frac{2^{n/4}-j}{2^n-n-1-\ell}$ when the graph is sampled from $\mathcal{H}_2$. So

$$\|\mathcal{D}_1^i - \mathcal{D}_2^i\|_{tvd} \leq \frac{2^{n/2}-j}{2^n-n-1-\ell} - \frac{2^{n/4}-j}{2^n-n-1-\ell} < \frac{1}{2^{n/4}}$$

If $S_i = \{v_0\}$, then the oracle returns a random edge that includes v_0. Let ℓ be the number of different edges among $e_1, \ldots, e_{i-1}$ and j of them contain v_0. For any subset of $S \subseteq V$ which contains v_0 and has size at least 3, if S is an edge among $e_1 \ldots, e_{i-1}$, then $e_i = S$ with probability $\frac{1}{2^{n/2}}$ if the graph is sampled from $\mathcal{H}_1$, and with probability $\frac{1}{2^{n/4}}$ if the graph is sampled from $\mathcal{H}_2$. If $S \setminus \{v_0\}$ is among the edges $e_1, \ldots, e_{i-1}$, then the probability that

$e_i = S$ is 0 for both cases. If neither S nor $S \setminus \{v_0\}$ are among the edges $e_1, \ldots, e_{i-1}$, then if the graph is generated by $\mathcal{H}_1$, the probability that S is in the graph is $\frac{2^{n/2}-j}{2^n-n-1-\ell}$. If S is indeed in the graph, then it gets sampled with probability $\frac{1}{2^{n/2}}$. So the probability that $e_i = S$ is $\frac{1}{2^{n/2}} \cdot \frac{2^{n/2}-j}{2^n-n-1-\ell} = \frac{2^{n/2}-j}{2^{n/2}(2^n-n-1-\ell)}$. Similarly, in the case of the graph generated by $\mathcal{H}_2$, the probability that $e_i = S$ is $\frac{2^{n/4}-j}{2^{n/4}(2^n-n-1-\ell)}$. Let $X = 2^n - n - 1$, then we have

$$\begin{aligned}
2\|\mathcal{D}_1^i - \mathcal{D}_2^i\|_{tvd} =& j \cdot \left(\frac{1}{2^{n/4}} - \frac{1}{2^{n/2}}\right) + (X-\ell) \cdot \left(\frac{2^{n/2}-j}{2^{n/2}(X-\ell)} - \frac{2^{n/4}-j}{2^{n/4}(X-\ell)}\right) \\
\leq& j \cdot \frac{1}{2^{n/4}} + \left(\frac{2^{n/2}-j}{2^{n/2}} - \frac{2^{n/4}-j}{2^{n/4}}\right) \\
\leq& \frac{2j}{2^{n/4}} \leq \frac{2i}{2^{n/4}}.
\end{aligned}$$

Now we are ready to prove the lemma. Let e_i^1 and e_i^2 be the random edges sampled by O_{edge} in the i^{th} query when the graph is sampled from $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively. Given a possible answer $\mathcal{A}_k = (e_1, e_2, \ldots, e_k)$ to the k queries, denote by $\mathcal{E}_i^1(\mathcal{A}_k)$ the event that $e_1^1 = e_1, \ldots, e_i^1 = e_i$ and by $\mathcal{E}_i^2(\mathcal{A}_k)$ as the event that $e_1^2 = e_1, \ldots, e_i^2 = e_i$. Then we can bound two times the total variation distance of the distributions of the answers when the graph is generated by $\mathcal{H}_1$ and $\mathcal{H}_2$ as below:

$$\begin{aligned}
&\sum_{\mathcal{A}_k=(e_1,\ldots,e_k)} \left|\Pr\left(\mathcal{E}_k^1(\mathcal{A}_k)\right) - \Pr\left(\mathcal{E}_k^2(\mathcal{A}_k)\right)\right| \\
=& \sum_{\mathcal{A}_k=(e_1,\ldots,e_k)} \left|\Pr\left(e_1^1 = e_1, \ldots, e_k^1 = e_k\right) - \Pr\left(e_1^2 = e_1, \ldots, e_k^2 = e_k\right)\right| \\
=& \sum_{\mathcal{A}_k=(e_1,\ldots,e_k)} \left|\Pr\left(\mathcal{E}_{k-1}^1(\mathcal{A}_k)\right) \cdot \Pr\left(e_k^1 = e_k | \mathcal{E}_{k-1}^1(\mathcal{A}_k)\right) - \Pr\left(\mathcal{E}_{k-1}^2(\mathcal{A}_k)\right) \cdot \Pr\left(e_k^2 = e_k | \mathcal{E}_{k-1}^2(\mathcal{A}_k)\right)\right| \\
\leq& \sum_{\mathcal{A}_k=(e_1,\ldots,e_k)} \Big(\left|\Pr\left(\mathcal{E}_{k-1}^1(\mathcal{A}_k)\right) - \Pr\left(\mathcal{E}_{k-1}^2(\mathcal{A}_k)\right)\right| \cdot \Pr\left(e_k^1 = e_k | \mathcal{E}_{k-1}^1(\mathcal{A}_k)\right) \\
&\qquad + \Pr\left(\mathcal{E}_{k-1}^2(\mathcal{A}_k)\right) \cdot \left|\Pr\left(e_k^1 = e_k | \mathcal{E}_{k-1}^1(\mathcal{A}_k)\right) - \Pr\left(e_k^2 = e_k | \mathcal{E}_{k-1}^2(\mathcal{A}_k)\right)\right| \Big)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\mathcal{A}_{k-1}=(e_1,\ldots,e_{k-1})} \Big(\left|\Pr\left(\mathcal{E}^1_{k-1}(\mathcal{A}_{k-1})\right) - \Pr\left(\mathcal{E}^2_{k-1}(\mathcal{A}_{k-1})\right)\right| \cdot \sum_e \Pr\left(e^1_k = e | \mathcal{E}^1_{k-1}(\mathcal{A}_{k-1})\right) \\
&\qquad\qquad + \Pr\left(\mathcal{E}^2_{k-1}(\mathcal{A}_{k-1})\right) \cdot \sum_e \left|\Pr\left(e^1_k = e | \mathcal{E}^1_{k-1}(\mathcal{A}_{k-1})\right) - \Pr\left(e^2_k = e | \mathcal{E}^2_{k-1}(\mathcal{A}_{k-1})\right)\right| \Big) \\
&\leq \sum_{\mathcal{A}_{k-1}=(e_1,\ldots,e_{k-1})} \Big(\left|\Pr\left(\mathcal{E}^1_{k-1}(\mathcal{A}_{k-1})\right) - \Pr\left(\mathcal{E}^2_{k-1}(\mathcal{A}_{k-1})\right)\right| + \Pr\left(\mathcal{E}^2_{k-1}(\mathcal{A}_{k-1})\right) \cdot \frac{2k}{2^{n/4}} \Big) \\
&= \sum_{\mathcal{A}_{k-1}=(e_1,\ldots,e_{k-1})} \left|\Pr\left(\mathcal{E}^1_{k-1}(\mathcal{A}_{k-1})\right) - \Pr\left(\mathcal{E}^2_{k-1}(\mathcal{A}_{k-1})\right)\right| + \frac{2k}{2^{n/4}} \\
&\leq \sum_{\mathcal{A}_{k-2}=(e_1,\ldots,e_{k-2})} \left|\Pr\left(\mathcal{E}^1_{k-2}(\mathcal{A}_{k-2})\right) - \Pr\left(\mathcal{E}^2_{k-2}(\mathcal{A}_{k-2})\right)\right| + \frac{2(k-1)}{2^{n/4}} + \frac{2k}{2^{n/4}} \\
&\quad \ldots \\
&\leq \sum_{i=1}^{k} \frac{2i}{2^{n/4}} \leq \frac{2k^2}{2^{n/4}}
\end{aligned}
$$

Thus any algorithm that makes at most k queries can distinguish between a graph generated from $\mathcal{H}_1$ and a graph generated from $\mathcal{H}_2$ with probability at most $\frac{1}{2} + \frac{k^2}{2^{n/4}}$.

CHAPTER 6

Communication Complexity of Hidden Point Chasing Problem and its Applications

In this chapter, we give the communication complexity of hidden point chasing problem (HPC), and its application for proving round complexity of streaming graph algorithms. We start with reviewing the well known set intersection (Set-Int) problem and defining hidden-pointer chasing (HPC) problem.

Set intersection (Set-Int) is a two-player communication problem in which Alice and Bob are given sets A and B from $[n]$, respectively, with the promise that there exists a unique element t such that $\{t\} = A \cap B$. The goal is for players to find the *target element* t. An $\Omega(n)$ communication lower bound for Set-Int follows directly from lower bounds for set disjointness [30, 65, 66, 166, 230]; see, e.g. [68] (this lower bound by itself is however not useful for our application).

The hidden-pointer chasing (HPC) problem is a four-party communication problem with players P_A, P_B, P_C, and P_D. Let $\mathcal{X} := \{x_1, \ldots, x_n\}$ and $\mathcal{Y} := \{y_1, \ldots, y_n\}$ be two disjoint universes.

1. For any $x \in \mathcal{X}$, P_A and P_B are given an instance (A_x, B_x) of Set-Int over the universe $\mathcal{Y}$ where $A_x \cap B_x = \{t_x\}$ for $t_x \in \mathcal{Y}$.

2. Similarly, for any $y \in \mathcal{Y}$, P_C and P_D are given an instance (C_y, D_y) of Set-Int over the universe $\mathcal{X}$ where $C_y \cap D_y = \{t_y\}$ for $t_y \in \mathcal{X}$.

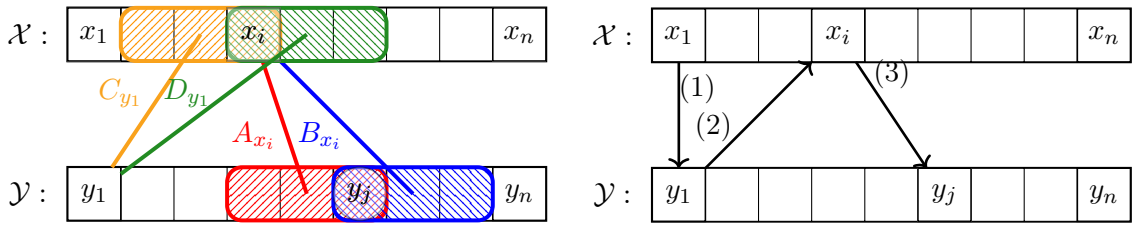
3. We define two mappings $f_{AB} : \mathcal{X} \to \mathcal{Y}$ and $f_{CD} : \mathcal{Y} \to \mathcal{X}$ such that:

   (a) for any $x \in \mathcal{X}$, $f_{AB}(x) = t_x \in \mathcal{Y}$ in the instance (A_x, B_x) of Set-Int.

   (b) for any $y \in \mathcal{Y}$, $f_{CD}(y) = t_y \in \mathcal{X}$ in the instance (C_y, D_y) of Set-Int.

4. Let $x_1 \in \mathcal{X}$ be an arbitrary fixed element of $\mathcal{X}$ known to all players. The pointers $z_0, z_1, z_2, z_3, \ldots$ are defined inductively as follows: $z_0 := x_1, z_1 := f_{AB}(z_0), z_2 := f_{CD}(z_1), z_3 := f_{AB}(z_2), \cdots$.

The *k-step hidden-pointer chasing* problem (HPC_k) is defined as the communication problem of finding the pointer z_k. See Figure 6.1 for an illustration.



(a) The sets $A_{x_i}, B_{x_i} \subseteq \mathcal{Y}$ of P_A and P_B for $x_i \in \mathcal{X}$, and $C_{y_1}, D_{y_1} \subseteq \mathcal{X}$ of P_C and P_D for $y_1 \in \mathcal{Y}$.

(b) $z_0 = x_1$, $z_1 = y_1$, $z_2 = x_i$, $z_3 = y_j$, implying that the answer to HPC_3 in this example is y_j.

Figure 6.1: Illustration of the HPC problem.

We define a *phase* (similar to a round) for protocols that solve HPC. In an odd (resp. even) phase, only P_C and P_D (resp. P_A and P_B) are allowed to communicate with each other, and the phase ends once a message is sent to P_A or P_B (resp. P_C or P_D). A protocol is called a *k-phase* protocol iff it uses at most k phases.

6.1. Main Results

It is easy to see that in $k+1$ phases, we can compute HPC_k with $O(k \cdot n)$ total communication by solving the $\mathsf{Set\text{-}Int}$ instances corresponding to $z_0, z_1, \ldots, z_k$ one at a time in each phase. We prove that if we only have k phases however, solving HPC_k requires a large communication.

Theorem 6.1 (Informal)**.** *Any k-phase protocol that outputs the correct solution to* HPC_k *with constant probability requires* $\Omega(n^2/k^2 + n)$ *bits of communication.*

Theorem 6.1 implies a new approach towards proving graph streaming lower bounds that sits squarely in the middle of previous methods: HPC is a problem that admits an "efficient" protocol when there is no limit on rounds of communication and yet is "hard" with even a

polynomial limitation on number of rounds. We use this result to prove strong pass lower bounds for some fundamental problems in graph streams via reductions from HPC.

Cut and Flow Problems. One of the main applications of Theorem 6.1 is the following result.

Theorem 6.2. *Any p-pass streaming algorithm that with a constant probability outputs the minimum s-t cut value in a weighted graph (undirected or directed) requires $\Omega(n^2/p^5)$ space.*

Prior to our work, the best lower bound known for this problem was an $n^{1+\Omega(1/p)}$ space lower bound for p-pass algorithms [141] (for weighted undirected graphs and unweighted directed graphs). Theorem 6.2 significantly improves upon this. In particular, it implies that $\widetilde{\Omega}(n^{1/5})$ passes are necessary for semi-streaming algorithms, exponentially improving upon the $\Omega(\frac{\log n}{\log\log n})$ lower bound of [141]. At the same time, Theorem 6.2 also shows that any streaming algorithm for this problem with a small number of passes, namely polylog(n) passes, requires $\widetilde{\Omega}(n^2)$ space, almost the same space as the trivial single-pass algorithm that stores the input graph entirely.

Our Theorem 6.2 should be contrasted with the results of [234] that imply an $\widetilde{O}(n^{5/3})$ space algorithm for unweighted minimum s-t cut on undirected graphs in only *two* passes (see Footnote 1).

By max-flow min-cut theorem, Theorem 6.2 also implies identical bounds for computing the value of maximum s-t flow in capacitated graphs, making progress on a question raised in [202] regarding the streaming complexity of maximum flow in directed graphs.

Lexicographically-First Maximal Independent Set. A maximal independent set (MIS) returned by the sequential greedy algorithm that visits the vertices of the graph in their lexicographical order is called the lexicographically-first MIS. We prove the following result for this problem.

Theorem 6.3. *Any p-pass streaming algorithm that with constant probability finds a lexicographically first maximal independent set of in a graph requires $\Omega(n^2/p^5)$ space.*

The lexicographically-first MIS has a rich history in computer science and in particular parallel algorithms [9, 58, 93, 200]. However, even though multiple variants of the independent set problem have been studied in the streaming model [18, 94, 95, 126, 142, 143, 145], we are not aware of any work on this particular problem (we remark that standard MIS problem admits an $\widetilde{O}(n)$ space $O(\log \log n)$ pass algorithm [126]). Besides being a fundamental problem in its own right, what makes this problem appealing for us is that it nicely illustrates the power of our techniques compared to previous approaches. The lexicographically-first MIS can be computed with $O(n)$ communication in the two-player communication model (or for any constant number of players) with no restriction on number of rounds by a direct simulation of the sequential algorithm. Hence, this problem perfectly fits the class of problems for which previous techniques cannot prove lower bounds beyond logarithmic passes. To our knowledge, this is the first super-logarithmic pass lower bound for any graph problem that admits an efficient protocol with no restriction on number of rounds.

6.1.1. Our Techniques

Our reductions take a different path than previous pointer chasing based reductions that used edges of the graph to directly encode pointers. In particular, our hidden-pointer chasing problem allows us encode a single pointer among $\Theta(n)$ edges and thus work with graphs with density $\Omega(n^2)$ and still keep a polynomial dependence on number of rounds in the communication lower bound. This results in space lower bounds of the form $n^2/p^{O(1)}$ for p-pass streaming algorithms.

The main technical contribution is the communication complexity lower bound for HPC in Theorem 6.1. This result is proved by combining inductive arguments for round/communication tradeoffs (see, e.g. [219, 254]) with direct-sum arguments for information complexity (see, e.g. [30, 32, 64, 67]) to account for the role of set intersection inside HPC. To make this argument work, we also need to prove a stronger lower bound for set intersection than currently known results (see, e.g. [68]). In particular, we prove that any protocol that can even slightly reduce the "uncertainty" about the intersecting element must have a "large"

communication and information complexity.

Our new lower bound for set intersection is also proved using tools from information complexity to reduce this problem to a primitive problem, namely set intersection itself on a universe of size two. This requires a novel argument to handle the protocols for set intersection that reduce the uncertainty about the intersecting element without necessarily making much "progress" on finding this element. Another challenge is that unlike typical direct-sum results in this context, say reducing disjointness to the AND problem; see, e.g. [30, 65, 66, 247], set intersection cannot be decomposed into *independent* instances of the primitive problem (this is similar-in-spirit to challenges in analyzing information complexity of set disjointness on *intersecting* distributions [82, 161] as opposed to (more standard) non-intersecting ones). Finally, we prove a lower bound for the primitive problem using the product structure of Hellinger distance for communication protocols (see, e.g. [30, 247]).

6.2. Proof Sketch of Theorem 6.1

In this section, we sketch the proof of Theorem 6.1 which is the main technical result of this chapter. Let $\mathsf{dist}_{\mathsf{SI}}$ be a hard distribution on instances (A, B) for Set-Int. In this distribution A and B are each sets of size almost $n/3$ such that they intersect in a unique element in the universe chosen uniformly at random. We define the distribution $\mathsf{dist}_{\mathsf{HPC}}$ over inputs of HPC as the distribution in which all instances (A_x, B_x) and (C_y, D_y) for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ are sampled independently from $\mathsf{dist}_{\mathsf{SI}}$ (note that $\mathsf{dist}_{\mathsf{HPC}}$ is not a product distribution as $\mathsf{dist}_{\mathsf{SI}}$ is not a product distribution).

Fix any k-phase deterministic protocol π_{HPC} for HPC_k throughout this section and suppose towards a contradiction that $\mathsf{CC}(\pi_{\mathsf{HPC}}) = o(n^2/k^2)$ (the lower bound extends to randomized protocols by Yao's minimax principle [252]). For any $j \in [k]$, we define Π_j as the set of all messages communicated by π_{HPC} in phase j and $\Pi := (\Pi_1, \ldots, \Pi_k)$ as the transcript of the protocol π_{HPC}. We further define $Z = (z_1, \ldots, z_k)$, $E_j := (\Pi^{<j}, Z^{<j})$ for any $j > 1$, and $E_1 = z_0$. We think of E_j as the information "easily known" to players at the beginning of phase j. The main step of the proof of Theorem 6.1 is the following key lemma which we

prove inductively.

Lemma 6.2.1 (Informal)**.** *For all $j \in [k]$:* $\mathbb{E}_{(E_j,\Pi_j)}\left[\|\mathrm{dist}(\mathsf{Z}_j \mid E_j,\Pi_j) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd}\right] = o(1)$.

Lemma 6.2.1 states that if the communication cost of a protocol is "small", i.e., is $o(n^2/k^2)$, then even after communicating the messages in the first j phases of the protocol, distribution of z_j is still "close" to being uniform. This in particular implies that at the end of the protocol, i.e., at the end of phase k, the target pointer z_k is essentially distributed as in its original distribution (which is uniform over $\mathcal{Y}$ or $\mathcal{X}$ depending on whether k is odd or even). Hence π_{HPC} should not be able to find z_k at the end of phase k. The proof of Theorem 6.1 follows easily from this intuition.

Proof Sketch of Lemma 6.2.1. The first step of proof is to show that finding the target element of a *uniformly at random* chosen instance of Set-Int (as opposed to an instance corresponding to any particular pointer) in HPC is not possible with low communication. For any $x \in \mathcal{X}$ and any $y \in \mathcal{Y}$, define the random variables $\mathsf{T}_x \in \mathcal{Y}$ and $\mathsf{T}_y \in \mathcal{X}$, which correspond to the target elements of Set-Int on (A_x, B_x) and (C_y, D_y), respectively. The following lemma formalizes the above statement. For simplicity, we only state it for T_x for $x \sim \mathcal{U}_{\mathcal{X}}$; an identical bound also hold for T_y for $y \sim \mathcal{U}_{\mathcal{Y}}$.

Lemma 6.2.2 (Informal)**.** *For $j \in [k]$:* $\mathbb{E}_{(E_j,\Pi_j)}\,\mathbb{E}_{x\sim\mathcal{U}_{\mathcal{X}}}\left[\|\mathrm{dist}(\mathsf{T}_x \mid E_j,\Pi_j) - \mathrm{dist}(\mathsf{T}_x)\|_{tvd}\right] = o(1)$.

Let us first see why Lemma 6.2.2 implies Lemma 6.2.1. The proof is by induction. Consider some phase $j \in [k]$ and suppose j is odd by symmetry. The goal is to prove that distribution of Z_j conditioned on $(E_j,\Pi_j) = (z_1,\ldots,z_{j-1},\Pi_1,\ldots,\Pi_{j-1},\Pi_j)$ is close to original distribution of Z_j (on average over choices of (E_j,Π_j)). Notice that since we assumed j is odd, Z_j is a function of the inputs to P_A and P_B. On the other hand, in an odd phase, only the players P_C and P_D communicate and hence Π_j is a function of the inputs to these players. Conditioning on E_j and using the rectangle property of deterministic protocols (see Fact 2.5.5), together with the fact that inputs to P_A, P_B are independent of inputs to

P_C, P_D, implies that $\mathsf{Z}_j \perp \Pi_j \mid E_j$. We now have:

(i) Conditioned on z_{j-1}, Z_j is the target element of the instance $(A_{z_{j-1}}, B_{z_{j-1}})$, i.e., $\mathsf{Z}_j = \mathsf{T}_{z_{j-1}}$.

(ii) z_{j-1} itself is distributed according to $\mathrm{dist}(\mathsf{Z}_{j-1} \mid E_{j-1}, \Pi_{j-1})$ (because we removed the conditioning on Π_j by the above argument).

(iii) $\mathrm{dist}(\mathsf{Z}_{j-1} \mid E_{j-1}, \Pi_{j-1})$ is close to the uniform distribution by induction.

As such we can now simply apply Lemma 6.2.2 (by replacing x with z_{j-1} since they essentially have the same distribution) and obtain that distribution of $\mathsf{Z}_j = \mathsf{T}_{z_{j-1}}$ with and without conditioning on (E_j, Π_j) is almost the same (averaged over choices of (E_j, Π_j)), proving the lemma.

Proof Sketch of Lemma 6.2.2 The proof of this lemma is based on a direct-sum style argument combined with a new result that we prove for Set-Int. The direct-sum argument implies that since x is chosen uniformly at random from n elements in $\mathcal{X}$, and protocol π_{HPC} is communicating $o(n^2)$ bits in total, then it can only reveal $o(n)$ bits of information about the instance (A_x, B_x). This part follows the standard direct-sum arguments for information complexity (see, e.g. [32, 67]) but we also need to take into account that if x is one of the pointers we conditioned on in E_j, then π_{HPC} may reveal more information about (A_x, B_x); fortunately, this event happens with negligible probability for $k \ll n$ and so the argument continues to hold.

By above argument, proving Lemma 6.2.2 reduces to showing that if a protocol reveals $o(n)$ bits of information about an instance of Set-Int, then the distribution of the target element varies from the uniform distribution in total variation distance by only $o(1)$. This is the main part of the proof of Lemma 6.2.2 and is precisely the content of our next technical result in the following section.

6.2.1. A New Communication Lower Bound for Set Intersection

We say that a protocol π_{SI} *ε-solves* Set-Int on the distribution $\text{dist}_{\mathsf{SI}}$ iff it can alter the distribution of the target element from its original distribution by at least ε in total variation distance, i.e., $\mathbb{E}_{\Pi_{\mathsf{SI}} \sim \Pi_{\mathsf{SI}}} \left[\|\text{dist}(\mathsf{T} \mid \Pi_{\mathsf{SI}}) - \text{dist}(\mathsf{T})\|_{tvd} \right] \geq \varepsilon$; here Π_{SI} and T are the random variables for the transcript of the protocol (including public randomness) and the target element, respectively.

To finish the proof of Lemma 6.2.2, we need to prove that a protocol that $\Omega(1)$-solves Set-Int has $\Omega(n)$ communication cost (even information cost). Note that ε-solving is an algorithmically simpler task than finding the target element. For example, a protocol may change the distribution of T to having $(1+\varepsilon)/n$ probability on $n/2$ elements and $(1-\varepsilon)/n$ probability on the remaining $n/2$. This ε-solves Set-Int yet the target element can only be found with probability $(1+\varepsilon)/n$ in this distribution. On the other hand, any protocol that finds the target element with probability $p \in (0,1)$ also p-solves Set-Int. Because of this, the lower bounds mentioned before for set intersection do not suffice for our purpose. Instead, we prove the following theorem in this paper.

Theorem 6.4 (Informal)**.** *Any protocol π_{SI} that ε-solves* Set-Int *on distribution $\text{dist}_{\mathsf{SI}}$ has internal information cost $\mathsf{IC}_{\text{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}}) = \Omega(\varepsilon^2 \cdot n)$.*

As information cost lower bounds communication cost (see Proposition 2.5.4), Theorem 6.4 also proves a communication lower bound for Set-Int (although we need the stronger result for information cost in our proofs). By our discussion earlier, Theorem 6.4 can be used to finalize the proof of Lemma 6.2.2 (and hence Theorem 6.1). We now give an overview of the proof of Theorem 6.4.

For an instance (A, B) of Set-Int, with a slight abuse of notation, we write $A := (a_1, \ldots, a_n)$ and $B := (b_1, \ldots, b_n)$ for $a_i, b_i \in \{0,1\}$ as characteristic vector of the sets given to Alice and Bob. Under this notation, the target element corresponds to the unique index $t \in [n]$ such that $(a_t, b_t) = (1,1)$. The proof of Theorem 6.4 is based on reducing Set-Int to a special case

of this problem on only 2 coordinates, which we define as the Pair-Int problem. In Pair-Int, Alice and Bob are given (x_1, x_2) and (y_1, y_2) in $\{0,1\}^2$ and their goal is to find the unique index $k \in \{1,2\}$ such that $(x_k, y_k) = (1,1)$. We use $\mathsf{dist}_{\mathsf{PI}}$ to denote the hard distribution for this problem which is equivalent to $\mathsf{dist}_{\mathsf{SI}}$ for $n = 2$.

Given a protocol π_{SI} for ε-solving Set-Int on $\mathsf{dist}_{\mathsf{SI}}$, we design a protocol π_{PI} for finding the index k in instances of Pair-Int sampled from $\mathsf{dist}_{\mathsf{PI}}$ with probability $1/2 + \Omega(\varepsilon)$. The reduction is as follows.

Reduction: Alice and Bob publicly sample $i, j \in [n]$ uniformly at random without replacement. Then, Alice sets $a_i = x_1$ and $a_j = x_2$ and Bob sets $b_i = y_1$ and $b_j = y_2$, using their given inputs in Pair-Int. The players sample the remaining coordinates of (A, B) in $[n] \setminus \{i, j\}$ using a combination of public and private randomness that we explain later in the proof sketch of Lemma 6.2.4. This sampling ensures that the resulting instance (A, B) of Set-Int is sampled from $\mathsf{dist}_{\mathsf{SI}}$ such that its target element is i when $k = 1$ and is j when $k = 2$. After this, the players run the protocol π_{SI} on (A, B) and let Π_{SI} be the transcript of this protocol. Using this, Bob computes the distribution $\mathsf{dist}(\mathsf{T} \mid \Pi_{\mathsf{SI}}) = (p_1, \ldots, p_n)$ which assigns probabilities to elements in $[n]$ as being the target element. Finally, Bob checks the value of p_i and p_j and return $k = 1$ if $p_i > p_j$ and $k = 2$ otherwise (breaking the ties consistently when $p_i = p_j$). The remainder of the proof consists of three main steps:

(i) Proving the correctness of protocol π_{PI}:

Lemma 6.2.3 (Informal)**.** *Protocol π_{PI} outputs the correct answer with probability $\frac{1}{2} + \Omega(\varepsilon)$.*

(ii) Proving an upper bound on "information cost" of π_{PI} (the reason for quotations is that strictly speaking this quantity is not the information cost of π_{PI} but rather a lower bound for it).

Lemma 6.2.4 (Informal)**.** *Let Π_{PI} denote the random variable for the transcript of the protocol π_{PI} and K be the random variable for the index k in distribution $\mathsf{dist}_{\mathsf{PI}}$.*

We have,

$$\mathbb{I}_{\mathsf{dist}_{\mathsf{PI}}}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}) + \mathbb{I}_{\mathsf{dist}_{\mathsf{PI}}}(\mathsf{Y}_1, \mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_1, \mathsf{X}_2, \mathsf{K}) \leq \frac{1}{n-1} \cdot \mathsf{IC}_{\mathsf{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}}).$$

(*iii*) Proving a lower bound on "information cost" (as used in Part (*ii*)) of protocols for Pair-Int:

Lemma 6.2.5. *If π_{PI} outputs the correct answer on $\mathrm{dist}_{\mathsf{PI}}$ with probability at least $\frac{1}{2} + \Omega(\varepsilon)$, then,*

$$\mathbb{I}_{\mathsf{dist}_{\mathsf{PI}}}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}) + \mathbb{I}_{\mathsf{dist}_{\mathsf{PI}}}(\mathsf{Y}_1, \mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_1, \mathsf{X}_2, \mathsf{K}) = \Omega(\varepsilon^2).$$

By Lemma 6.2.4, $\mathsf{IC}_{\mathsf{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}})$ is $\Omega(n)$ times larger than LHS of Lemma 6.2.5, and this, combined with Lemma 6.2.3, implies that information cost of π_{SI} needs to be $\Omega(\varepsilon^2) \cdot \Omega(n)$, proving Theorem 6.4.

Proof Sketch of Lemma 6.2.3. Let us again consider a protocol π_{SI} such that dist($\mathsf{T} \mid \Pi_{\mathsf{SI}}$) is putting $(1+\varepsilon)/n$ mass over $n/2$ elements and $(1-\varepsilon)/n$ mass on the remaining ones. Suppose that the correct answer to the instance of Pair-Int is index 1. We know that in this case, the index i chosen by π_{PI} will be the target index t in the instance (A, B). A key observation here is that the index j however can be any of the coordinates in instance (A, B) other than the target element with the same probability. As such, parameters p_i and p_j used to decide the answer in π_{PI} are distributed as follows: p_i is sampled from dist($\mathsf{T} \mid \Pi_{\mathsf{SI}}$) and hence has value $(1+\varepsilon)/n$ with probability $(1+\varepsilon)/2$ and $(1-\varepsilon)/n$ with probability $(1-\varepsilon)/2$. On the other hand, p_j is chosen uniformly at random from $(p_1, \ldots, p_n)$ and hence is $(1+\varepsilon)/n$ or $(1-\varepsilon)/n$ with the same probability of half. Thus $p_i > p_j$ with probability $1/2 + \Omega(\varepsilon)$ and hence π_{PI} has $\Omega(\varepsilon)$ advantage over random guessing.

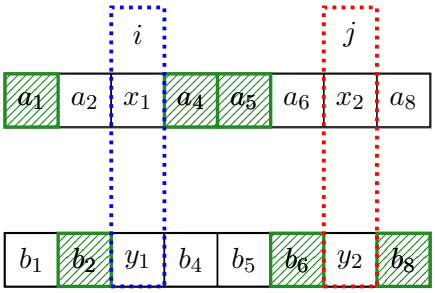
The proof of Lemma 6.2.3 then formalizes the observations above and extend this argument to any protocol π_{SI} that ε-solves Set-Int no matter how it alters the distribution of the target

element.

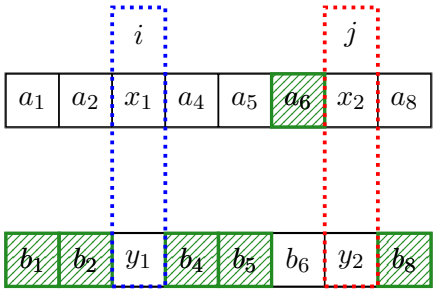
Proof Sketch of Lemma 6.2.4. We first note that the LHS in Lemma 6.2.4 is *not* the internal information cost of π_{PI} due to further conditioning on K (this term can only be smaller than $\mathsf{IC}_{\mathsf{dist}_{\mathsf{PI}}}(\pi_{\mathsf{PI}})$). Hence, Lemma 6.2.4 is proving a "weaker" statement than a direct-sum result for information cost of π_{PI} based on π_{SI}. The reason for settling for this weaker statement has to do with the fact that the coordinates in distribution $\mathsf{dist}_{\mathsf{SI}}$ are *not* chosen independently (see Section 6.4.1 for more detail).

The intuition behind the proof is as follows. The LHS in Lemma 6.2.5 is the information revealed about the input of players (in $\mathsf{Pair\text{-}Int}$) averaged over choices of $k = 1$ and $k = 2$. Let us assume $k = 1$ by symmetry. In this case, this quantity is simply the information revealed about (x_2, y_2) by the protocol as $(x_1, y_1) = (1, 1)$ and hence has no entropy. However, when $k = 1$, (x_2, y_2) is embedded in index j, i.e., $(x_2, y_2) = (a_j, b_j)$ and has the same distribution as all other coordinates in A_{-i}, B_{-i}. As such, since the protocol π_{SI} called inside π_{PI} is oblivious to the choice of j, the information revealed about (a_j, b_j) in average is smaller than the information revealed by π_{SI} about A_{-i}, B_{-i} (which itself is at most the information cost of π_{SI}) by a factor of $n - 1$.

This outline oversimplifies many details. One such detail is the way of ensuring a "symmetric treatment" of both indices i and j. This is crucial for the above argument to work for both $k = 1$ and $k = 2$ cases simultaneously, without the players knowing which index the "averaging" of information is being done for (index j in the context of the discussion above). The key step in making this information-theoretic argument work is the following public-private sampling: Alice and Bob use public randomness to pick an integer $\ell \in [n-2]$ uniformly at random and then pick a set S of size ℓ uniformly at random from $[n] \setminus \{i, j\}$. Next, the players sample $a_{i'}$ and $b_{j'}$ for $i' \in S$ and $j' \in ([n] \setminus \{i, j\}) \setminus S$ from $\mathsf{dist}_{\mathsf{SI}}$ again using public randomness. Finally, each player samples the remaining coordinates in the input using private randomness from $\mathsf{dist}_{\mathsf{SI}}$. Figure 6.2 gives an example.



(a) An example with $\ell = 3$ and $S = \{1, 4, 5\}$:
$\{a_1, a_4, a_5, b_2, b_6, b_8\}$ is sampled publicly.
$\{a_2, a_6, a_8\}$ and $\{b_1, b_4, b_5\}$ are sampled privately.



(b) An example with $\ell = 1$ and $S = \{6\}$:
$\{a_6, b_1, b_2, b_4, b_5, b_8\}$ is sampled publicly.
$\{a_1, a_2, a_4, a_5, a_8\}$ and $\{b_6\}$ are sampled privately.

Figure 6.2: Illustration of the process of sampling of instances of Set-Int in π_{PI} for $n = 8$. In these examples, $i = 3$ and $j = 7$ and hence $(a_3, a_7) = (x_1, x_2)$ and $(b_3, b_7) = (y_1, y_2)$. of ℓ and S.

Proof Sketch of Lemma 6.2.5. Let $\Pi_{[x_1x_2,\, y_1y_2]}$ denote the transcript of the protocol condition on the inputs (x_1, x_2) and (y_1, y_2) to Alice and Bob. Suppose towards a contradiction that the LHS of Lemma 6.2.5 is $o(\varepsilon^2)$. By focusing on the conditional terms when $k = 1$, we can show that distribution of $\Pi_{[1x_2',\, 1y_2']}$ and $\Pi_{[1x_2'',\, 1y_2'']}$ for all choices of (x_2', y_2') and (x_2'', y_2'') in the support of $\mathrm{dist}_{\mathsf{PI}}$ are quite close. This is intuitively because the information revealed about (x_2, y_2) by π_{PI} conditioned on $k = 1$ is small (the same result holds for $\Pi_{[x_2'1,\, y_2'1]}$ and $\Pi_{[x_2''1,\, y_2''1]}$ by $k = 2$ terms).

Up until this point, there is no contradiction as the answer to inputs $(1, *)$,$(1, *)$ to Alice and Bob is always 1 and hence there is no problem with the corresponding transcripts in $\Pi_{[1*,\, 1*]}$ to be similar (similarly for $\Pi_{[*1,\, *1]}$ separately). However, we combine this with the cut-and-paste property of randomized protocols based on Hellinger distance (see Fact 2.5.6) to argue that in fact the distribution of $\Pi_{[10,\, 10]}$ and $\Pi_{[01,\, 01]}$ are also similar. This then implies that $\Pi_{[1*,\, 1*]}$ essentially has the same distribution as $\Pi_{[*1,\, *1]}$; but then this is a contradiction as the answer to the protocol (which is only a function of the transcript) needs to be different between these two types of inputs.

6.3. The Set Intersection Problem

Starting from this section, we delve into the formal proofs of our results. This section contains our new lower bound for the set intersection problem (stated informally in Theorem 6.4). Recall that Set-Int is a two-player communication problem in which Alice and Bob are given sets A and B from $[n]$, respectively, with the promise that there exists a unique element t such that $\{t\} = A \cap B$. The goal is for Alice and Bob to find t, referred to as the *target element*. It is sometimes more convenient to consider the characteristic vector of sets A and B rather than the sets directly. Hence, with a slight abuse of notation, we write $A := (a_1, \ldots, a_n) \in \{0,1\}^n$ and $B := (b_1, \ldots, b_n) \in \{0,1\}^n$ where $a_i = 1$ (resp. $b_i = 1$) iff the element i belongs to the set A (resp. to B). In this notation, the target element t corresponds to the *unique* index where $(a_t, b_t) = (1,1)$.

The Set-Int problem is closely related to the well-known *set disjointness* problem. It is in fact straightforward to prove an $\Omega(n)$ lower bound on the communication complexity of Set-Int using a simple reduction from the set disjointness problem. However, in this paper, we are interested in an algorithmically simpler variant of this problem which we define below.

6.3.1. Problem Statement

Consider the following distribution $\text{dist}_{\mathsf{SI}}$ for Set-Int.

Distribution $\text{dist}_{\mathsf{SI}}$ on sets (A, B) from the universe $[n]$:

1. Define μ as the uniform distribution over the set $\{(0,0), (0,1), (1,0)\}$.
2. For $i \in [n]$, choose (a_i, b_i) independently from distribution μ.
3. Sample an element $t \in [n]$ uniformly at random and change $(a_t, b_t) = (1,1)$.

Rather than finding the target element t, we are only interested in slightly reducing the "uncertainty" about its identity as formalized below.

Definition 6.3.1. *We say that a protocol π_{SI} **ε-solves** the Set-Int problem on the distribution*

$\mathrm{dist}_{\mathsf{SI}}$ *iff*

$$\mathop{\mathbb{E}}_{\Pi_{\mathsf{SI}} \sim \Pi_{\mathsf{SI}}} \left[\|\mathrm{dist}(\mathsf{T} \mid \Pi_{\mathsf{SI}}) - \mathcal{U}_{[n]}\|_{tvd} \right] \geq \varepsilon, \tag{6.1}$$

where T *is the random variable for the target element and* $\mathcal{U}_{[n]}$ *is the uniform distribution on* $[n]$*.*

Let us first consider two "extreme examples" of a protocol that ε-solves Set-Int and see how much communication is needed to realize each one.

Example 6.1. *One way of ensuring Eq (6.1) is to have protocols that after communication can rule out* $\Theta(\varepsilon \cdot n)$ *elements as candidates for* t *and leave the target element to be uniformly distributed on the remaining* $n - \Theta(\varepsilon \cdot n)$ *elements.*

Intuitively, such a protocol should require a large communication as it is making a significant "progress" towards finding the target element. Indeed, if the communication cost of this protocol is small, we can run this protocol again on the remaining candidates and shrink their number further, and continue doing this until we find the target element t*, without making a large communication. This contradicts the* $\Omega(n)$ *communication lower bound for finding the element* t *exactly.*

Example 6.2. *Another way of satisfying Eq (6.1) is to have protocols that simply change the probability mass of the target element* t *on half of the elements from* $1/n$ *to* $(1+\varepsilon)/n$*, and on the remaining half from* $1/n$ *to* $(1-\varepsilon)/n$*.*

Analyzing the communication cost of such protocols is distinctly more delicate. On the surface, it does not seem that the protocol has made much "progress" towards finding the target element t *as nearly all elements are still quite likely candidates for being the target. Hence, to show such protocols require large communication, we now need to go beyond reducing this problem to finding the target element* t *exactly. Roughly speaking, we show that to be able to make such a change in distribution of* t*, the protocol needs to communicate non-trivial*

information for every potential element, hence requiring a large communication again.

In the following, we show that no matter how a protocol decides to change the variation distance of t from its original distribution, it needs a large communication. However, we also encourage the reader to consider our arguments in the context of the above two examples for concreteness.

6.3.2. Communication Complexity of ε-solving Set-Int

We prove the following lower bound on the information cost of protocols for ε-solving Set-Int.

Theorem 6.5. *Suppose π_{SI} is a protocol for* Set-Int *on instances (A, B) sampled from $\mathrm{dist}_{\mathsf{SI}}$. Let Π_{SI} denote the transcript of the protocol π_{SI}. If $\mathbb{E}_{\Pi_{\mathsf{SI}} \sim \Pi_{\mathsf{SI}}} \left[\| \mathrm{dist}(\mathsf{T} \mid \Pi_{\mathsf{SI}}) - \mathcal{U}_{[n]} \|_{tvd} \right] \geq \varepsilon$, i.e., π_{SI} ε-solves* Set-Int*, then the internal information cost of π_{SI} on $\mathrm{dist}_{\mathsf{SI}}$ is $\mathsf{IC}_{\mathrm{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}}) = \Omega(\varepsilon^2 \cdot n)$.*

We shall remark that for our purpose, we crucially use the fact that the lower bound in Theorem 6.5 is for the internal information cost and for the distribution $\mathrm{dist}_{\mathsf{SI}}$. However, as information cost lower bounds communication cost by Proposition 2.5.4, this immediately implies that communication complexity of Set-Int is also large, which is of independent interest.

Corollary 6.3.2. *Any protocol π_{SI} for ε-solving* Set-Int *on distribution $\mathrm{dist}_{\mathsf{SI}}$ needs to communicate $\Omega(\varepsilon^2 \cdot n)$ bits of communication, i.e., $\mathsf{CC}_{\mathrm{dist}}(\pi) = \Omega(\varepsilon^2 \cdot n)$.*

One standard approach to proving the lower bound in Theorem 6.5 is to reduce the Set-Int problem—via a direct-sum type argument—to *many* instances of a *simpler* problem, and then prove the lower bound for the simpler problem directly. To do so, we reduce Set-Int to the same problem on only two coordinates, which we refer to as the *pair intersection* problem, denoted by Pair-Int. In Pair-Int, Alice and Bob are given tuples $(x_1, x_2) \in \{0,1\}^2$ and $(y_1, y_2) \in \{0,1\}^2$, respectively (we also use the concise notation $[x_1x_2, y_1y_2]$ to denote the joint inputs to the players), with the promise that there exists a unique index $k \in \{1, 2\}$

such that $(x_k, y_k) = (1, 1)$. The goal is to output the index k. Note that this problem is equivalent to Set-Int when $n = 2$ modulo the fact that here we actually care about finding k as opposed to ε-solving (to avoid ambiguity, we use k to denote the target element for Pair-Int and t for Set-Int). Consider the following distribution which is equivalent to $\text{dist}_{\mathsf{SI}}$ for $n = 2$.

Distribution $\text{dist}_{\mathsf{PI}}$ on tuples (x_1, x_2) and (y_1, y_2) from $\{0, 1\}^2$.

1. For $i \in \{1, 2\}$, choose (x_i, y_i) uniformly at random from distribution μ (defined in $\text{dist}_{\mathsf{SI}}$).
2. Pick $k \in \{1, 2\}$ uniformly at random and change (x_k, y_k) to $(1, 1)$.

We prove that any protocol that ε-solves Set-Int on $\text{dist}_{\mathsf{SI}}$ with internal information cost $o(\varepsilon^2 \cdot n)$ bits can be used to obtain a protocol for Pair-Int that only reveals $o(\varepsilon^2)$ bits of information about the input (with respect to distribution $\text{dist}_{\mathsf{PI}}$) but is able to solve this problem with probability at least $1/2 + \varepsilon$ on distribution $\text{dist}_{\mathsf{PI}}$. We then prove that such a protocol cannot exist for Pair-Int. We should note that the notion of information revealed for Pair-Int that we use is rather non-standard (it neither corresponds to internal information cost nor to external information cost that are typically studied). We elaborate more on this later in Lemma 6.3.6.

Proof of Theorem 6.5

In the following, let π_{SI} be any protocol for Set-Int that satisfies Eq (6.1), i.e., ε-solves Set-Int on $\text{dist}_{\mathsf{SI}}$. We use this protocol to obtain a protocol π_{PI} for Pair-Int.

Protocol π_{PI}: The protocol for Pair-Int using a protocol π_{SI} for Set-Int.

Input: An instance $[x_1x_2, y_1y_2] \sim \text{dist}_{\mathsf{PI}}$.

Output: $k \in \{1, 2\}$ as the answer to Pair-Int.

1. **Sampling the instance.** The players create an instance (A, B) of Set-Int as follows

(see Figure 6.2 on page 199 for an illustration):

(a) Using public coins, Alice and Bob sample $i, j \in [n]$ uniformly without replacement.

(b) Alice sets $a_i = x_1$ and $a_j = x_2$ and Bob sets $b_i = y_1$ and $b_j = y_2$, using their given inputs in Pair-Int.

(c) Using public coins, Alice and Bob sample $\ell \in \{0, 1, \ldots, n-2\}$ uniformly at random and then pick an ℓ-subset S of $[n] \setminus \{i, j\}$ uniformly at random. Let $\overline{S} := ([n] \setminus \{i, j\}) \setminus S$.

(d) Using public coins, Alice and Bob sample $A_S, B_{\overline{S}}$ independently from distribution μ (defined in $\text{dist}_{\mathsf{SI}}$).

(e) Using private coins, Alice samples the remaining coordinates in $A_{\overline{S}}$ so that joint distribution of each coordinate is μ. Similarly, Bob samples the coordinates in B_S.

2. **Computing the answer.** Alice and Bob run the protocol π_{SI} on (A, B) and let Π_{SI} be the transcript of the protocol. They compute the answer to Pair-Int as follows:

(a) The players compute the distribution $\text{dist}(\mathsf{T} \mid \Pi_{\mathsf{SI}}) = (p_1, \ldots, p_n)$ where T denotes the random variable for the target element of Set-Int.

(b) Fix a total ordering $\succ_{\Pi_{\mathsf{SI}}}$ on $[n]$ such that for $x \neq y \in [n]$, $x \succ_{\Pi_{\mathsf{SI}}} y$ iff $p_x > p_y$ or $p_x = p_y$ and $x > y$. We use $x \prec_{\Pi_{\mathsf{SI}}} y$ to mean $y \succ_{\Pi_{\mathsf{SI}}} x$.

(c) Return 1 if $i \succ_{\Pi_{\mathsf{SI}}} j$ and 2 otherwise.

The following observations are in order. Firstly, we note that the rather peculiar way of sampling the instances (A, B) in π_{PI} via public and private randomness is only for the purpose of making the information-theoretic arguments needed to reduce Set-Int to Pair-Int work; for the purpose of correctness of the reduction, we only need the fact that these instances are sampled from $\text{dist}_{\mathsf{SI}}$ as captured by the following observation.

Observation 6.3.3. For an input $[x_1x_2, y_1y_2] \sim \text{dist}_{\mathsf{PI}}$, the distribution of the instances

(A, B) constructed in π_{PI} is $\mathrm{dist}_{\mathsf{SI}}$, where target $t = i$ when $x_1 \wedge y_1 = 1$ and target $t = j$ when $x_2 \wedge y_2 = 1$.

The following observation states a key property of the "non-target" index in $\mathrm{dist}_{\mathsf{PI}}$.

Observation 6.3.4. Conditioned on $x_1 \wedge y_1 = 0$ and any fixed choice of (A, B), the index i in π_{PI} is uniformly distributed on $[n] \setminus \{j\}$ (similarly for index j if $x_2 \wedge y_2 = 0$).

Proof. Conditioned on $x_1 \wedge y_1 = 0$, the distribution of (a_i, b_i) in (A, B) is μ, the same as all other indices except for j. □

The proof of Theorem 6.5 consists of three main steps: bounding the error probability of protocol π_{PI}, analyzing the information cost of π_{PI} in terms of information cost of π_{SI}, and proving a lower bound on the information cost of π_{PI} based on its error probability. Formally, in the first step we prove that:

Lemma 6.3.5 (Correctness of π_{PI})**.** *For instances sampled from* $\mathrm{dist}_{\mathsf{PI}}$*,* π_{PI} *outputs the correct answer with probability at least* $\frac{1}{2} + \Omega(\varepsilon)$ *(over the randomness of the distribution and the protocol).*

In the second step, we show that:

Lemma 6.3.6 (Information cost of π_{PI})**.** *Let* Π_{PI} *denote the random variable for the transcript of the protocol* π_{PI} *and* K *be the random variable for the index* k *in distribution* $\mathrm{dist}_{\mathsf{PI}}$*. We have,*

$$\mathbb{I}_{\mathrm{dist}_{\mathsf{PI}}}(\mathsf{X}_1, \mathsf{X}_2 \,;\, \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}) + \mathbb{I}_{\mathrm{dist}_{\mathsf{PI}}}(\mathsf{Y}_1, \mathsf{Y}_2 \,;\, \Pi_{\mathsf{PI}} \mid \mathsf{X}_1, \mathsf{X}_2, \mathsf{K}) \leq \frac{1}{n-1} \cdot \mathsf{IC}_{\mathrm{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}}).$$

The LHS in Lemma 6.3.6 is *not* the internal information cost of π_{PI} due to further conditioning on K. In fact, it is not hard to show that this quantity can only be smaller than the internal information cost of π_{PI}. Hence, Lemma 6.3.6 is proving a "weaker" statement

than a direct-sum result for internal information cost of π_{PI} based on π_{SI}. The reason for settling for this weaker statement has to do with the fact that the coordinates in distribution $\mathrm{dist}_{\mathsf{SI}}$ are *not* chosen independently and so the stronger bound does not seem to be true for our reduction[1]. Nevertheless, we show in the third part of the argument that this weaker statement suffices for our purpose.

In the final step of the proof, we prove that any protocol for Pair-Int that has a small error probability should have a large information cost with respect to the measure in Lemma 6.3.6.

Lemma 6.3.7 (Information complexity of Pair-Int)**.** *Suppose π_{PI} outputs the correct answer on $\mathrm{dist}_{\mathsf{PI}}$ with probability at least $\frac{1}{2} + \Omega(\varepsilon)$. Then,*

$$\mathbb{I}_{\mathrm{dist}_{\mathsf{PI}}}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}) + \mathbb{I}_{\mathrm{dist}_{\mathsf{PI}}}(\mathsf{Y}_1, \mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_1, \mathsf{X}_2, \mathsf{K}) = \Omega(\varepsilon^2).$$

We prove each of these three lemmas in the following sections. Before that, we show Theorem 6.5 follows easily from these lemmas.

Proof of Theorem 6.5 (assuming Lemma 6.3.5, Lemma 6.3.6, and Lemma 6.3.7). Suppose towards a contradiction that π_{SI} is a protocol that ε-solves Set-Int on $\mathrm{dist}_{\mathsf{SI}}$ and has information cost $\mathsf{IC}_{\mathrm{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}}) = o(\varepsilon^2 \cdot n)$. Create the protocol π_{PI} using π_{SI} as described in the reduction above. We have,

- By Lemma 6.3.5, π_{PI} outputs the correct answer on $\mathrm{dist}_{\mathsf{PI}}$ w.p. at least $\frac{1}{2} + \Omega(\varepsilon)$.
- By Lemma 6.3.6, $\mathbb{I}_{\mathrm{dist}_{\mathsf{PI}}}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}) + \mathbb{I}_{\mathrm{dist}_{\mathsf{PI}}}(\mathsf{Y}_1, \mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_1, \mathsf{X}_2, \mathsf{K}) = o(\varepsilon^2)$.

However, these two properties contradict Lemma 6.3.7. As such, the internal information cost of π_{SI} on $\mathrm{dist}_{\mathsf{SI}}$ should be $\Omega(\varepsilon^2 \cdot n)$, finalizing the proof. □

[1] Similar issues arise when analyzing information complexity of set disjointness on *intersecting* distributions [161] as opposed to the more standard case of non-intersecting distributions (e.g. [30, 65, 66, 247]).

Proof of Lemma 6.3.5: Correctness of Protocol π_{PI}

The following is a re-statement of Lemma 6.3.5 that we prove in this section.

Lemma (Restatement of Lemma 6.3.5)**.** *For an instance* $[x_1x_2, y_1y_2] \sim \text{dist}_{\mathsf{PI}}$*,* π_{PI} *outputs the correct answer with probability at least* $\frac{1}{2} + \Omega(\varepsilon)$ *(over the randomness of the distribution and the protocol).*

To give some intuition about this lemma, let us consider the Example 6.1 and Example 6.2. Suppose the correct answer to the instance of Pair-Int is index 1 and protocol π_{SI} that we use in reduction is of the type described in Example 6.1. We know that the set of $n - \Theta(\varepsilon \cdot n)$ elements computed by $\text{dist}_{\mathsf{SI}}$ definitely contains element i. What can be said about element j here? By Observation 6.3.4, the element j is chosen uniformly at random from all elements $[n] \setminus \{i\}$, *even* conditioned on a choice of A and B. As such, with probability $\Theta(\varepsilon)$, element j does not belong to the set of candidates for the target element computed by π_{SI}. In this case, protocol π_{PI} outputs the correct answer. This allows us to infer that π_{PI} is able to get $\Theta(\varepsilon)$ advantage over random guessing, exactly what is asserted by Lemma 6.3.5. A similar argument also works if protocol π_{SI} is of the type in Example 6.2. We now prove this lemma for general protocols.

Proof of Lemma 6.3.5. Assume $x_1 \wedge y_1 = 1$, i.e., index 1 is the correct answer to Pair-Int (the other case is symmetric). Let (A, B) be the instance of Set-Int constructed by π_{PI} and let Π_{SI} be the transcript of the protocol π_{SI} on (A, B) which is communicated inside π_{PI}. Recall that $\text{dist}(\mathsf{T} \mid \Pi_{\mathsf{SI}}) = (p_1, \ldots, p_n)$ is defined in π_{PI}. Also, define I and J as the random variables for indices i and j in π_{PI}. We claim,

$$\Pr\left(\pi_{\mathsf{PI}} \text{ errs} \mid x_1 \wedge y_1 = 1\right) = \mathop{\mathbb{E}}_{\Pi_{\mathsf{SI}} \sim \Pi_{\mathsf{SI}} \mid \mathsf{T}=\mathsf{I}} \left[\Pr\left(\mathsf{I} \prec_{\Pi_{\mathsf{SI}}} \mathsf{J} \mid \Pi_{\mathsf{SI}} = \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I}\right)\right]. \tag{6.2}$$

This is by construction of the protocol as $x_1 \wedge y_1 = 1$ and $\mathsf{T} = \mathsf{I}$ are equivalent, and conditioned on $x_1 \wedge y_1 = 1$, the correct answer is the index 1 which would be output by the protocol iff $i \succ_{\Pi_{\mathsf{SI}}} j$.

For any fixed transcript Π_{SI}, the bound in RHS of Eq (6.2) is only a function of the distribution of (I, J). Hence, let us examine $\text{dist}(\mathsf{I}, \mathsf{J} \mid \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I}) = \text{dist}(\mathsf{I} \mid \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I}) \cdot \text{dist}(\mathsf{J} \mid \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I} = i)$. For any $\ell \in [n]$, we have,

$$\Pr_{\text{dist}_{\mathsf{PI}}}(\mathsf{I} = \ell \mid \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I}) = \Pr_{\text{dist}_{\mathsf{SI}}}(\text{target element is } \ell \mid \Pi_{\mathsf{SI}}) = p_\ell. \tag{6.3}$$

This is simply by Observation 6.3.3 that implies instances created in π_{PI} are sampled from $\text{dist}_{\mathsf{SI}}$ and because we conditioned on $\mathsf{T} = \mathsf{I}$. On the other hand, conditioned on $\mathsf{T} = \mathsf{I} = i$, for any $\ell \in [n] \setminus \{i\}$,

$$\Pr_{\text{dist}_{\mathsf{PI}}}(\mathsf{J} = \ell \mid \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I} = i) = \Pr_{\text{dist}_{\mathsf{PI}}}(\mathsf{J} = \ell \mid \mathsf{T} = \mathsf{I} = i) = \frac{1}{n-1}. \tag{6.4}$$

This is by Observation 6.3.4 as Π_{SI} is only a function of (A, B), while J is independent of (A, B) (conditioned on $\mathsf{J} \neq \mathsf{T}$) and is uniform on any index which is not the target element.

Now that we have determined the distribution of (I, J) (conditioned on Π_{SI} and $\mathsf{T} = \mathsf{I}$), our goal is to simply bound the RHS of Eq (6.2) (for any fixed choice of Π_{SI}). Intuitively, we should expect this quantity to be small as we are picking I by gravitating towards higher rank numbers according to $\succ_{\Pi_{\mathsf{SI}}}$, while P_{J} is chosen independent of $\succ_{\Pi_{\mathsf{SI}}}$. We formalize this intuition in the following.

Claim 6.3.8. *Let* $\delta := \|\text{dist}(\mathsf{I} \mid \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I}) - \mathcal{U}_{[n]}\|_{tvd}$; *then* $\Pr(\mathsf{I} \prec_{\Pi_{\mathsf{SI}}} \mathsf{J} \mid \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I}) \leq \frac{1}{2} - \Omega(\delta)$.

Proof of Claim 6.3.8. In the following, all random variables are conditioned on $(\Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I})$ and hence with a slight abuse of notation we drop this conditioning throughout the proof. Recall that $\text{dist}(\mathsf{I}) = (p_1, \ldots, p_n)$ (by Eq (6.3)) and without loss of generality assume $p_1 \leq p_2 \leq \ldots \leq p_n$ as we can always rename the indices to obtain this property (and breaking the ties as in the protocol π_{PI} by the original index). As for the distribution of J, note that for any $\ell \in [n]$, $\Pr(\mathsf{J} \in [\ell + 1, n] \mid \mathsf{I} = \ell) = \frac{n-\ell}{n-1}$ by Eq (6.4). Note that after this renaming,

$\mathsf{I} \prec_{\Pi_{\mathsf{SI}}} \mathsf{J}$ iff $\mathsf{I} < \mathsf{J}$. Hence, we have,

$$\Pr\left(\mathsf{I} \prec_{\Pi_{\mathsf{SI}}} \mathsf{J}\right) = \Pr\left(\mathsf{I} < \mathsf{J}\right) = \sum_{\ell=1}^{n} \Pr\left(\mathsf{I} = \ell\right) \Pr(\mathsf{J} \in [\ell+1, n] \mid \mathsf{I} = \ell) = \sum_{\ell=1}^{n} p_\ell \cdot \frac{n-\ell}{n-1}.$$

Let $k \in [n]$ be the largest index such that $p_k < 1/n$. Define $q := \sum_{\ell=1}^{k} p_\ell$ as the total probability mass of indices with probability less than $1/n$. We have,

$$\delta = \|\mathsf{I} - \mathcal{U}_{[n]}\|_{tvd} = \frac{1}{2} \cdot \sum_{\ell=1}^{n} \left| p_\ell - \frac{1}{n} \right| = \frac{1}{2} \cdot \left((\frac{k}{n} - q) + ((1-q) - \frac{n-k}{n}) \right) = \frac{k}{n} - q \tag{6.5}$$

which implies that $q = \frac{k}{n} - \delta$. By the equation above for $\Pr\left(\mathsf{I} < \mathsf{J}\right)$, we have,

$$\Pr\left(\mathsf{I} < \mathsf{J}\right) = \sum_{\ell=1}^{k} p_\ell \cdot \frac{n-\ell}{n-1} + \sum_{\ell=k+1}^{n} p_\ell \cdot \frac{n-\ell}{n-1}.$$

Now, using the assumption that $p_1 \leq p_2 \leq \cdots \leq p_n$ and by the inequality of Proposition 2.2.1,

$$\begin{aligned}
\Pr\left(\mathsf{I} < \mathsf{J}\right) &\leq \frac{1}{k} \sum_{\ell=1}^{k} p_\ell \sum_{\ell=1}^{k} \frac{n-\ell}{n-1} + \frac{1}{n-k} \sum_{\ell=k+1}^{n} p_l \sum_{\ell=k+1}^{n} \frac{n-\ell}{n-1} \\
&= \frac{q}{k} \cdot \frac{k \cdot (2n-k-1)}{2n-2} + \frac{1-q}{n-k} \cdot \frac{(n-k-1)(n-k)}{2n-2} \\
&= q \cdot \frac{2n-k-1}{2n-2} + (1-q) \cdot \frac{n-k-1}{2n-2} = \frac{n-k-1}{2n-2} + q \cdot \frac{n}{2n-2} \\
&= \frac{1}{2} - \frac{k - n \cdot q}{2n-2} \underset{\text{Eq (6.5)}}{=} \frac{1}{2} - \frac{n\delta}{2n-2} < 1/2 - \delta/2,
\end{aligned}$$

completing the proof. □

We are now ready to finalize the proof of Lemma 6.3.5.

$$\begin{aligned}
\Pr\left(\pi_{\mathsf{PI}} \text{ errs} \mid x_1 \wedge y_1 = 1\right) &\underset{\text{Eq (6.2)}}{=} \mathop{\mathbb{E}}_{\Pi_{\mathsf{SI}} \sim \mathsf{\Pi_{SI}} \mid \mathsf{T}=\mathsf{I}} \left[\Pr\left(\mathsf{I} \prec_{\Pi_{\mathsf{SI}}} \mathsf{J} \mid \mathsf{\Pi_{SI}} = \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I}\right)\right] \\
&\underset{\text{Claim 6.3.8}}{\leq} \mathop{\mathbb{E}}_{\Pi_{\mathsf{SI}} \sim \mathsf{\Pi_{SI}} \mid \mathsf{T}=\mathsf{I}} \left[\frac{1}{2} - \Omega\left(\|\text{dist}(\mathsf{I} \mid \mathsf{\Pi_{SI}} = \Pi_{\mathsf{SI}}, \mathsf{T} = \mathsf{I}) - \mathcal{U}_{[n]}\|_{tvd}\right)\right]
\end{aligned}$$

$$= \mathop{\mathbb{E}}_{\Pi_{\mathsf{SI}} \sim \Pi_{\mathsf{SI}}} \left[\frac{1}{2} - \Omega\left(\|\mathrm{dist}(\mathsf{T} \mid \Pi_{\mathsf{SI}} = \Pi_{\mathsf{SI}}) - \mathcal{U}_{[n]}\|_{tvd} \right) \right]$$

(distribution of $\mathsf{I} = \mathsf{T}$ and $\Pi_{\mathsf{SI}} \perp \mathsf{T} = \mathsf{I}$)

$$\leq \frac{1}{2} - \Omega(\varepsilon),$$

where the last inequality is because π_{SI} ε-solves Set-Int. We can also do the same exact analysis for the case when $x_2 \wedge y_2 = 1$, hence obtaining that $\Pr(\pi_{\mathsf{PI}} \text{ errs}) = \frac{1}{2} - \Omega(\varepsilon)$. □

Proof of Lemma 6.3.6: Information Cost of Protocol π_{PI}

We prove this lemma by a direct-sum type argument that shows if the (internal) information cost of π_{SI} is small, then protocol π_{PI} is revealing a small information about its input *assuming conditioning on the target element*. We emphasize that this information revealed is *not* equivalent with the internal information cost as we are conditioning on some information not known to neither Alice nor Bob. The following is a restatement of Lemma 6.3.6 that we prove in this section.

Lemma (Restatement of Lemma 6.3.6)**.** *Let Π_{PI} denote the random variable for the transcript of the protocol π_{PI} and K be the random variable for index k in distribution* $\mathrm{dist}_{\mathsf{PI}}$*. We have,*

$$\mathbb{I}_{\mathrm{dist}_{\mathsf{PI}}}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}) + \mathbb{I}_{\mathrm{dist}_{\mathsf{PI}}}(\mathsf{Y}_1, \mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_1, \mathsf{X}_2, \mathsf{K}) \leq \frac{1}{n-1} \cdot \mathsf{IC}_{\mathrm{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}}).$$

The intuition behind the proof is as follows. The LHS in Lemma 6.3.6 is the information revealed about the input of players (in Pair-Int) averaged over choices of $k = 1$ and $k = 2$. Let us assume $k = 1$, as the other case is symmetric. In this case, this quantity is simply the information revealed about (x_2, y_2) by the protocol as $(x_1, y_1) = (1, 1)$ and hence has 0 information (once we have conditioned on the event $k = 1$). However, when $k = 1$, (x_2, y_2) is embedded in index j, i.e., $(x_2, y_2) = (a_j, b_j)$ and have the same distribution as all other coordinates in A_{-i}, B_{-i}. As such, since the protocol π_{SI} called inside π_{PI} is oblivious to the choice of j, the information revealed about (a_j, b_j) in average is smaller than the information

revealed by π_{SI} about A_{-i}, B_{-i} (which itself is at most the internal information cost of π_{SI}), by a factor of $n-1$ (i.e., the number of coordinates in $[n] \setminus \{i\}$ we are averaging over).

The outline above oversimplifies many details. One such detail is the way of ensuring a "symmetric treatment" of both indices i and j through the rather peculiar choice of public-private sampling in π_{PI} (via the choices of ℓ and S). This is crucial for the above argument to work for both $k=1$ and $k=2$ cases simultaneously, without the players knowing which index the "averaging" of information is being done for (index j in the context of the discussion above).

Proof of Lemma 6.3.6. For simplicity of exposition, we drop the subscript $\mathrm{dist}_{\mathsf{PI}}$ from all mutual information terms with the understanding that all random variables are distributed according to $\mathrm{dist}_{\mathsf{PI}}$ (and the randomness of protocol π_{PI} on $\mathrm{dist}_{\mathsf{PI}}$) unless explicitly stated otherwise.

We bound the first term in LHS above (the second term can be bounded the same way). By expanding the conditional mutual information term we have,

$$\mathbb{I}(\mathsf{X}_1, \mathsf{X}_2 \,;\, \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}) = \frac{1}{2} \cdot \mathbb{I}(\mathsf{X}_1, \mathsf{X}_2 \,;\, \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}=1) + \frac{1}{2} \cdot \mathbb{I}(\mathsf{X}_1, \mathsf{X}_2 \,;\, \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}=2). \tag{6.6}$$

We now focus on the first term in the LHS of Eq (6.6). We have,

$$\begin{aligned}
\mathbb{I}(\mathsf{X}_1, \mathsf{X}_2 \,;\, \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}=1) &= \mathbb{I}(\mathsf{X}_2 \,;\, \Pi_{\mathsf{PI}} \mid \mathsf{Y}_2, \mathsf{K}=1) \\
&\qquad ((\mathsf{X}_1, \mathsf{Y}_1) \text{ is always equal to } (1,1) \text{ in } \mathrm{dist}_{\mathsf{PI}} \text{ conditioned on } \mathsf{K}=1) \\
&= \mathbb{I}(\mathsf{X}_2 \,;\, \Pi_{\mathsf{SI}} \mid \mathsf{Y}_2, \mathsf{I}, \mathsf{J}, \mathsf{S}, \mathsf{L}, \mathsf{A}_{\mathsf{S}}, \mathsf{B}_{\overline{\mathsf{S}}}, \mathsf{K}=1) \\
&\qquad (\pi_{\mathsf{PI}} \text{ runs } \pi_{\mathsf{SI}} \text{ with public randomness } \mathsf{I}, \mathsf{J}, \mathsf{S}, \mathsf{L}, \mathsf{A}_{\mathsf{S}}, \mathsf{B}_{\overline{\mathsf{S}}} \ (\mathsf{L} \text{ is for } \ell) \text{ and by Proposition 2.5.3}) \\
&= \sum_{i \neq j} \frac{1}{n(n-1)} \cdot \mathbb{I}(\mathsf{A}_j \,;\, \Pi_{\mathsf{SI}} \mid \mathsf{B}_j, \mathsf{L}, \mathsf{S}, \mathsf{A}_{\mathsf{S}}, \mathsf{B}_{\overline{\mathsf{S}}}, \mathsf{I}=i, \mathsf{J}=j, \mathsf{K}=1). \\
&\qquad ((\mathsf{X}_2, \mathsf{Y}_2) \text{ is embedded in } (\mathsf{A}_j, \mathsf{B}_j) \text{ conditioned on } \mathsf{J}=j)
\end{aligned}$$

Recall that T denotes the unique index in $[n]$ in instances $(A, B) \sim \mathsf{dist}_{\mathsf{SI}}$ which is equal to $(1, 1)$. Note that $\mathsf{T} = i$ conditioned on $\mathsf{I} = i$ and $\mathsf{K} = 1$, and that conditioning on the event $\mathsf{T} = i$ has the same effect on all random variables above as conditioning on the joint event $\mathsf{I} = i, \mathsf{K} = 1$. Hence, we can write the RHS above as,

$$\begin{aligned}\mathbb{I}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K} = 1) &= \frac{1}{n(n-1)} \sum_{i \neq j} \mathbb{I}(\mathsf{A}_j \,; \Pi_{\mathsf{SI}} \mid \mathsf{B}_j, \mathsf{L}, \mathsf{S}, \mathsf{A}_{\mathsf{S}}, \mathsf{B}_{\overline{\mathsf{S}}}, \mathsf{T} = i, \mathsf{J} = j) \\ &\leq \frac{1}{n(n-1)} \sum_{i \neq j} \mathbb{I}(\mathsf{A}_j \,; \Pi_{\mathsf{SI}} \mid \mathsf{L}, \mathsf{S}, \mathsf{A}_{\mathsf{S}}, \mathsf{B}_{-i}, \mathsf{T} = i, \mathsf{J} = j).\end{aligned}$$

(as $\mathsf{A}_j \perp \mathsf{B}_{-i} \mid \mathsf{B}_j$ (and other variables above) and hence we can apply Proposition 2.4.2)

By further expanding the conditional mutual information term in RHS over L and S,

$$\begin{aligned}&\mathbb{I}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K} = 1) \\ &\leq \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{\ell=0}^{n-2} \sum_{\substack{S \subseteq [n] \setminus \{i,j\} \\ |S| = \ell}} \frac{1}{n-1} \binom{n-2}{\ell}^{-1} \qquad (6.7) \\ &\quad \cdot \mathbb{I}(\mathsf{A}_j \,; \Pi_{\mathsf{SI}} \mid \mathsf{A}_S, \mathsf{B}_{-i}, \mathsf{L} = \ell, \mathsf{S} = S, \mathsf{T} = i, \mathsf{J} = j) \\ &= \frac{1}{n(n-1) \cdot (n-1)!} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{\ell=0}^{n-2} \sum_{\substack{S \subseteq [n] \setminus \{i,j\} \\ |S| = \ell}} ((n-2-\ell)!\ell!) \cdot \mathbb{I}(\mathsf{A}_j \,; \Pi_{\mathsf{SI}} \mid \mathsf{A}_S, \mathsf{B}_{-i}, \mathsf{T} = i), \qquad (6.8)\end{aligned}$$

by reorganization of the terms and dropping the conditioning on events $\mathsf{L} = \ell, \mathsf{S} = S, \mathsf{J} = j$ as the distribution of remaining random variables are independent of these events. We now have the following auxiliary claim.

Claim 6.3.9. *For any choice of $i \in [n]$,*

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{\ell=0}^{n-2} \sum_{\substack{S \subseteq [n] \setminus \{i,j\} \\ |S| = \ell}} ((n-2-\ell)!\ell!) \cdot \mathbb{I}(\mathsf{A}_j \,; \Pi_{\mathsf{SI}} \mid \mathsf{A}_S, \mathsf{B}_{-i}, \mathsf{T} = i)$$

$$= \sum_{\sigma \in \mathcal{S}_{-i}} \sum_{\ell=0}^{n-2} \mathbb{I}(\mathsf{A}_{\sigma(\ell+1)}\,; \Pi_{\mathsf{SI}} \mid \mathsf{A}_{\sigma(<\ell+1)}, \mathsf{B}_{-i}, \mathsf{T} = i),$$

where $\mathcal{S}_{-i}$ is the set of all permutations of $[n] \setminus \{i\}$.

Proof. Fix any (j, S) in the LHS. For integer $\ell = |S|$, there are exactly $((n-2-\ell)!\ell!)$ permutations $\sigma \in \mathcal{S}_{-i}$ such that (i) $\sigma(\ell+1) = j$ and (ii) $\{\sigma(1), \ldots, \sigma(\ell)\} = S$. Hence, $\mathbb{I}(\mathsf{A}_j\,; \Pi_{\mathsf{SI}} \mid \mathsf{A}_S, \mathsf{B}_{-i}, \mathsf{T} = i)$ for (j, S) appears exactly $((n-2-\ell)!\ell!)$ times in RHS as $\mathbb{I}(\mathsf{A}_{\sigma(\ell+1)}\,; \Pi_{\mathsf{SI}} \mid \mathsf{A}_{\sigma(<\ell+1)}, \mathsf{B}_{-i}, \mathsf{T} = i)$ (for appropriate choices of σ as described above), proving the claim.

□

By applying Claim 6.3.9 to the RHS of Eq (6.8), we obtain that,

$$\begin{aligned}
&\mathbb{I}(\mathsf{X}_1, \mathsf{X}_2\,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K} = 1) \\
&\leq \frac{1}{n(n-1)(n-1)!} \sum_{i=1}^{n} \sum_{\sigma \in \mathcal{S}_{-i}} \sum_{\ell=0}^{n-2} \mathbb{I}(\mathsf{A}_{\sigma(\ell+1)}\,; \Pi_{\mathsf{SI}} \mid \mathsf{A}_{\sigma(<\ell+1)}, \mathsf{B}_{-i}, \mathsf{T} = i) \\
&= \frac{1}{n(n-1)(n-1)!} \sum_{i=1}^{n} \sum_{\sigma \in \mathcal{S}_{-i}} \mathbb{I}(\mathsf{A}_{-i}\,; \Pi_{\mathsf{SI}} \mid \mathsf{B}_{-i}, \mathsf{T} = i) \\
&\qquad\qquad \text{(by chain rule of mutual information in Fact 2.4.1-(6))} \\
&= \frac{1}{n(n-1)} \sum_{i=1}^{n} \mathbb{I}(\mathsf{A}_{-i}\,; \Pi_{\mathsf{SI}} \mid \mathsf{B}_{-i}, \mathsf{T} = i) \qquad \text{(as } |\mathcal{S}_{-i}| = (n-1)!\text{)} \\
&= \frac{1}{n-1} \cdot \mathbb{I}(\mathsf{A}\,; \Pi_{\mathsf{SI}} \mid \mathsf{B}, \mathsf{T}) \\
&\text{(as } (\mathsf{A}_{\mathsf{T}}, \mathsf{B}_{\mathsf{T}}) = (1,1) \text{ in } \mathrm{dist}_{\mathsf{PI}} \text{ and hence we can add them to the information term)} \\
&\leq \frac{1}{n-1} \cdot \mathbb{I}(\mathsf{A}\,; \Pi_{\mathsf{SI}} \mid \mathsf{B}) = \frac{1}{n-1} \cdot \mathbb{I}_{\mathrm{dist}_{\mathsf{SI}}}(\mathsf{A}\,; \Pi_{\mathsf{SI}} \mid \mathsf{B}),
\end{aligned}$$

where the last inequality is because $\Pi_{\mathsf{SI}} \perp \mathsf{T} \mid \mathsf{A}, \mathsf{B}$ (as the transcript is only a function of the inputs) and hence we can apply Proposition 2.4.3, and the last equality is because by Observation 6.3.3, joint distribution of $\mathrm{dist}_{\mathsf{PI}}$ and randomness of the protocol π_{PI} is the same

as distribution $\mathsf{dist}_{\mathsf{SI}}$. Using the same exact analysis (by switching the role of indices i and j and noting that the rest is all symmetric), we also obtain the following bound for the second term of Eq (6.6),

$$\mathbb{I}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K} = 2) \leq \frac{1}{n-1} \cdot \mathbb{I}_{\mathsf{dist}_{\mathsf{SI}}}(\mathsf{A} \,; \Pi_{\mathsf{SI}} \mid \mathsf{B}).$$

Plugging in these bounds in Eq (6.6), we obtain that,

$$\mathbb{I}_{\mathsf{dist}_{\mathsf{PI}}}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}) \leq \frac{1}{n-1} \cdot \mathbb{I}_{\mathsf{dist}_{\mathsf{SI}}}(\mathsf{A} \,; \Pi_{\mathsf{SI}} \mid \mathsf{B}). \tag{6.9}$$

Similarly, the second term in the LHS of Lemma 6.3.6 can be upper bounded using a similar analysis (by switching the role of A and B, and S and $\overline{\mathsf{S}}$ and noting that the rest is all symmetric), implying the following bound:

$$\mathbb{I}_{\mathsf{dist}_{\mathsf{PI}}}(\mathsf{Y}_1, \mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_1, \mathsf{X}_2, \mathsf{K}) \leq \frac{1}{n-1} \cdot \mathbb{I}_{\mathsf{dist}_{\mathsf{SI}}}(\mathsf{B} \,; \Pi_{\mathsf{SI}} \mid \mathsf{A}). \tag{6.10}$$

Summing up the LHS and RHS in Eq (6.9) and Eq (6.10), finalizes the proof. □

Proof of Lemma 6.3.7: Information Complexity of Pair-Int

We now prove the final step of the proof of Theorem 6.5. The following is a restatement of Lemma 6.3.7.

Lemma (Restatement of Lemma 6.3.7)**.** *Suppose π_{PI} outputs the correct answer on $\mathsf{dist}_{\mathsf{PI}}$ with probability at least $\frac{1}{2} + \Omega(\varepsilon)$. Then,*

$$\mathbb{I}_{\mathsf{dist}_{\mathsf{PI}}}(\mathsf{X}_1, \mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{K}) + \mathbb{I}_{\mathsf{dist}_{\mathsf{PI}}}(\mathsf{Y}_1, \mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_1, \mathsf{X}_2, \mathsf{K}) = \Omega(\varepsilon^2).$$

The idea behind the proof of Lemma 6.3.7 is as follows. Recall that $\Pi_{[x_1x_2, y_1y_2]}$ denotes the transcript of the protocol condition on the input being $[x_1x_2, y_1y_2]$. Suppose towards the contradiction that the LHS of Lemma 6.3.7 is $o(\varepsilon^2)$ instead. By focusing on the conditional

terms when $k = 1$, we can show that distribution of $\Pi_{[1x_2', 1y_2']}$ and $\Pi_{[1x_2'', 1y_2'']}$ for all choices of (x_2', y_2') and (x_2'', y_2'') in the support of $\mathsf{dist}_{\mathsf{PI}}$ (basically everything except for $(1, 1)$) are quite close. This is intuitively because the information revealed about (x_2, y_2) by π_{PI} conditioned on $k = 1$ is small. Similarly, by focusing on the $k = 2$ terms, we obtain the same result for $\Pi_{[x_2'1, y_2'1]}$ and $\Pi_{[x_2''1, y_2''1]}$.

Up until this point, there is no contradiction as the answer to $[1*, 1*]$ is always 1 and hence there is no problem with the corresponding transcripts in $\Pi_{[1*, 1*]}$ to be similar (similarly for $\Pi_{[*1, *1]}$ separately). However, we combine the previous part with the cut-and-paste property of randomized protocols (Fact 2.5.6) to argue that in fact the distribution of $\Pi_{[10, 10]}$ and $\Pi_{[01, 01]}$ are also similar. This then basically implies that $\Pi_{[1*, 1*]}$ essentially has the same distribution as $\Pi_{[*1, *1]}$; but then this is a contradiction as the answer to the protocol (which is only a function of the transcript) needs to be different between these two types of inputs. We now formalize the proof.

Proof of Lemma 6.3.7. The distribution of random variables below is always $\mathsf{dist}_{\mathsf{PI}}$ (and the randomness of the protocol π_{PI} on $\mathsf{dist}_{\mathsf{PI}}$) and hence we drop the subscript $\mathsf{dist}_{\mathsf{PI}}$ from all mutual information terms. Suppose towards a contradiction that the LHS in the lemma statement is $o(\varepsilon^2)$. As we showed in Eq (6.6) and the subsequent equation in the proof of Lemma 6.3.6, the LHS can be written as

$$\frac{1}{2} \cdot (\mathbb{I}(\mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_2, \mathsf{K} = 1) + \mathbb{I}(\mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_2, \mathsf{K} = 1)) + \frac{1}{2} \cdot (\mathbb{I}(\mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_2, \mathsf{K} = 1) + \mathbb{I}(\mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_2, \mathsf{K} = 1)) = o(\varepsilon^2). \tag{6.11}$$

By bounding each of the above term above separately by $o(\varepsilon^2)$ and expanding the mutual information terms, we prove the following claim.

Claim 6.3.10. *Assuming Eq (6.11),*

$$(1)\ \mathbb{I}(\mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_2 = 0, \mathsf{K} = 1) = o(\varepsilon^2), \qquad (2)\ \mathbb{I}(\mathsf{Y}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{X}_2 = 0, \mathsf{K} = 1) = o(\varepsilon^2),$$

$$(3)\ \mathbb{I}(\mathsf{X}_1\,;\Pi_{\mathsf{PI}}\mid \mathsf{Y}_1=0,\mathsf{K}=2)=o(\varepsilon^2),\qquad (4)\ \mathbb{I}(\mathsf{Y}_1\,;\Pi_{\mathsf{PI}}\mid \mathsf{X}_1=0,\mathsf{K}=2)=o(\varepsilon^2).$$

Proof. To prove the first equation, we write the first term in Eq (6.11) as follows:

$$\begin{aligned}\mathbb{I}(\mathsf{X}_2\,;\Pi_{\mathsf{PI}}\mid \mathsf{Y}_2,\mathsf{K}=1)&=\frac{2}{3}\cdot\mathbb{I}(\mathsf{X}_2\,;\Pi_{\mathsf{PI}}\mid \mathsf{Y}_2=0,\mathsf{K}=1)+\frac{1}{3}\cdot\mathbb{I}(\mathsf{X}_2\,;\Pi_{\mathsf{PI}}\mid \mathsf{Y}_2=1,\mathsf{K}=1)\\&=\frac{2}{3}\cdot\mathbb{I}(\mathsf{X}_2\,;\Pi_{\mathsf{PI}}\mid \mathsf{Y}_2=0,\mathsf{K}=1),\end{aligned}$$

since for $(\mathsf{X}_2,\mathsf{Y}_2)\sim \mathsf{dist}_{\mathsf{PI}}\mid \mathsf{K}=1$, if $\mathsf{Y}_2=1$, then X_2 is always equal to 0 and hence the second term above is zero. As the LHS of above equation is $o(\varepsilon^2)$ by Eq (6.11) (and non-negativity of mutual information in Fact 2.4.1-(2)), we obtain the first equation in the statement of the claim. The remaining equations can be proven exactly the same. □

We now use Claim 6.3.10, to bound the distance between different transcripts of the protocol. Recall that $\Pi_{[x_1x_2,\,y_1y_2]}$ denotes the transcript of the protocol conditioned on the input (x_1,x_2) to Alice, and (y_1,y_2) to Bob.

Claim 6.3.11. *Assuming Eq (6.11),*

$$(1)\ \mathrm{h}^2(\Pi_{[11,\,10]},\Pi_{[10,\,10]})=o(\varepsilon^2),\qquad (2)\ \mathrm{h}^2(\Pi_{[10,\,11]},\Pi_{[10,\,10]})=o(\varepsilon^2),$$
$$(3)\ \mathrm{h}^2(\Pi_{[11,\,01]},\Pi_{[01,\,01]})=o(\varepsilon^2),\qquad (4)\ \mathrm{h}^2(\Pi_{[01,\,11]},\Pi_{[01,\,01]})=o(\varepsilon^2).$$

Proof. We write the LHS of the first equation in Claim 6.3.10 in terms of the KL-divergence using Fact 2.4.5. Define $\Pi_{[1*,\,10]}$ as the distribution of Π conditioned on the given value for x_1,y_1,y_2 (leaving out the assignment for x_2). We have,

$$\begin{aligned}\mathbb{I}(\mathsf{X}_2\,;\Pi_{\mathsf{PI}}\mid \mathsf{Y}_2=0,\mathsf{K}=1)&\underset{\text{Fact 2.4.5}}{=}\mathop{\mathbb{E}}_{x_2\sim \mathsf{X}_2\mid \mathsf{Y}_2=0,\mathsf{K}=1}[\mathbb{D}(\Pi_{[1x_2,\,10]}\;||\;\Pi_{[1*,\,10]})]\\&=\frac{1}{2}\cdot\mathbb{D}(\Pi_{[10,\,10]}\;||\;\Pi_{[1*,\,10]})+\frac{1}{2}\cdot\mathbb{D}(\Pi_{[11,\,10]}\;||\;\Pi_{[1*,\,10]})\\&\underset{\text{Fact 2.4.9}}{\geq}\mathrm{h}^2(\Pi_{[10,\,10]},\Pi_{[11,\,10]}).\end{aligned}$$

The distribution of X_2 conditioned on $\mathsf{Y}_2 = 0, \mathsf{K} = 1$ in $\mathrm{dist}_{\mathsf{PI}}$ is uniform over $\{0,1\}$ (hence the second equality). As such, $\Pi_{[1*,10]} = \frac{1}{2} \cdot \left(\Pi_{[10,10]} + \Pi_{[11,10]}\right)$ and so we can apply Fact 2.4.9 to obtain the last inequality. As $\mathbb{I}(\mathsf{X}_2 \,; \Pi_{\mathsf{PI}} \mid \mathsf{Y}_2 = 0, \mathsf{K} = 1) = o(\varepsilon^2)$ by Claim 6.3.10, we obtain the first equation (note that h is symmetric). The remaining equations can be proven similarly. □

The next step is to use the cut-and-paste property (Fact 2.5.6) of randomized protocols to prove the following claim.

Claim 6.3.12. *Assuming Eq (6.11),* $\mathrm{h}^2(\Pi_{[10,10]}, \Pi_{[01,01]}) = o(\varepsilon^2)$.

Proof. We start with proving the following two equations first:

$$(1)\ \mathrm{h}^2(\Pi_{[11,11]}, \Pi_{[10,10]}) = o(\varepsilon^2), \qquad (2)\ \mathrm{h}^2(\Pi_{[11,11]}, \Pi_{[01,01]}) = o(\varepsilon^2).$$

For the first equation,

$$\begin{aligned}
\mathrm{h}^2(\Pi_{[11,11]}, \Pi_{[10,10]}) &= \mathrm{h}^2(\Pi_{[11,10]}, \Pi_{[10,11]}) && \text{(by the cut-and-paste property in Fact 2.5.6)} \\
&\le \left(\mathrm{h}(\Pi_{[11,10]}, \Pi_{[10,10]}) + \mathrm{h}(\Pi_{[10,10]}, \Pi_{[10,11]})\right)^2 && \text{(by triangle inequality)} \\
&\le 2 \cdot \left(\mathrm{h}^2(\Pi_{[11,10]}, \Pi_{[10,10]}) + \mathrm{h}^2(\Pi_{[10,10]}, \Pi_{[10,11]})\right) \\
& && \text{(by Cauchy-Schwartz)} \\
&= o(\varepsilon^2). && \text{(by parts (1) and (2) of Claim 6.3.11)}
\end{aligned}$$

The second equation can be proven similarly using parts (3) and (4) of Claim 6.3.11. We can now prove the claim as follows:

$$\begin{aligned}
\mathrm{h}^2(\Pi_{[10,10]}, \Pi_{[01,01]}) &\le \left(\mathrm{h}(\Pi_{[10,10]}, \Pi_{[11,11]}) + \mathrm{h}(\Pi_{[11,11]}, \Pi_{[01,01]})\right)^2 && \text{(by triangle inequality)} \\
&\le 2 \cdot \left(\mathrm{h}^2(\Pi_{[10,10]}, \Pi_{[11,11]}) + \mathrm{h}^2(\Pi_{[11,11]}, \Pi_{[01,01]})\right) \\
& && \text{(by Cauchy-Schwartz)}
\end{aligned}$$

$$= o(\varepsilon^2). \qquad \text{(by part (1) and (2) of the equation above)}$$

This concludes the proof. □

Define $I_1 := \{[10,10],[11,10],[10,11]\}$ and $I_2 := \{[01,01],[11,01],[01,11]\}$. The tuples in $I_1 \cup I_2$ partition all the input tuples in the support of $\mathrm{dist}_{\mathsf{PI}}$ and moreover, for every tuple in I_1, the correct answer to Pair-Int is the first index, while for every tuple in I_2, the correct answer is the second index. We now bound the total variation distance between every pair of tuples in I_1 and I_2.

Claim 6.3.13. *Assuming Eq (6.11), for every* $(T_1, T_2) \in I_1 \times I_2$, $\|\Pi_{T_1} - \Pi_{T_2}\|_{tvd} = o(\varepsilon)$.

Proof. Proving the claim amounts to proving the following nine equations:

$$(1)\ \|\Pi_{[10,10]} - \Pi_{[01,01]}\|_{tvd} = o(\varepsilon),$$
$$(2)\ \|\Pi_{[10,10]} - \Pi_{[11,01]}\|_{tvd} = o(\varepsilon),$$
$$(3)\ \|\Pi_{[10,10]} - \Pi_{[01,11]}\|_{tvd} = o(\varepsilon),$$
$$(4)\ \|\Pi_{[11,10]} - \Pi_{[01,01]}\|_{tvd} = o(\varepsilon),$$
$$(5)\ \|\Pi_{[11,10]} - \Pi_{[11,01]}\|_{tvd} = o(\varepsilon),$$
$$(6)\ \|\Pi_{[11,10]} - \Pi_{[01,11]}\|_{tvd} = o(\varepsilon),$$
$$(7)\ \|\Pi_{[10,11]} - \Pi_{[01,01]}\|_{tvd} = o(\varepsilon),$$
$$(8)\ \|\Pi_{[10,11]} - \Pi_{[11,01]}\|_{tvd} = o(\varepsilon),$$
$$(9)\ \|\Pi_{[10,11]} - \Pi_{[01,11]}\|_{tvd} = o(\varepsilon),$$

The first equation can be proven as follows:

$$\|\Pi_{[10,10]} - \Pi_{[01,01]}\|_{tvd} \le \sqrt{2} \cdot \mathrm{h}(\Pi_{[10,10]}, \Pi_{[01,01]}) = o(\varepsilon),$$

where the inequality is by Fact 2.4.8 and the equality is by Claim 6.3.12. This proves the

equation (1) above. Now note that,

$$\|\Pi_{[10,10]} - \Pi_{[11,01]}\|_{tvd} \leq \|\Pi_{[10,10]} - \Pi_{[01,01]}\|_{tvd} + \|\Pi_{[01,01]} - \Pi_{[11,01]}\|_{tvd}$$

(by triangle inequality)

$$\leq o(\varepsilon) + \sqrt{2} \cdot \mathrm{h}(\Pi_{[01,01]}, \Pi_{[11,01]})$$

(by equation (1) above for the first term and Fact 2.4.8 for the second)

$$= o(\varepsilon).$$ (by part (3) of Claim 6.3.11)

This proves the equation (2). All the remaining equations can now be proven using a similar argument as above by first relating the distance between the two variables to the distance between $\|\Pi_{[10,10]} - \Pi_{[11,01]}\|_{tvd}$ (which we know is $o(\varepsilon)$ by equation (1)) using triangle inequality, and then use Fact 2.4.8 combined with Claim 6.3.11 to bound each of the remaining terms with $o(\varepsilon)$. □

We are now almost done. By Claim 6.3.13, if we assume Eq (6.11), then for every $(T_1, T_2) \in I_1 \times I_2$, $\|\Pi_{T_1} - \Pi_{T_2}\|_{tvd} = o(\varepsilon)$. On the other hand, for π_{PI} to be able to output the correct answer with probability $1/2 + \Omega(\varepsilon)$ (over the randomness of the protocol and the distribution), for at least one pair $(T_1, T_2) \in I_1 \times I_2$, we should have $\|\Pi_{T_1} - \Pi_{T_2}\|_{tvd} = \Omega(\varepsilon)$ as the output of the protocol on T_1 (resp. T_2) is only a function of Π_{T_1} (resp. Π_{T_2}), and hence otherwise would be the same with probability $1 - o(\varepsilon)$ by Fact 2.4.6. This implies that assuming Eq (6.11), the protocol errs with probability at least $1/2 - o(\varepsilon)$, which is a contradiction. Hence Eq (6.11) cannot hold □

6.4. The Hidden-Pointer Chasing Problem

Recall that the hidden-pointer chasing (HPC) problem is a *four-party* communication problem with players P_A, P_B, P_C, and P_D defined as follows. Let $\mathcal{X} := \{x_1, \ldots, x_n\}$ and $\mathcal{Y} := \{y_1, \ldots, y_n\}$ be two disjoint universes of size n each. We define HPC as follows:

1. For any $x \in \mathcal{X}$, P_A and P_B are given an instance (A_x, B_x) of Set-Int over the uni-

verse $\mathcal{Y}$ where $A_x \cap B_x = \{t_x\}$ for a single target element $t_x \in \mathcal{Y}$. We define $\boldsymbol{A} := \{A_{x_1}, \ldots, A_{x_n}\}$ and $\boldsymbol{B} := \{B_{x_1}, \ldots, B_{x_n}\}$ as the whole input to P_A and P_B, respectively.

2. For any $y \in \mathcal{Y}$, P_C and P_D are given an instance (C_y, D_y) of Set-Int over the universe $\mathcal{X}$ where $C_y \cap D_y = \{t_y\}$ for a single target element $t_y \in \mathcal{X}$. We define $\overline{C} := \{C_{y_1}, \ldots, C_{y_n}\}$ and $\boldsymbol{D} := \{D_{y_1}, \ldots, D_{y_n}\}$ as the whole input to P_C and P_D, respectively.

3. We define two mappings $f_{AB} : \mathcal{X} \to \mathcal{Y}$ and $f_{CD} : \mathcal{Y} \to \mathcal{X}$ such that:

   (a) for any $x \in \mathcal{X}$, $f_{AB}(x) = t_x \in \mathcal{Y}$ in the instance (A_x, B_x) of Set-Int.

   (b) for any $y \in \mathcal{Y}$, $f_{CD}(y) = t_y \in \mathcal{X}$ in the instance (C_y, D_y) of Set-Int.

4. Let $x_1 \in \mathcal{X}$ be an arbitrary fixed element of $\mathcal{X}$ known to all players. The pointers $z_0, z_1, z_2, z_3, \ldots$ are defined inductively as follows:

$$z_0 := x_1, \qquad z_1 := f_{AB}(z_0), \qquad z_2 := f_{CD}(z_1), \qquad z_3 := f_{AB}(z_2), \qquad \ldots.$$

For any integer $k \geq 1$, the k-step hidden-pointer chasing problem, denoted by HPC_k is defined as the communication problem of finding the pointer z_k. See Figure 6.1 on page 189 for an illustration.

6.4.1. Communication Complexity of HPC_k

It is easy to see that in $k+1$ phases, we can compute HPC_k with $O(k \cdot n)$ total communication: we simply skip the first phase; in the second phase, P_A and P_B solve the Set-Int instance (A_{z_0}, B_{z_0}) with $O(n)$ communication to compute $z_1 = f_{AB}(z_0)$ and send this pointer to P_C and P_D; P_C and P_D in the next phase compute $f_{CD}(z_1)$ and the players continue like this to find the pointer z_k, which takes $k+1$ phases in total.

In the following, we prove that if we only have k phases however, solving HPC_k requires $\Omega(n^2/k^2 + n)$ bits of communication.

Theorem 6.6. *For any integer $k \geq 1$, any k-phase protocol that outputs the correct solution to HPC_k with constant probability requires $\Omega(n^2/k^2 + n)$ bits of communication.*

The rest of this section is devoted to the proof of Theorem 6.6. We start with defining our hard distribution of instances for HPC_k and then use this distribution to prove the lower bound.

A Hard Distribution for HPC

The hard distribution for HPC is simply the product of distribution $\mathrm{dist}_{\mathsf{SI}}$ for every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

Distribution $\mathrm{dist}_{\mathsf{HPC}}$ on tuples $(\boldsymbol{A}, \boldsymbol{B}, \overline{C}, \boldsymbol{D})$ from the universes $\mathcal{X}$ and $\mathcal{Y}$:

1. For any $x \in \mathcal{X}$, sample $(A_x, B_x) \sim \mathrm{dist}_{\mathsf{SI}}$ from the universe $\mathcal{Y}$ *independently.*
2. For any $y \in \mathcal{Y}$, sample $(C_y, D_y) \sim \mathrm{dist}_{\mathsf{SI}}$ from the universe $\mathcal{X}$ *independently.*

The following simple observation is in order.

Observation 6.4.1. Distribution $\mathrm{dist}_{\mathsf{HPC}}$ is <u>not</u> a product distribution. However, in this distribution:

(i) The inputs to P_A and P_B are independent of the inputs to P_C and P_D, i.e., $(\boldsymbol{A}, \boldsymbol{B}) \perp (\overline{C}, \boldsymbol{D})$.

(ii) For any $x \in \mathcal{X}$, (A_x, B_x) is independent of all other $(A_{x'}, B_{x'})$ for $x' \neq x \in \mathcal{X}$. Similarly for all $y, y' \in \mathcal{Y}$ and (C_y, D_y) and $(C_{y'}, D_{y'})$.

Based on this observation, we also have the following simple property.

Proposition 6.4.2. *Let π_{HPC} be any deterministic protocol for HPC_k on $\mathrm{dist}_{\mathsf{HPC}}$. Then, for any transcript Π of π_{HPC}, $(\boldsymbol{A}, \boldsymbol{B}) \perp (\overline{C}, \boldsymbol{D}) \mid \Pi = \Pi$.*

Proof. Follows from the rectangle property of the protocol π_{HPC} (Fact 2.5.5). In particular,

the same exact argument as in the two-player case implies that if $[(\boldsymbol{A}_1, \boldsymbol{B}_1), (\overline{C}_1, \boldsymbol{D}_1)]$ and $[(\boldsymbol{A}_2, \boldsymbol{B}_2), (\overline{C}_2, \boldsymbol{D}_2)]$ are mapped to the same transcript Π, then $[(\boldsymbol{A}_1, \boldsymbol{B}_1), (\overline{C}_2, \boldsymbol{D}_2)]$ and $[(\boldsymbol{A}_2, \boldsymbol{B}_2), (\overline{C}_1, \boldsymbol{D}_1)]$ are mapped to Π as well. Hence, since $(\boldsymbol{A}, \boldsymbol{B}) \perp (\overline{C}, \boldsymbol{D})$ by Observation 6.4.1, the inputs corresponding to the same protocol would also be independent of each other, namely, $(\boldsymbol{A}, \boldsymbol{B}) \perp (\overline{C}, \boldsymbol{D}) \mid \Pi = \Pi$. □

Proof of Theorem 6.6: A Communication Lower Bound for HPC_k

We prove the lower bound for any arbitrary deterministic protocol π_{HPC} and then apply Yao's minimax principle [252] to extend it to randomized protocols as well. We first setup some notation.

Notation. Fix any k-phase *deterministic* protocol π_{HPC} for HPC_k throughout the proof. We use $j = 1$ to k to index the phases of this protocol, as well as the pointers $z_1, \ldots, z_k$. For any $j \in [k]$, we define Π_j as the set of all messages communicated by π_{HPC} in phase j and $\Pi := (\Pi_1, \ldots, \Pi_k)$ as the transcript of the protocol π_{HPC}.

For any $x \in \mathcal{X}$ and any $y \in \mathcal{Y}$, we define the random variables $\mathsf{T}_x \in \mathcal{Y}$ and $\mathsf{T}_y \in \mathcal{X}$, which correspond to the target elements of the Set-Int problem on (A_x, B_x) and (C_y, D_y), respectively.

We further define $\mathsf{E}_j := (\Pi^{<j}, \mathsf{Z}^{<j})$ for any $j > 1$ and $\mathsf{E}_1 = z_0$, i.e., the first pointer. We can think of E_j as the information "easily known" to all players at the beginning of phase j.

The main step of the proof of Theorem 6.6 is the following key lemma which we prove inductively.

Lemma 6.4.3. *Let* $\mathsf{CC}(\pi_{\mathsf{HPC}}) := \mathsf{CC}_{\mathrm{dist}_{\mathsf{HPC}}}(\pi_{\mathsf{HPC}})$. *There exists an absolute constant* $c > 0$ *such that for all* $j \in [k]$*:*

$$\mathop{\mathbb{E}}_{(E_j, \Pi_j)} \Big[\|\mathrm{dist}(\mathsf{Z}_j \mid E_j, \Pi_j) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd} \Big] \leq j \cdot c \cdot \Big(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}}) + k \cdot \log n} + k}{n} \Big).$$

Recall that distribution of each pointer z_j is uniform over its support, i.e., over $\mathcal{X}$ if j is even, and over $\mathcal{Y}$ if j is odd. Intuitively speaking, Lemma 6.4.3 states that if communication cost of a protocol is "small", i.e., is $o(n^2/k^2)$, then even after communicating the messages in the first j phases of the protocol, distribution of z_j is still "close" to being uniform. In other words, the first j phases of the protocol do not reveal "any useful information" about z_j. This in particular implies that at the end of the protocol, i.e., at the end of phase k, the target pointer z_k is still uniform and π_{HPC} should not be able to find it. We first formalize this inution and use it to prove Theorem 6.6 and then present a proof of Lemma 6.4.3 which is the heart of the argument.

Proof of Theorem 6.6 (assuming Lemma 6.4.3). The $\Omega(n)$ term in the lower bound trivially follows from the $\Omega(n)$ lower bound for set intersection (e.g. Theorem 6.5 with constant ε). In the following we prove the first (and the main) term. Note that for this purpose, we can assume $k = o(\sqrt{n})$ as otherwise the dominant term would already be the second term.

Let π_{HPC} be any deterministic protocol for HPC_k for $k = o(\sqrt{n})$ with communication cost $\mathsf{CC}_{\mathrm{dist}_{\mathsf{HPC}}}(\pi_{\mathsf{HPC}}) = o(n^2/k^2)$. Recall that $\mathrm{dist}(\mathsf{Z}_k) = \mathcal{U}_{\mathcal{X}}$ if k is even and $\mathrm{dist}(\mathsf{Z}_k) = \mathcal{U}_{\mathcal{Y}}$ if k is odd. Let us assume by symmetry that k is even. By Lemma 6.4.3, we have,

$$\begin{aligned}
\mathop{\mathbb{E}}_{(E_k,\Pi_k)}\Big[\|\mathrm{dist}(\mathsf{Z}_k \mid E_k,\Pi_k) - \mathcal{U}_{\mathcal{X}}\|_{tvd}\Big] &\le k\cdot c\cdot\Big(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}}) + k\cdot\log n} + k}{n}\Big)\\
&= k\cdot c\cdot\left(o(\frac{1}{k}) + o(\frac{\sqrt{\log n}}{n^{3/4}}) + o(\frac{k}{n})\right)\\
&= o(\frac{k}{k}) + o(\frac{k\cdot\sqrt{\log n}}{n^{3/4}}) + o(\frac{k^2}{n}) = o(1), \qquad (6.12)
\end{aligned}$$

as c is an absolute constant.

On the other hand, (E_k, Π_k) contains the whole transcript Π of the protocol and hence the output of the protocol π_{HPC} is fixed conditioned on (E_k, Π_k). We use $O(E_k, \Pi_k)$ to denote

this output. We have,

$$
\begin{aligned}
&\Pr_{(E_k,\Pi_k)}\left(\pi_{\mathsf{HPC}} \text{ is correct}\right)\\
&= \mathop{\mathbb{E}}_{(E_k,\Pi_k)} \Pr_{\mathsf{Z}_k|(E_k,\Pi_k)}\left(\mathsf{Z}_k = O(E_k,\Pi_k)\right)\\
&\underset{\text{Fact 2.4.6}}{\leq} \mathop{\mathbb{E}}_{(E_k,\Pi_k)}\left[\Pr_{\mathsf{Z}_k\sim\mathcal{U}_{\mathcal{X}}}\left(\mathsf{Z}_k = O(E_k,\Pi_k)\right) + \|\mathrm{dist}(\mathsf{Z}_k \mid E_k,\Pi_k) - \mathcal{U}_{\mathcal{X}}\|_{tvd}\right]\\
&\leq \frac{1}{n} + \mathop{\mathbb{E}}_{(E_k,\Pi_k)}\left[\|\mathrm{dist}(\mathsf{Z}_k \mid E_k,\Pi_k) - \mathcal{U}_{\mathcal{X}}\|_{tvd}\right] \underset{\text{Eq (6.12)}}{\leq} \frac{1}{n} + o(1).
\end{aligned}
$$

Hence, π_{HPC} cannot output the correct solution with at least a constant probability of success, proving the lower bound for deterministic algorithms.

To finalize, we can extend this (distributional) lower bound to randomized protocols by the easy direction of Yao's minimax principle [252], namely by an averaging argument that picks the "best" choice for randomness of the protocol. This concludes the proof. □

Proof of Lemma 6.4.3

The following is a restatement of Lemma 6.4.3.

Lemma (Restatement of Lemma 6.4.3)**.** *Let* $\mathsf{CC}(\pi_{\mathsf{HPC}}) := \mathsf{CC}_{\mathrm{dist}_{\mathsf{HPC}}}(\pi_{\mathsf{HPC}})$. *There exists an absolute constant* $c > 0$ *such that for all* $j \in [k]$*:*

$$
\mathop{\mathbb{E}}_{(E_j,\Pi_j)}\left[\|\mathrm{dist}(\mathsf{Z}_j \mid E_j,\Pi_j) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd}\right] \leq j \cdot c \cdot \left(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}}) + k \cdot \log n} + k}{n}\right).
$$

The proof of Lemma 6.4.3 consists of two main steps. We first show that finding the target element of a *uniformly at random* chosen instance of Set-Int (as opposed to the instance corresponding to any particular pointer) in HPC is not possible unless we make a large communication. Then, we prove inductively that in each phase j, the distribution of the pointer z_j is close to uniform and hence by the argument in the first step, we should not be able to find the target element t_{z_j} associated with z_j and use this to finalize the proof. The following lemma captures the first part.

Lemma 6.4.4. *There exists an absolute constant $c > 0$ such that for any $j \in [k]$,*

$$\mathop{\mathbb{E}}_{(E_j,\Pi_j)} \mathop{\mathbb{E}}_{x\sim\mathcal{U}_\mathcal{X}} [\|\mathrm{dist}(\mathsf{T}_x \mid E_j, \Pi_j) - \mathrm{dist}(\mathsf{T}_x)\|_{tvd}] \le c \cdot \Big(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}}) + j \cdot \log n} + j}{n}\Big),$$

$$\mathop{\mathbb{E}}_{(E_j,\Pi_j)} \mathop{\mathbb{E}}_{y\sim\mathcal{U}_\mathcal{Y}} [\|\mathrm{dist}(\mathsf{T}_y \mid E_j, \Pi_j) - \mathrm{dist}(\mathsf{T}_y)\|_{tvd}] \le c \cdot \Big(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}}) + j \cdot \log n} + j}{n}\Big).$$

The proof of this lemma is based on a direct-sum style argument combined with Theorem 6.5. For intuition, consider a protocol that uses $o(n^2)$ communication in its first j phases and assume by way of contradiction that it can reduce the LHS of one of the equations in Lemma 6.4.4 by $\Omega(1)$. Using a direct-sum style argument, we can then argue that the transcript of the first j phases of this protocol only reveal $o(n)$ bits of information about a uniformly at random chosen instance (A_x, B_x) of Set-Int but is enough to $\Omega(1)$-solve the instance (A_x, B_x) (according to Definition 6.3.1), which is in contradiction with our bounds in Theorem 6.5. Note that in this discussion, for the sake of simplicity, we neglected the role of extra conditioning on $Z^{<j}$ in E_j in the LHS of equations; handling this extra conditioning results in the extra additive factor in RHS.

Proof of Lemma 6.4.4. We only prove the first equation; the second one can be proven analogously. Suppose towards a contradiction that this equation does not hold. We use π_{HPC} to design a protocol π_{SI} that can ε-solve the Set-Int problem (A_x, B_x) for a uniformly at random chosen $x \in \mathcal{X}$ and appropriately chosen $\varepsilon \in (0, 1)$ to be determined later (see Definition 6.3.1 for the notion of ε-solve).

Protocol π_{SI}: The protocol for ε-solving Set-Int using a protocol π_{HPC} for HPC_k.

Input: An instance $(A, B) \sim \mathrm{dist}_{\mathsf{SI}}$ over the universe $\mathcal{Y}$.

1. **Sampling the instance.** Alice and Bob create an instance $(\boldsymbol{A}, \boldsymbol{B}, \overline{C}, \boldsymbol{D})$ of HPC_k as follows (see Figure 6.3 below for an illustration):
   (a) Using public coins, Alice and Bob sample an index $i \in [n]$ uniformly at random,

and Alice sets $A_{x_i} = A$ and Bob sets $B_{x_i} = B$ using their given inputs in Set-Int.

(b) Using public coins, Alice and Bob sample A_{x_j} and B_{x_k} from $\text{dist}_{\mathsf{SI}}$ for all $j < i < k$.

(c) Using private coins, Alice samples A_{x_k} for $k > i$ such that $(A_{x_k}, B_{x_k}) \sim \text{dist}_{\mathsf{SI}}$. Similarly Bob samples B_{x_j} for $j < i$. This completes construction of $(\boldsymbol{A}, \boldsymbol{B})$.

(d) Using public coins, Alice and Bob sample $(\overline{C}, \boldsymbol{D})$ completely from $\text{dist}_{\mathsf{HPC}}$ (this is possible by Observation 6.4.1 as $(\boldsymbol{A}, \boldsymbol{B}) \perp (\overline{C}, \boldsymbol{D})$).

2. **Computing the answer.** Alice and Bob first check whether x_i belongs to $z_0, z_1, \ldots, z_{j-1}$ or not. To do so, they start computing these pointers using the fact that for any underlying instance $(A_x, B_x) \in (\boldsymbol{A}, \boldsymbol{B}) \setminus (A_{x_i}, B_{x_i})$ either Alice or Bob knows the entire instance. They terminate the protocol if ever x_i belongs to one of the pointers computed so far. We use Π^* to denote the transcript of the protocol in this step (which is either $z_1, \ldots, z_{j-1}$ or some prefix of it ending in x_i).
3. Next, Alice and Bob run the protocol π_{HPC} on the instance $(\boldsymbol{A}, \boldsymbol{B}, \overline{C}, \boldsymbol{D})$ until its j-th phase by Alice playing P_A, Bob playing P_B, and both Alice and Bob simulating P_C and P_D with no communication (this is possible as both Alice and Bob know $(\overline{C}, \boldsymbol{D})$ entirely).
4. The players return $\Pi_{\mathsf{SI}} := (\Pi_1, \ldots, \Pi_j, \Pi^*)$.

Similar to the case of the sampling in protocol π_{PI} in Section 6.3, here also the public-private randomness sampling of the instance of HPC inside π_{SI} is only for the sake of the information theoretic arguments; for the rest of the analysis, we only care that the distribution of the instances of HPC sampled in π_{SI} is $\text{dist}_{\mathsf{HPC}}$. We first determine the parameter ε for which π_{SI} ε-solves Set-Int.

Claim 6.4.5. *π_{SI} ε-solves Set-Int on $\text{dist}_{\mathsf{SI}}$ for*

$$\varepsilon \geq \mathop{\mathbb{E}}_{(E_j, \Pi_j)} \mathop{\mathbb{E}}_{x \sim \mathcal{U}_{\mathcal{X}}} \Big[\|\text{dist}(\mathsf{T}_x \mid E_j, \Pi_j) - \text{dist}(\mathsf{T}_x)\|_{tvd} \Big] - \frac{j}{n},$$

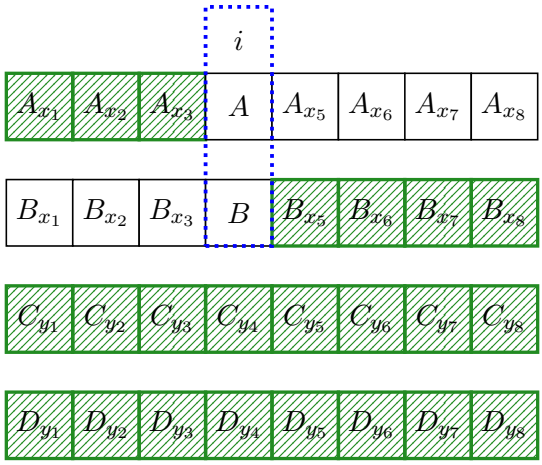


Figure 6.3: Illustration of the process of sampling of instances of HPC in π_{SI} for $n = 8$. In this example, $i = 4$ and hence $(A_{x_4}, B_{x_4}) = (A, B)$ and the players sample $\{A_{x_1}, A_{x_2}, A_{x_3}, B_{x_5}, B_{x_6}, B_{x_7}, B_{x_8}\}$ as well as the entire $\overline{C}$ and $\boldsymbol{D}$ using public randomness. Then, Alice samples $\{A_{x_5}, A_{x_6}, A_{x_7}, A_{x_8}\}$ and Bob samples $\{B_{x_1}, B_{x_2}, B_{x_3}\}$ using private randomness, respectively.

where $(E_j, \Pi_j, \mathsf{T}_x)$ *are distributed according to* $\mathrm{dist}_{\mathsf{HPC}}$.

Proof. By Definition 6.3.1, π_{SI} ε-solves Set-Int for $\varepsilon := \mathbb{E}_{\Pi_{\mathsf{SI}}}\left[\|\mathrm{dist}(\mathsf{T} \mid \Pi_{\mathsf{SI}}) - \mathrm{dist}(\mathsf{T})\|_{tvd}\right]$. We thus bound the RHS of this equation. We have,

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{\Pi_{\mathsf{SI}}}\left[\|\mathrm{dist}(\mathsf{T} \mid \Pi_{\mathsf{SI}}) - \mathrm{dist}(\mathsf{T})\|_{tvd}\right] \\
&= \mathop{\mathbb{E}}_{(E_j,\Pi_j,\Pi_{\mathsf{SI}},i)}\left[\|\mathrm{dist}(\mathsf{T}_{x_i} \mid \Pi_{\mathsf{SI}}) - \mathrm{dist}(\mathsf{T}_{x_i})\|_{tvd}\right] && (\text{as } \mathsf{T} = \mathsf{T}_{x_i} \text{ for } \mathsf{I} = i) \\
&= \mathop{\mathbb{E}}_{(E_j,\Pi_j)} \mathop{\mathbb{E}}_{i} \mathop{\mathbb{E}}_{\Pi_{\mathsf{SI}}|(E_j,\Pi_j,i)}\left[\|\mathrm{dist}(\mathsf{T}_{x_i} \mid \Pi_{\mathsf{SI}}) - \mathrm{dist}(\mathsf{T}_{x_i})\|_{tvd}\right] \\
& && (\text{as } (\mathsf{E}_j, \Pi_j) \perp \mathsf{I}) \\
&= \mathop{\mathbb{E}}_{(E_j,\Pi_j)}\left[\sum_{i=1}^{n} \frac{1}{n} \mathop{\mathbb{E}}_{\Pi_{\mathsf{SI}}|(E_j,\Pi_j,i)}\left[\|\mathrm{dist}(\mathsf{T}_{x_i} \mid \Pi_{\mathsf{SI}}) - \mathrm{dist}(\mathsf{T}_{x_i})\|_{tvd}\right]\right] \\
& && (\text{distribution of } i \text{ is uniform over } [n]) \\
&= \mathop{\mathbb{E}}_{(E_j,\Pi_j)}\Big[\sum_{x_i \in Z^{<j}} \frac{1}{n} \cdot \|\mathrm{dist}(\mathsf{T}_{x_i} \mid Z^{<j'}) - \mathrm{dist}(\mathsf{T}_{x_i})\|_{tvd}
\end{aligned}
$$

$$+ \sum_{x_i \notin Z^{<j}} \frac{1}{n} \cdot \|\mathrm{dist}(\mathsf{T}_{x_i} \mid E_j, \Pi_j) - \mathrm{dist}(\mathsf{T}_{x_i})\|_{tvd}\Big]$$

$$(\Pi^* := Z^{<j'} \text{ for some } j' < j-1 \text{ when } x_i \in Z^{<j} \text{ and is otherwise equal to } E_j, \Pi_j))$$

$$= \mathop{\mathbb{E}}_{(E_j,\Pi_j)} \left[\sum_{x_i \notin Z^{<j}} \frac{1}{n} \cdot \|\mathrm{dist}(\mathsf{T}_{x_i} \mid E_j, \Pi_j) - \mathrm{dist}(\mathsf{T}_{x_i})\|_{tvd} \right]$$

$$(\mathsf{T}_{x_i} \perp \Pi^* \text{ and so } \|\mathrm{dist}(\mathsf{T}_{x_i} \mid Z^{<j'}) - \mathrm{dist}(\mathsf{T}_{x_i})\|_{tvd} = \|\mathrm{dist}(\mathsf{T}_{x_i}) - \mathrm{dist}(\mathsf{T}_{x_i})\|_{tvd} = 0)$$

$$\geq \mathop{\mathbb{E}}_{(E_j,\Pi_j)} \mathop{\mathbb{E}}_{i} \Big[\|\mathrm{dist}(\mathsf{T}_{x_i} \mid E_j, \Pi_j) - \mathrm{dist}(\mathsf{T}_{x_i})\|_{tvd}\Big] - \frac{j}{n}.$$

(as total variation distance is bounded by one $\left|Z^{<j}\right| = j$)

Replacing x_i for i chosen randomly from $[n]$ above by $x \sim \mathcal{U}_{\mathcal{X}}$ concludes the proof. □

The RHS in Claim 6.4.5 is the quantity we aim to bound in this lemma (minus the extra additive j/n term). To do so, we are going to bound the internal information cost of π_{SI} by the communication cost of π_{HPC} in the following claim and then use Theorem 6.5 to relate this quantity to ε.

Claim 6.4.6. $\mathsf{IC}_{\mathrm{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}}) = O\Big(\frac{\mathsf{CC}(\pi_{\mathsf{HPC}})}{n} + \frac{j \cdot \log n}{n}\Big)$.

Proof. For any $i \in [n]$, define $\boldsymbol{A}^{<i} := \{A_{x_1}, \ldots, A_{x_{i-1}}\}$, $\boldsymbol{B}^{>i} := \{B_{x_{i+1}}, \ldots, B_{x_n}\}$. Recall that the internal information cost of π_{SI} is $\mathsf{IC}_{\mathrm{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}}) := \mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}) + \mathbb{I}(\mathsf{B}\,;\Pi_{\mathsf{SI}} \mid \mathsf{A})$. In the following, we focus on bounding the first term. The second term can be bounded exactly the same by symmetry.

As $(\mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \overline{C}, \boldsymbol{D})$ is sampled via public randomness in π_{SI}, by Proposition 2.5.3,

$$\mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}) = \mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \overline{C}, \boldsymbol{D}) \leq \mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}).$$

The inequality is by Proposition 2.4.3 as we now show $\mathsf{A} \perp (\overline{C}, \boldsymbol{D}) \mid \Pi_{\mathsf{SI}}, \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}$ (and hence conditioning on $(\overline{C}, \boldsymbol{D})$ can only decrease the mutual information). This is because $\mathsf{A} \perp (\overline{C}, \boldsymbol{D}) \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}$ by Observation 6.4.1 and Π_{SI} is transcript of a deterministic

protocol plus $z_1, \ldots, z_j$ obtained deterministically and hence we can apply Proposition 6.4.2.

Define a random variable $\Theta \in \{0, 1\}$ where $\Theta = 1$ iff in Line (2) of protocol π_{SI}, we terminate the protocol. In other words $\Theta = 1$ iff $x_i \in Z^{<j}$. Since $\mathsf{A} \perp \Theta \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}$, further conditioning on Θ can only increase the mutual information term above by Proposition 2.4.2, hence,

$$\begin{aligned}
\mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}) &\leq \mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta) \\
&= \frac{n-j}{n} \cdot \mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta = 0) + \frac{j}{n} \cdot \mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta = 1) \\
&\leq \frac{n-j}{n} \cdot \mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta = 0), \qquad (6.13)
\end{aligned}$$

since conditioned on $\Theta = 1$, the protocol Π_{SI} is simple some prefix of $\mathsf{Z}^{<j}$ and is hence independent of the input (A, B) and carries no information about A (see Fact 2.4.1-(2)). We now further bound the RHS of Eq (6.13). When $\Theta = 0$, $\Pi_{\mathsf{SI}} = (\mathsf{Z}^{<j}, \Pi_1, \ldots, \Pi_j) = (\mathsf{E}^{<j}, \Pi_j)$. Hence, we can write,

$$\begin{aligned}
\mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta = 0) &\leq \mathbb{I}(\mathsf{A}\,;\mathsf{E}_j, \Pi_j \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta = 0) \\
&= \mathbb{I}(\mathsf{A}\,;\mathsf{Z}^{<j} \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta = 0) \\
&\quad + \mathbb{I}(\mathsf{A}\,;\Pi^{<j}, \Pi_j \mid \mathsf{Z}^{<j}, \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta = 0) \\
&\quad \text{(by chain rule in Fact 2.4.1-(6) and since } \mathsf{E}_j = (\Pi^{<j}, \mathsf{Z}^{<j})) \\
&\leq \mathbb{I}(\mathsf{A}\,;\Pi \mid \mathsf{Z}^{<j}, \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta = 0),
\end{aligned}$$

as $\mathsf{A} \perp \mathsf{Z}^{<j} \mid \Theta = 0$ (and other variables) and hence the first term is zero, and in the second term Π contains $\Pi^{<j}, \Pi_j$ (plus potentially other terms) and so having Π in instead can only increase the information. By further expanding the conditional information term above,

$$\begin{aligned}
&\mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}} \mid \mathsf{B}, \mathsf{I}, \boldsymbol{A}^{<\mathsf{I}}, \boldsymbol{B}^{>\mathsf{I}}, \Theta = 0) \\
&\qquad \leq \mathop{\mathbb{E}}_{(Z^{<j}, i) \mid \Theta = 0} \left[\mathbb{I}(\mathsf{A}\,;\Pi \mid \mathsf{B}, \boldsymbol{A}^{<i}, \boldsymbol{B}^{>i}, \mathsf{I} = i, \mathsf{Z}^{<j} = Z^{<j}, \Theta = 0)\right]
\end{aligned}$$

$$= \mathop{\mathbb{E}}_{Z^{<j}|\Theta=0}\left[\sum_{\substack{i=1\\ i\notin Z^{<j}}}^{n} \frac{1}{n-j}\mathbb{I}(\mathsf{A}_{x_i};\Pi \mid \mathsf{B}_{x_i}, \boldsymbol{A}^{<i}, \boldsymbol{B}^{>i}, \mathsf{I}=i, \mathsf{Z}^{<j}=Z^{<j}, \Theta=0)\right]$$

(conditioned on $\Theta=0$, i is chosen uniformly at random from $Z^{<j}$; also $(\mathsf{A},\mathsf{B})=(\mathsf{A}_{x_i},\mathsf{B}_{x_i})$)

$$= \mathop{\mathbb{E}}_{Z^{<j}|\Theta=0}\left[\sum_{i\notin Z^{<j}} \frac{1}{n-j}\cdot\mathbb{I}(\mathsf{A}_{x_i};\Pi \mid \mathsf{B}_{x_i}, \boldsymbol{A}^{<i}, \boldsymbol{B}^{>i}, \mathsf{Z}^{<j}=Z^{<j}, \Theta=0)\right]$$

(we dropped the conditioning on $\mathsf{I}=i$ as all remaining variables are independent of this event)

$$= \mathop{\mathbb{E}}_{Z^{<j}|\Theta=0}\left[\sum_{i\notin Z^{<j}} \frac{1}{n-j}\cdot\mathbb{I}(\mathsf{A}_{x_i};\Pi \mid \boldsymbol{A}^{<i}, \boldsymbol{B}, \mathsf{Z}^{<j}=Z^{<j}, \Theta=0)\right]$$

(as $\mathsf{A}_{x_i} \perp \boldsymbol{B}^{<i} \mid \mathsf{B}_{x_i}, \boldsymbol{A}^{<i}$ by Observation 6.4.1 and hence we can apply Proposition 2.4.2)

$$\le \mathop{\mathbb{E}}_{Z^{<j}|\Theta=0}\left[\sum_{i=1}^{n} \frac{1}{n-j}\cdot\mathbb{I}(\mathsf{A}_{x_i};\Pi \mid \boldsymbol{A}^{<i}, \boldsymbol{B}, \mathsf{Z}^{<j}=Z^{<j}, \Theta=0)\right]$$

(mutual information is non-negative by Fact 2.4.1-(2) and so we can add the terms in $Z^{<j}$ as well)

$$= \mathop{\mathbb{E}}_{Z^{<j}|\Theta=0}\left[\sum_{i=1}^{n} \frac{1}{n-j}\cdot\mathbb{I}(\mathsf{A}_{x_i};\Pi \mid \boldsymbol{A}^{<i}, \boldsymbol{B}, \mathsf{Z}^{<j}=Z^{<j}, \Theta=0)\right]$$

$$= \frac{1}{n-j}\cdot\mathop{\mathbb{E}}_{Z^{<j}|\Theta=0}\left[\mathbb{I}(\boldsymbol{A};\Pi \mid \boldsymbol{B}, \mathsf{Z}^{<j}=Z^{<j}, \Theta=0)\right]$$

(by chain rule in Fact 2.4.1-(6))

$$= \frac{1}{n-j}\cdot\mathbb{I}(\boldsymbol{A};\Pi \mid \boldsymbol{B}, \mathsf{Z}^{<j}, \Theta=0)$$ (by Proposition 2.4.4)

$$\le \frac{1}{n-j}\cdot\left(\mathbb{I}(\boldsymbol{A};\Pi \mid \boldsymbol{B}, \Theta=0)+\mathbb{H}(\mathsf{Z}^{<j})\right)$$

$$= \frac{1}{n-j}\cdot\left(\mathbb{I}(\boldsymbol{A};\Pi \mid \boldsymbol{B})+\mathbb{H}(\mathsf{Z}^{<j})\right)$$

(transcript of the protocol π_{HPC} (namely Π) on input $(\boldsymbol{A},\boldsymbol{B})$ is independent of Θ)

$$\le \frac{1}{n-j}\cdot\left(\mathbb{H}(\Pi)+\mathbb{H}(\mathsf{Z}^{<j})\right) \le \frac{\mathsf{CC}(\pi_{\mathsf{HPC}})}{n-j}+\frac{j\cdot\log n}{n-j}.$$

(by sub-additivity of entropy (Fact 2.4.1-(4)) and Fact 2.4.1-(1))

By plugging in this bound in Eq (6.13), we have that,

$$\mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}}\mid \mathsf{B}) \leq \frac{n-j}{n}\cdot\left(\frac{\mathsf{CC}(\pi_{\mathsf{HPC}})}{n-j}+\frac{j\cdot\log n}{n-j}\right)=\frac{\mathsf{CC}(\pi_{\mathsf{HPC}})}{n}+\frac{j\cdot\log n}{n}.$$

By symmetry, we can also prove the same bound on $\mathbb{I}(\mathsf{B}\,;\Pi_{\mathsf{SI}}\mid \mathsf{A})$. As such, we have,

$$\mathbb{I}(\mathsf{A}\,;\Pi_{\mathsf{SI}}\mid \mathsf{B})+\mathbb{I}(\mathsf{B}\,;\Pi_{\mathsf{SI}}\mid \mathsf{A}) \leq 2\cdot\left(\frac{\mathsf{CC}(\pi_{\mathsf{HPC}})}{n}+\frac{j\cdot\log n}{n}\right).$$

We shall note that strictly speaking the factor 2 above is not needed (similar to the proof of Proposition 2.5.4) but as this factor is anyway suppressed through O-notation later in the proof, the above bound suffices for our purpose. □

Now by Claim 6.4.6, we have that

$$\mathsf{IC}_{\mathrm{dist}_{\mathsf{SI}}}(\pi_{\mathsf{SI}})=O\Big(\frac{\mathsf{CC}(\pi_{\mathsf{HPC}})}{n}+\frac{j\cdot\log n}{n}\Big).$$

Combined with Theorem 6.5, this implies that π_{SI} can only ε-solves Set-Int for parameter ε such that

$$\varepsilon^2\cdot n=O\Big(\frac{\mathsf{CC}(\pi_{\mathsf{HPC}})}{n}+\frac{j\cdot\log n}{n}\Big)\implies \varepsilon=O\Big(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}})+j\cdot\log n}}{n}\Big).$$

On the other hand, by Claim 6.4.5, we know that

$$\varepsilon\geq \mathop{\mathbb{E}}_{(E_j,\Pi_j)}\mathop{\mathbb{E}}_{x\sim\mathcal{U}_{\mathcal{X}}}\Big[\|\mathrm{dist}(\mathsf{T}_x\mid E_j,\Pi_j)-\mathrm{dist}(\mathsf{T}_x)\|_{tvd}\Big]-\frac{j}{n}.$$

which implies

$$\mathop{\mathbb{E}}_{x\sim\mathcal{U}_{\mathcal{X}}}\Big[\|\mathrm{dist}(\mathsf{T}_x\mid E_j,\Pi_j)-\mathrm{dist}(\mathsf{T}_x)\|_{tvd}\Big]=O\Big(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}})+j\cdot\log n}+j}{n}\Big).$$

This concludes the proof. □

Before getting to the proof of Lemma 6.4.3, we also need the following simple claim based on the rectangle property of the protocol π_{HPC}.

Claim 6.4.7. *For any $j \in [k]$ and choice of (E_j, Π_j), $\mathrm{dist}(\mathsf{Z}_j \mid E_j, \Pi_j) = \mathrm{dist}(\mathsf{Z}_j \mid E_j)$.*

Proof. This is because for any $j \in [k]$, $\mathsf{Z}_j \perp \mathsf{\Pi}_j \mid E_j$: Conditioned on $\mathsf{E}_j = E_j = (Z^{<j}, \Pi^{<j})$, $\mathsf{\Pi}_j$ is only a function of $(\boldsymbol{A}, \boldsymbol{B})$ if j is even and a function of $(\overline{C}, \boldsymbol{D})$ if j is odd. On the other hand, Z_j is only a function of $(\boldsymbol{A}, \boldsymbol{B})$ if j is odd and a function of $(\overline{C}, \boldsymbol{D})$ if j is even. Finally, by Observation 6.4.1, $(\boldsymbol{A}, \boldsymbol{B}) \perp (\overline{C}, \boldsymbol{D})$ and this continues to hold even when we condition on E_j by the rectangle property of the protocol π_{HPC}; hence the claim follows. □

We are now finally ready to prove Lemma 6.4.3.

Proof of Lemma 6.4.3. Let c be the constant in Lemma 6.4.4. We prove Lemma 6.4.3 by induction. We start with the proof of the base case for $j = 1$ and then prove the inductive step.

Base case. Recall that we defined $E_1 = z_0$ which is deterministically fixed. This, together with Claim 6.4.7, implies that $\mathrm{dist}(\mathsf{Z}_1 \mid E_1, \Pi_1) = \mathrm{dist}(\mathsf{Z}_1)$, which finalizes proof of the base case.

Induction step. Let us now prove the lemma inductively for $j > 1$. We have,

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{(E_j, \Pi_j)} \Big[\|\mathrm{dist}(\mathsf{Z}_j \mid E_j, \Pi_j) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd} \Big] \\
&\underset{\text{Claim 6.4.7}}{=} \mathop{\mathbb{E}}_{E_j} \Big[\|\mathrm{dist}(\mathsf{Z}_j \mid E_j) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd} \Big] \\
&= \mathop{\mathbb{E}}_{(Z^{<j}, \Pi^{<j})} \Big[\|\mathrm{dist}(\mathsf{Z}_j \mid Z^{<j}, \Pi^{<j}) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd} \Big] \quad \text{(by definition of } E_j := (Z^{<j}, \Pi^{<j})) \\
&= \mathop{\mathbb{E}}_{(Z^{<j}, \Pi^{<j})} \Big[\|\mathrm{dist}(\mathsf{T}_{z_{j-1}} \mid Z^{<j-1}, z_{j-1}, \Pi^{<j}) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd} \Big]. \\
&\qquad\qquad \text{(by definition, the pointer } \mathsf{Z}_j = \mathsf{T}_{z_{j-1}})
\end{aligned}
$$

We can write the RHS above as:

$$\mathop{\mathbb{E}}_{(E_j,\Pi_j)}\Big[\|\mathrm{dist}(\mathsf{Z}_j \mid E_j,\Pi_j) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd}\Big]$$
$$= \mathop{\mathbb{E}}_{(Z^{<j-1},\Pi^{<j})} \mathop{\mathbb{E}}_{z_{j-1}\sim \mathsf{Z}_{j-1}|(Z^{<j-1},\Pi^{<j})} \Big[\|\mathrm{dist}(\mathsf{T}_{z_{j-1}} \mid Z^{<j-1},\Pi^{<j}) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd}\Big].$$

This is because $\mathsf{T}_{z_{j-1}} \perp (\mathsf{Z}_{j-1} = z_{j-1}) \mid Z^{<j-1},\Pi^{<j}$: if $j-1$ is odd, $\mathsf{T}_{z_{j-1}}$ is a function of $(\overline{C},\boldsymbol{D})$ and if $j-1$ is even, $\mathsf{T}_{z_{j-1}}$ is a function of $(\boldsymbol{A},\boldsymbol{B})$. On the other hand, if $j-1$ is odd, then Z_{j-1} is a function of $(\boldsymbol{A},\boldsymbol{B})$ and if even, then Z_{j-1} is a function of $(\overline{C},\boldsymbol{D})$. Finally, by Proposition 6.4.2, $(\boldsymbol{A},\boldsymbol{B}) \perp (\boldsymbol{B},\boldsymbol{D}) \mid \Pi^{<j}$, proving the conditional independence.

Now notice that distribution of z_{j-1} in the expectation-term above is $\mathrm{dist}(\mathsf{Z}_{j-1} \mid E_{j-1},\Pi_{j-1})$. By symmetry, let us assume $j-1$ is odd and hence $z_{j-1} \in \mathcal{Y}$. Using Fact 2.4.6 and since total variation distance is bounded by 1 always, we can upper bound RHS above with:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{(E_j,\Pi_j)}\Big[\|\mathrm{dist}(\mathsf{Z}_j \mid E_j,M_j) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd}\Big] \\
&\quad\le \mathop{\mathbb{E}}_{(Z^{<j-1},\Pi^{<j})}\Bigg[\mathop{\mathbb{E}}_{(z_{j-1}\sim\mathcal{U}_{\mathcal{Y}})}\Big[\|\mathrm{dist}(\mathsf{T}_{z_{j-1}} \mid Z^{<j-1},\Pi^{<j}) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd}\Big]\Bigg] \\
&\qquad\qquad + \mathop{\mathbb{E}}_{(Z^{<j-1},\Pi^{<j})}\left[\|\mathrm{dist}(\mathsf{Z}_{j-1} \mid E_{j-1},\Pi_{j-1}) - \mathcal{U}_{\mathcal{Y}}\|_{tvd}\right] \\
&\quad= \mathop{\mathbb{E}}_{(E_{j-1},\Pi_{j-1})} \mathop{\mathbb{E}}_{y\sim\mathcal{U}_{\mathcal{Y}}}\Big[\|\mathrm{dist}(\mathsf{T}_y \mid E_{j-1},\Pi_{j-1}) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd}\Big] \\
&\qquad\qquad + \mathop{\mathbb{E}}_{(E_{j-1},\Pi_{j-1})}\left[\|\mathrm{dist}(\mathsf{Z}_{j-1} \mid E_{j-1},\Pi_{j-1}) - \mathrm{dist}(\mathsf{Z}_{j-1})\|_{tvd}\right],
\end{aligned}$$

where in the first term above we only changed the name of variable z_{j-1} to y and in the second term we used $\mathrm{dist}(\mathsf{Z}_{j-1}) = \mathcal{U}_{\mathcal{Y}}$. By Lemma 6.4.4, we can bound the first term and by induction, we can bound the second one. Hence,

$$\begin{aligned}
\mathop{\mathbb{E}}_{(E_j,\Pi_j)}\Big[\|\mathrm{dist}(\mathsf{Z}_j \mid E_j,\Pi_j) - \mathrm{dist}(\mathsf{Z}_j)\|_{tvd}\Big] &\le c\cdot\left(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}}) + j\cdot\log n} + j}{n}\right) \\
&\quad + (j-1)\cdot c\cdot\left(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}}) + k\cdot\log n} + k}{n}\right)
\end{aligned}$$

$$\leq j \cdot c \cdot \Big(\frac{\sqrt{\mathsf{CC}(\pi_{\mathsf{HPC}}) + k \cdot \log n} + k}{n}\Big).$$

(where we replaced $j \leq k$ by k in the first term)

This concludes the proof. □

6.5. Graph Streaming Lower Bounds

We now present our graph streaming lower bounds using reductions from the hidden-pointer chasing problem. In particular, we prove the following two results in this section.

Theorem 6.7 (Formalizing Theorem 6.2)**.** *For any integer $p \geq 1$, any p-pass streaming algorithm that with a constant probability outputs the minimum s-t cut value in a weighted directed or undirected graph $G(V, E, w)$ requires $\Omega(n^2/p^5)$ bits of space.*

By max-flow min-cut theorem, Theorem 6.7 also holds for streaming algorithms that can compute the value of maximum s-t flow in a capacitated graph (directed or undirected).

Theorem 6.8 (Formalizing Theorem 6.3)**.** *For any integer $p \geq 1$, any p-pass streaming algorithm that with a constant probability outputs the lexicographically-first maximal independent set of an undirected graph $G(V, E)$ requires $\Omega(n^2/p^5)$ bits of space.*

We prove Theorem 6.7 and Theorem 6.8 in Section 6.5.1 and Section 6.5.2, respectively.

6.5.1. Weighted Minimum s-t Cut Problem

We prove Theorem 6.7 by a reduction from our hidden-pointer chasing (HPC) problem. We first give the lower bound for directed graphs and then show how to extend it using standard techniques to undirected graphs.

We turn an instance $(\boldsymbol{A}, \boldsymbol{B}, \overline{C}, \boldsymbol{D})$ of HPC_k over universes $\mathcal{X}$ and $\mathcal{Y}$ of n elements, into a weighted directed graph $G(V, E, w)$. The reduction is as follows (see Figure 6.4 for an example):

- The vertex-set V of G is partitioned into $k+1$ layers $V_0, \ldots, V_k$ each of size n plus the

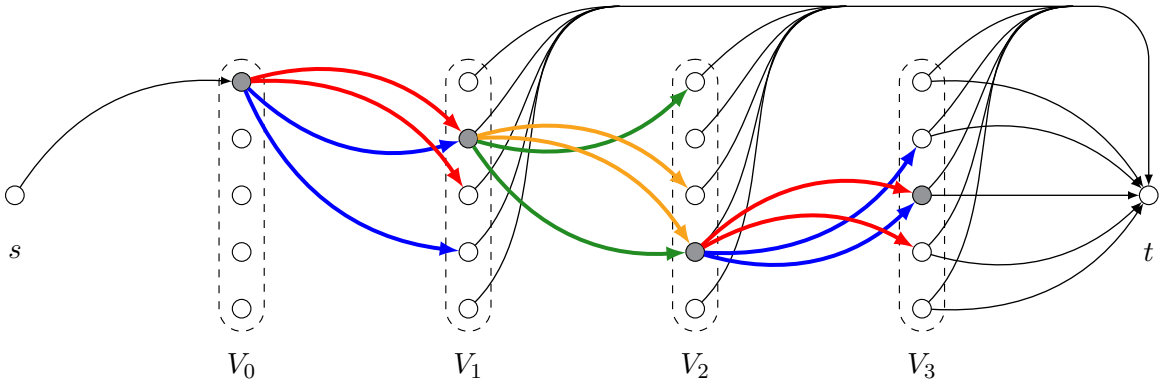


Figure 6.4: Illustration of the graph in the reduction for minimum s-t cut from HPC_3 with $n = 5$. The black (thin) edges form input-independent gadgets while blue, red , brown, and green (thick) edges depend on the inputs of P_A, P_B, P_C, and P_D, respectively. Marked nodes denote the vertices corresponding to pointers $z_0, \ldots, z_3$. The input-dependent edges incident on "non-pointer" vertices are omitted. This construction has parallel edges but Remark 6.5.5 shows how to remove them.

source and sink vertices s and t. We denote the i-th vertex in layer V_j by v_i^j.

- Define the following sequence of weights $w_0, w_1, \ldots, w_k$ where $w_j := (n+1)^{k+1-j}$ for all $j \in [k]$. Hence, $w_k = (n+1)$ and $w_j = (n+1) \cdot w_{j+1}$ for all $j < k$.

- The edge-set E of G contains the following input-independent edges.

  - source s is connected to v_1^0 with weight $w(s, v_1^0) = w_0$.

  - for $0 < j \leq k$, every vertex v_i^j in layer V_j is connected to sink t with weight $w(v_i^j, t) = w_j$.

  - any vertex v_i^k in layer V_k is connected to sink t with weight $w(v_i^k, t) = i - 1$ (notice that v_i^k also has another edge of weight w_k to t by the previous part).

- The edge-set E also contains the following input-dependent edges.

  - for all $i \in [n]$, if $A_{x_i} \in \boldsymbol{A}$ (resp. $B_{x_i} \in \boldsymbol{B}$) contains $y_{i'} \in \mathcal{Y}$, we connect v_i^j in layer V_j to $v_{i'}^{j+1}$ in layer V_{j+1} with weight $w(v_i^j, v_{i'}^{j+1}) = w_{j+1}$ for every *even* $0 \leq j < k$.[2]

[2] Note that we will add two edges between v_i^j and $v_{i'}^{j+1}$ iff $y_{i'} \in A_{x_i} \cap B_{x_i}$ and we will keep both copies

– for all $i \in [n]$, if $C_{y_i} \in \overline{C}$ (resp. $D_{y_i} \in \boldsymbol{D}$) contains $x_{i'} \in \mathcal{X}$, we connect v_i^j in layer V_j to $v_{i'}^{j+1}$ in layer V_{j+1} with weight $w(v_i^j, v_{i'}^{j+1}) = w_{j+1}$ for every *odd* $0 < j < k$.

This concludes the description of the weighted graph $G(V, E, w)$ in the reduction. It is straightforward to verify that this graph can be constructed from an instance $(\boldsymbol{A}, \boldsymbol{B}, \overline{C}, \boldsymbol{D})$ with no communication between the players. We now prove the following key lemma which establishes the correctness of the reduction.

Lemma 6.5.1. *Let w^* be the weight of a minimum s-t cut in graph G in the reduction. Let the pointer z_k be x_{i^*} (resp. y_{i^*}) if k is even (resp. odd). Then $i^* = (w^* \bmod (n+1)) + 1$.*

Proof. We prove this lemma by considering the maximum s-t flow in G and then use the duality of maximum flow and minimum cut to conclude the proof. For the flow problem, we assume that the capacity $c(e)$ of an edge $e = (u, v)$ in G is equal to the total weight of the edges (in w) that connect u to v (recall that G may have parallel edges; see Footnote 2).

We start with some definitions. Define u_j in layer V_j to be the vertex corresponding to the pointer z_j, namely, for all even (resp. odd) values of j, $u_j = v_i^j$ where $x_i = z_j$ (resp. $y_i = z_j$). Furthermore, let $\mathcal{P} := \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_k \cup \{P^*\}$ be a collection of flow paths defined as follows: For any $j \in [k]$, the set of paths $\mathcal{P}_j := \left\{(s, u_0, u_1, \ldots, u_{j-1}, v_i^j, t) \mid (u_{j-1}, v_i^j) \in E\right\}$ and each path in $\mathcal{P}_j$ carries w_j units of flow; moreover, $P^* = (s, u_0, u_1, \ldots, u_k, t)$ and carries $i^* - 1$ units of flow. See Figure 6.5 for an illustration.

We have the following auxiliary claim.

Claim 6.5.2. *For any $j \in [k]$, capacity of the edge $e = (u_{j-1}, u_j)$ is $c(e) = 2w_j$.*

Proof. Suppose $u_{j-1} = v_i^{j-1}$ and $u_j = v_{i'}^j$ and assume that j is odd; the even j case is symmetric. Since j is odd, $y_{i'}$ is contained in both A_{x_i} and B_{x_i}. Hence, there are two parallel edges from u_{j-1} to u_j each of weight w_j. So the capacity of (u_{j-1}, u_j) is $2w_j$. □

of these edges in G (see also Remark 6.5.5 on how to remove the parallel edges).

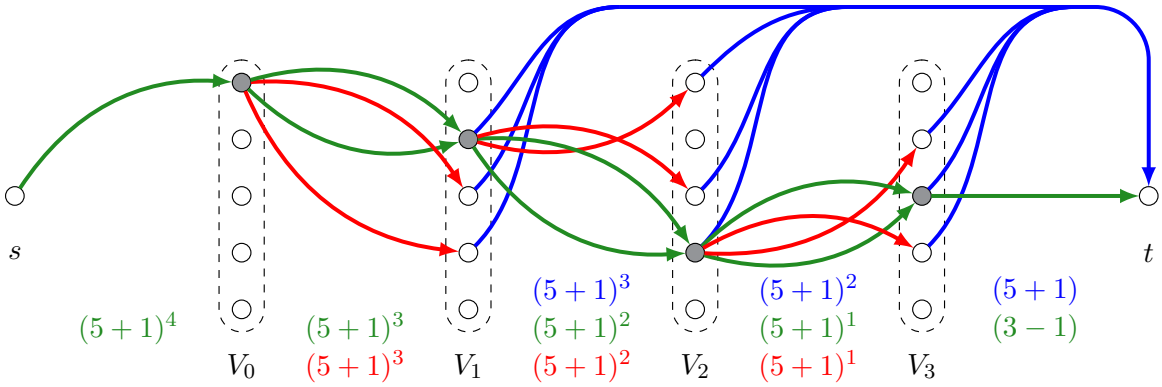


Figure 6.5: Illustration of the flow paths in $\mathcal{P}$ in the proof of Lemma 6.5.1 for $n = 5$ and $k = 3$. The green edges belong to P^* while red and blue edges are the edges that belong to a path in some $\mathcal{P}_j$ but not P^*. The numbers denote the value of the flow sent over each outgoing edge in the corresponding layer with the same color. The value of this flow mod $(n+1)$ is $(i^* - 1)$ where $i^* = 3$.

We claim that $\mathcal{P}$ gives a maximum flow in graph G. This proves the lemma as for all $j \in [k]$, the contribution of each path in $\mathcal{P}_j$ to the flow mod $(n+1)$ is 0. Hence P^* determines the value of the flow mod $(n+1)$ which is $(i^* - 1)$ and i^* encodes the pointer z_k. The proof consists of the following two claims that ensure feasibility and optimality of $\mathcal{P}$, respectively.

Claim 6.5.3. *$\mathcal{P}$ induces a feasible flow in $G(V, E, w)$ with capacity w_e on every edge $e \in E$.*

Proof. Since all the paths in $\mathcal{P}$ are s-t paths, for any vertex in $V \setminus \{s, t\}$, the amount of flow going in that vertex is equal to the amount of flow going out of it. Hence, the flow is preserved on all vertices in $V \setminus \{s, t\}$. It thus remains to prove that no edge is assigned a flow more than its capacity.

Any edge e *not* in P^* is contained in at most one path in $\mathcal{P}$. For paths in $\mathcal{P}_j$, these are edges (u_{j-1}, v_i^j) and (v_i^j, t) for some $j \in [k]$ and $i \in [n]$. The amount of flow on these paths is then equal to $w_j = w(v_i^j, t)$ by construction and hence the flow on these edges does not exceed their capacity.

We now prove the result for edges in P^*. First consider the edge (u_k, t). There are two paths in $\mathcal{P}$ that contain (u_k, t): the path P^* that carries $i^* - 1$ units of flow and the path in $\mathcal{P}_k$

that carries w_k units of flow. As $u_k = v^k_{i^*}$, the capacity of the edge (u_k, t) is also $w_k + (i^* - 1)$ (as there are two edges connecting $v^k_{i^*}$ to t with weights w_k and $(i^* - 1)$). Hence the flow on these edges also does not exceed their capacity.

We next prove that for every $j \in [k]$, there are at most $2w_j$ units of flow passing through (u_{j-1}, u_j). By Claim 6.5.2, this implies that the flow on these edges does not exceed capacity. The proof is by induction for $j = k$ down to $j = 0$ in this order, where the base case is (u_{k-1}, u_k). All the paths that contain this edge also contain (u_k, t), so there are $w_k + i^* - 1 < 2w_k$ units of flow passing through this edge by the previous part of the argument.

For the induction step, consider the flow paths that contain (u_{j-1}, u_j). There is exactly one path in $\mathcal{P}_j$ that contains this edge and that path carries w_j units of flow by definition. There are also at most $n - 1$ paths in $\mathcal{P}_{j+1}$ that contain (u_{j-1}, u_j) but do not contain (u_j, u_{j+1}). The total flow these paths are carrying is at most $(n - 1) \cdot w_{j+1}$. All other paths in $\mathcal{P}$ that contain (u_{j-1}, u_j) also contain (u_j, u_{j+1}) and hence by the induction hypothesis, these paths carry at most $2w_{j+1}$ units of flow. So the total flow going through (u_{j-1}, u_j) is at most $w_j + (n - 1)w_{j+1} + 2w_{j+1} \leq 2w_j$, proving the induction hypothesis.

Finally, consider the edge (s, u_0). There are at most $n - 1$ paths in $\mathcal{P}_1$ that contain (s, u_0) but not (u_0, u_1). The total flow passing through these paths is at most $(n-1) \cdot w_1$. All other paths in $\mathcal{P}$ contain (u_0, u_1); these paths carry at most $2w_1$ units of flow as we proved above by induction. So the total flow passing through (s, u_0) is at most $(n - 1) \cdot w_1 + 2w_1 = w_0$ which is equal to the capacity of (s, u_0). □

Claim 6.5.4. *There is no s-t path in the residual graph of G with respect to the flow paths in $\mathcal{P}$.*

Proof. We prove by induction that in the residual graph, s can only reach u_j in layer V_j (strictly speaking, we will prove that if some other vertex in V_j is reachable from s, then the path can only go through t, but in the end we will prove that t is not reachable from s).

The base case trivially holds as s only has an outgoing edge to a single vertex in V_0, namely, the vertex $v_1^0 = u_0$. Furthermore, the outgoing edges of vertices in V_0 do not belong to any flow path in $\mathcal{P}$. For the induction step, consider the layer V_{j+1}. By the induction hypothesis, s can only reach u_j in V_j. For any vertex v_i^{j+1} which is not u_{j+1}, if the edge (u_j, v_i^{j+1}) exists in G, then it is contained in a path in $\mathcal{P}_{j+1}$ which carries w_{j+1} units of flow. As the capacity of this edge is also w_{j+1}, the direction of this edge in the residual graph is from v_i^{j+1} to u_j. Moreover, no outgoing edge of v_i^{j+1} (except for the one going to t) is contained in any path in $\mathcal{P}$. This means that in the residual graph, v_i^{j+1} is not reachable from s, proving the induction hypothesis.

By the above argument, the only vertex reachable from s in V_k is u_k. Now consider the sink t. For any $j \in [k]$, (u_j, t) is contained in a path in $\mathcal{P}_j$ and thus its flow matches its capacity. For edge (u_k, t), there are two paths in $\mathcal{P}$ that contain this edge, the first one is in $\mathcal{P}_k$ which carries w_k units of flow and the other is P^* which carries $i^* - 1$ units of flow. So $(u_k, t) = (v_{i^*}^k, t)$ is also full. Thus t is not reachable from s. □

Claim 6.5.3 and Claim 6.5.4 prove that $\mathcal{P}$ induces a maximum s-t flow in G. We are now done as the amount of flow carried by all flow paths in $\mathcal{P}$ is divisible by $n+1$ except for P^*. This is because the flow carried by each path in $\mathcal{P}_j$ for $j \in [k]$ is of weight w_j and $(n+1)$ is a factor of w_j. As the flow carried by P^* is $i^* - 1$, the total flow in $\mathcal{P}$ is $K \cdot (n+1) + (i^* - 1)$ for some integer $K \geq 1$. By max-flow min-cut duality, $w^* \bmod (n+1) = i^* - 1$. □

We can now prove Theorem 6.7 using this reduction, the standard connection between space complexity of streaming algorithms and communication complexity, and our communication lower bound for hidden-pointer chasing in Theorem 6.6.

Proof of Theorem 6.7. Let $\mathcal{A}$ be a p-pass streaming algorithm for computing the value of a minimum s-t cut in weighted directed graphs. To avoid confusion, in the following, we use N to denote the number of vertices in the graph G and n for the size of universes in HPC. Hence, our goal is to prove a lower bound of $\Omega(N^2/p^5)$ on the space complexity of $\mathcal{A}$.

We give a reduction from HPC_k for $k = 2p+1$. Given an instance of HPC_k, the players first construct the graph $G(V, E, w)$ in the reduction of this section based on their inputs with no communication. Next, they create a stream σ of edges of E such that edges depending on input to P_D appear first, then P_C, P_B and P_A in this order and input-independent edges appear last. The players run $\mathcal{A}$ on σ and communicate the state of $\mathcal{A}$ between each other whenever necessary to compute the value of a minimum weighted s-t cut in G.

By Lemma 6.5.1, the value of the minimum s-t cut in G immediately determines the pointer z_k, hence proving the correctness of the protocol. The number of phases and communication cost of this protocol can be determined as follows. Each pass of the streaming algorithm translates into at most two phases in the protocol and hence the resulting protocol has strictly smaller than k phases. The total communication by players in this protocol is at most $O(k \cdot S)$ where S denotes the space complexity of $\mathcal{A}$. As such, by Theorem 6.6, we have, $k \cdot S = \Omega(n^2/k^2)$ which implies $S = \Omega(n^2/k^3)$. Since the total number of vertices in the graph is $N = O(k \cdot n)$ and $k = \Theta(p)$, we obtain a lower bound of $\Omega(N^2/p^5)$ on the space complexity of $\mathcal{A}$, finalizing the proof for the directed graphs.

To extend the results to undirected graphs, we can simply use the standard reduction of finding a maximum flow in directed graphs to finding a maximum flow in undirected graphs described in, for example [197] (see also Appendix C.2 in [240]). This reduction works by turning each directed edge $e = (u, v)$ with capacity c_e in the graph to three undirected edges $\{s, v\}$, $\{u, v\}$ and $\{t, u\}$ each with capacity c_e. It is then easy to see that after pushing an initial flow of (s, v, u, t) with c_e units of flow on every edge (u, v), the residual graph obtained would be equivalent to the original directed graph. Hence, solving s-t maximum flow on this undirected graph would also solve the problem for the original directed graph (see [197, 240] for the formal proof). As thus reduction can be done on the graph $G(V, E, w)$ constructed in this section with no further communication between the players, the results in this proof extend to undirected graphs as well, finalizing the proof. □

Remark 6.5.5. The reduction in this section creates a multi-graph G. However, we can

easily transform this graph to a simple graph without changing the minimum cut value, while increasing the number of vertices by only a constant factor. The transformation is as follows: turn any vertex v_i^j in layer V_j of the graph G into three vertices w_i^j, a_i^j and b_i^j. Connect w_i^j to a_i^j and b_i^j with edges of weight w_0 (which is effectively infinity). The input-independent edges going out of v_i^j to t now goes out of w_i^j to t instead. For any odd j, any edge $(v_i^j, v_{i'}^{j+1})$ is now turned into an edge $(a_i^j, w_{i'}^{j+1})$ if the edge was added because of A_{x_i} and $(b_i^j, w_{i'}^{j+1})$ if it was added because of B_{x_i}. We do the same for even values of j by using C_{y_i} and D_{y_i} instead. It is easy to see that the weight of minimum s-t is the same in this new graph and that this graph does not have any parallel edges anymore.

6.5.2. The Lexicographically-First MIS Problem

Proof of Theorem 6.8 is also by a reduction from the hidden-pointer chasing (HPC) problem. We turn an instance $(\boldsymbol{A}, \boldsymbol{B}, \overline{C}, \boldsymbol{D})$ of HPC_k over universes $\mathcal{X}$ and $\mathcal{Y}$, into an undirected graph $G(V, E)$. The reduction is as follows (see Figure 6.6 for an example):

- The vertex-set V of G is partitioned into $k+1$ layers $V_0, \ldots, V_k$ each of size n plus a single vertex s (hence G has $(k+1)n+1$ vertices). We denote the i-th vertex in layer V_j by v_i^j. In the lexicographic order, the vertices in layer V_0 appear first, followed by vertices in $V_1, \ldots, V_k$ in this order. Inside each layer V_j, the ordering is by the index, i.e., in the order $v_1^j, \ldots, v_n^j$.

- The edge-set E contains the following edges:

  - vertex v_1^0 is connected to all other vertices in V^0.

  - for all $i \in [n]$, if $A_{x_i} \in \boldsymbol{A}$ (resp. $B_{x_i} \in \boldsymbol{B}$) does *not* contain $y_{i'} \in \mathcal{Y}$, we connect v_i^j in layer V_j to $v_{i'}^{j+1}$ in layer V_{j+1} for every *even* $0 \leq j < k$.

  - for all $i \in [n]$, if $C_{y_i} \in \overline{C}$ (resp. $D_{y_i} \in \boldsymbol{D}$) does *not* contain $x_{i'} \in \mathcal{X}$, we connect v_i^j in layer V_j to $v_{i'}^{j+1}$ in layer V_{j+1} for every *odd* $0 < j < k$.

This concludes the description of the graph $G(V, E)$ in the reduction. It is straightfor-

ward to verify that this graph can be constructed from an instance $(\boldsymbol{A}, \boldsymbol{B}, \overline{C}, \boldsymbol{D})$ with no communication between the players. We now establish the correctness of the reduction.

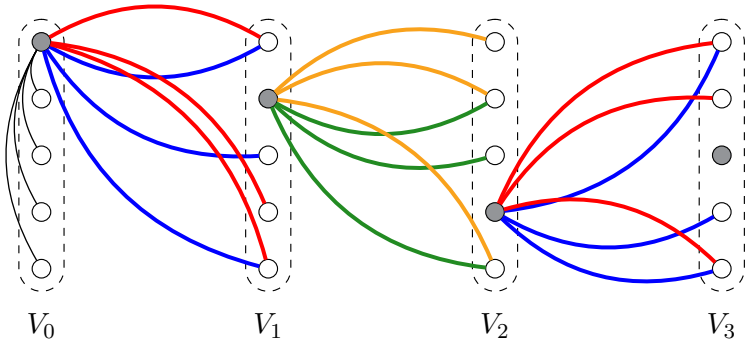


Figure 6.6: Illustration of the graph in reducing lexicographically-first MIS from HPC_3 with $n = 5$. The black (thin) edges incident on s are input-independent while blue, red , brown, and green (thick) edges depend on the inputs of P_A, P_B, P_C, and P_D, respectively. The marked nodes denote the vertices corresponding to pointers $z_0, \ldots, z_3$. The edges incident on "non-pointer" vertices are omitted. This construction has parallel edges but similar to Remark 6.5.5, we can remove them.

Lemma 6.5.6. *In the reduction above, the pointer $z_k = x_i$ (resp. $z_k = y_i$) when k is even (resp. odd) iff v_i^k belongs to the lexicographically-first* MIS *of G.*

Proof. Let $\mathcal{M}$ be the lexicographically-first MIS of G. We prove by induction that for any even (resp. odd) $j \in \{0, 1, \ldots, k\}$, there is a unique vertex v_i^j from layer V_j that belongs to $\mathcal{M}$ and that vertex corresponds to the pointer z_j, namely, $x_i = z_j$ (resp. $y_i = z_j$).

The base case is trivial since $z_0 = x_1$, v_1^0 appears first in the lexicographical ordering of vertices, and v_1^0 is connected to all vertices in layer V_0. We now prove the induction step. Suppose j is even; the other case is symmetric. By induction hypothesis, v_i^j is the unique vertex in layer V_j that belongs to $\mathcal{M}$ where $x_i = z_j$. By construction of G, v_i^j is connected to all vertices in layer $j + 1$ except for the vertex $v_{i'}^{j+1}$, where $\{y_{i'}\} = A_{x_i} \cap B_{x_i}$. Hence, $v_{i'}^{j+1}$ is the unique index in V_{j+1} that belongs to $\mathcal{M}$. The proof is concluded by noting that $z_{j+1} = y_{i'}$ by definition. □

Proof of Theorem 6.8 now follows from Lemma 6.5.6 and Theorem 6.6 the same exact way

as in proof of Theorem 6.7 in the last section. For completeness, we present this proof here.

Proof of Theorem 6.8. Let $\mathcal{A}$ be a p-pass streaming algorithm for finding the lexicographically-first MIS of an undirected graph. To avoid confusion, in the following, we use N to denote the number of vertices in the graph G and n for the size of universes in HPC. Hence, our goal is to prove a lower bound of $\Omega(N^2/p^5)$ on the space complexity of $\mathcal{A}$.

We give a reduction from HPC_k for $k = 2p+1$. Given an instance of HPC_k, the players first construct the graph $G(V, E)$ in the reduction of this section based on their inputs with no communication. Next, they create a stream σ of edges of E such that edges depending on input to P_D appear first, then P_C, P_B and P_A in this order and input-independent edges appear last. The players then run $\mathcal{A}$ on σ and communicate the state of $\mathcal{A}$ between each other whenever necessary to find the lexicographically-first MIS $\mathcal{M}$ of G.

By Lemma 6.5.6, the vertex in layer V_k of G that belongs to $\mathcal{M}$ determines the pointer z_k, hence proving the correctness of the protocol. The number of phases and communication cost of this protocol can be determined as follows. Each pass of the streaming algorithm translates into at most two phases in the protocol and hence the resulting protocol has strictly smaller than k phases. The total communication by players in this protocol is at most $O(k \cdot S)$ where S denotes the space complexity of $\mathcal{A}$. As such, by Theorem 6.6, we have, $k \cdot S = \Omega(n^2/k^2)$ which implies $S = \Omega(n^2/k^3)$. Since the total number of vertices in the graph is $N = O(k \cdot n)$ and $k = \Theta(p)$, we obtain a lower bound of $\Omega(N^2/p^5)$ on the space complexity of $\mathcal{A}$, finalizing the proof. □

We also note that similar to the previous section, we can also turn the graph G in the reduction of this section to a simple graph with no parallel edges using essentially the same gadget. We omit the details.

CHAPTER 7

Round Complexity for Submodular Function Minimization

In this chapter, we study the round complexity for submodular function minimization (SFM). A study of this question was initiated by Balkanski and Singer in [28] who proved that any polynomial query SFM algorithm must proceed in $\Omega(\frac{\log N}{\log \log N})$ rounds. This still leaves open the possibility of polynomial query poly-logarithmic round algorithms. Indeed for the related problem of submodular function *maximization* subject to cardinality constraint, in a different paper [27], Balkanski and Singer showed that the correct answer is indeed $\tilde{\Theta}(\log N)$. They proved that with polynomially many queries no constant factor approximation is possible with $o\left(\frac{\log N}{\log \log N}\right)$ rounds, while an 1/3-approximation can be obtained in $O(\log N)$-rounds[1]. Can the situation be the same for SFM?

7.1. Main Results

We prove a ***polynomial*** lower bound on the number of rounds needed by any polynomial query SFM algorithm.

Theorem 7.1. *For any constant $\delta > 0$ and any $1 \leq c \leq N^{1-\delta}$, any randomized algorithm for SFM on an N element universe making $\leq N^c$ evaluation oracle queries per round and succeeding with probability $\geq 2/3$ must have $\Omega\left(\frac{N^{1/3}}{(c \log N)^{1/3}}\right)$ rounds-of-adaptivity. This is true even when the range of the submodular function is $\{-N, -N+1, \ldots, N-1, N\}$, and even if the algorithm is only required to output the value of the minimum.*

We note that a polynomial lower bound on the number of rounds holds even if the algorithm is allowed to make $2^{N^{1-\delta}}$ queries per round for any $\delta > 0$, and the lower bound on the number of rounds is $\tilde{\Omega}(N^{1/3})$ for polynomial query algorithms. Our construction also proves lower bounds on the number of rounds required for *approximate* submodular function minimization. In this problem, one assumes via scaling that the function's range is in $[-1, +1]$ and the goal is to return a set whose value is within an additive ε from the minimum. We

[1] This result has since been improved [25, 86, 87, 111, 112, 196]; see Section 7.1.1 for details.

can prove an $\widetilde{\Omega}(1/\varepsilon)$-lower bound on the number of rounds required for approximate SFM. The only previous work ruling out ε-approximate minimizers is another work of Balkanski and Singer [26] who proved that *non-adaptive* algorithms, that is single round algorithms, cannot achieve any non-trivial approximation with polynomially many queries.

The submodular functions we construct to prove Theorem 7.1 are closely related to the rank functions of nested matroids, a special kind of laminar matroids. As a result, we prove a similar result as in Theorem 7.1 for matroid intersection.

Theorem 7.2. *For any constant $\delta > 0$ and any $1 \leq c \leq N^{1-\delta}$, any randomized algorithm for matroid intersection on an N element universe making $\leq N^c$* rank-oracle *queries per round and succeeding with probability $\geq 2/3$ must have $\Omega\left(\frac{N^{1/3}}{(c \log N)^{1/3}}\right)$ rounds-of-adaptivity. This is true even when the two matroids are* nested matroids*, a special class of laminar matroids, and also when the algorithm is only required to output the value of the optimum.*

In particular, any algorithm making polynomially many queries to the rank oracle must have $\widetilde{\Omega}(N^{1/3})$ rounds of adaptivity, even to figure out the size of the largest common independent set. That is, even the "decision" version of the question (is the largest cardinality at least some parameter K) needs polynomially many rounds of adaptivity.

Our results shows that in the general query model, SFM and matroid intersection cannot be solved in polynomial time in poly-logarithmic rounds, even with randomization. This is in contrast to *specific* explicitly described succinct SFM and matroid intersection problems. For instance, global minimum cuts in an undirected graph is in **NC** [177], finding minimum s-t-cuts with poly-bounded capacities is in **RNC** [181], and linear and graphic matroid intersection is in **RNC** [214]. More recently, inspired by some of these special cases, Gurjar and Rathi [140] defined a class of submodular functions called *linearly representable* submodular functions and gave **RNC** algorithms for the same.

Our lower bounding submodular functions fall in a class introduced by Balkanski and Singer [28] which we call *partition submodular functions*. Given a partition $\mathcal{P} = (P_1, \ldots, P_r)$

of the universe U, the value of a partition submodular function $f(S)$ depends only on the *cardinalities* of the $|S \cap P_i|$'s. In particular, $f(S) = h(\vec{x})$ where $\vec{x}$ is an r-dimensional non-negative integer valued vector with $\vec{x}_i := |S \cap P_i|$, and h is a discrete submodular function on a hypergrid. Note that when $r = 1$, the function h is a univariate concave function, and when $r = n$ we obtain general submodular functions. Thus, partition submodular functions form a nice way of capturing the complexity of a submodular function.

The [28] functions are partition submodular and they prove an $\Omega(r)$-lower bound for their specific functions. Their construction idea has a bottleneck of $r = O(\log N)$, and thus cannot prove a polynomial lower bound. Our lower bound functions are also partition submodular, and we also prove an $\Omega(r)$ lower bound though we get r to be polynomially large in the size of the universe. Furthermore, our partition submodular functions turn out to be closely related to ranks of nested matroids which lead to our lower bound for parallel matroid intersection.

7.1.1. Related Work

For parallel algorithms, the depth required for the "decision" version and the "search" version may be vastly different. In a thought provoking paper [182], Karp, Upfal and Wigderson considered this question. In particular, they proved that any efficient algorithm that finds a maximum independent set in a *single* (even a partition) matroid with access to an *independence oracle* must proceed in $\widetilde{\Omega}(N^{1/3})$ rounds. On the other hand, with access to a *rank* oracle which takes S and returns $r(S)$, the size of the largest independent set in S, there is a simple algorithm[2] which makes N queries in a single round and finds the optimal answer. Our lower bound shows that for matroid intersection, rank oracles also suffer a polynomial lower bound, even for the decision version of the problem. At this point, we should mention a very recent work of Ghosh, Gurjar, and Raj [129] which showed that if there existed polylogarithmic round algorithms for the (weighted) decision version for matroid intersection with rank-oracles, then in fact there exists *deterministic* polylogarithmic round algorithms for the *search* version. A similar flavor result is also present in [214]. Unfortunately, our

[2]Order elements as $e_1, \ldots, e_N$ and query $r(\{e_1, \ldots, e_i\})$ for all i, and return the points at which the rank changes.

result proves that polylogarithmic depth is impossible for arbitrary matroids (even nested ones), even when access is via rank oracles.

The rounds-of-adaptivity versus query complexity question has seen a lot of recent work on submodular function *maximization*. As mentioned before, Balkanski and Singer [27] introduced this problem in the context of maximizing a non-negative monotone submodular function $f(S)$ subject to a cardinality constraint $|S| \leq k$. This captures **NP**-hard problems, has a *sequential* greedy $(1-\frac{1}{e})$-approximation algorithm [216], and obtaining anything better requires [215, 245] exponentially many queries. [27] showed that obtaining even an $O\left(\frac{1}{\log N}\right)$-approximation with polynomially many queries requires $\Omega\left(\frac{\log N}{\log\log N}\right)$ rounds, and gave an $O(\log N)$-round, polynomial query, $\frac{1}{3}$-approximation. Soon afterwards, several different groups [25, 86, 87, 111, 112, 114] gave $\left(1-\frac{1}{e}-\varepsilon\right)$-approximation algorithms making polynomially many queries which run in $\text{poly}(\log N, \frac{1}{\varepsilon})$-rounds, even when the constraint on which S to pick is made more general. More recently, Li, Liu and Vondrák [196] showed that the dependence of the number of rounds on ε (the distance from $1-1/e$) must be a polynomial. Also related is the question of maximizing a non-negative non-monotone submodular function without any constraints. It is known that a random set gives a $\frac{1}{4}$-approximation, and a sequential "double-greedy" $\frac{1}{2}$-approximation was given by Buchbinder, Feldman, Naor, and Schwartz [71], and this approximation factor is tight [116]. Chen, Feldman, and Karabasi [89] gave a nice parallel version obtaining an $\left(\frac{1}{2}-\varepsilon\right)$-approximation in $O(\frac{1}{\varepsilon})$-rounds.

In the continuous optimization setting, the question of understanding the "parallel complexity" of minimizing a non-smooth convex function was first studied by Nemirovski [217]. In particular, the paper studied the problem of minimizing a bounded-norm convex (non-smooth) function over the unit ℓ_∞ ball in N-dimensions, and showed that any polynomial query (value oracle or gradient oracle) algorithm which comes ε-close must have $\widetilde{\Omega}(N^{1/3}\ln(1/\varepsilon))$ rounds of adaptivity. Nemirovski [217] conjectured that the lower bound should be $\widetilde{\Omega}(N\ln(1/\varepsilon))$, and this is still an open question. When the dependence on ε is allowed to be polynomial,

then the sequential vanilla gradient descent outputs an ε-minimizer in $O(1/\varepsilon^2)$-rounds (over Euclidean unit norm balls), and the question becomes whether parallelism can help over gradient descent in some regimes of ε. Duchi, Bartlett, and Wainwright [105] showed an $O(N^{1/4}/\varepsilon)$-query algorithm which is better than gradient-descent when $\frac{1}{\varepsilon^2} > \sqrt{N}$. A matching lower bound in this regime was shown recently by Bubeck et al. [70], and this paper also gives another algorithm which has better depth dependence in some regime of ε. It is worth noting that submodular function minimization can also be thought of as minimizing the Lovász extension which is a non-smooth convex function. Unfortunately, the domain of interest (the unit cube) has ℓ_2-radius $\sqrt{N}$, and the above algorithms do not imply "dimension-free" ε-additive approximations for submodular function minimization. Our work shows that $\Omega(1/\varepsilon)$-rounds are needed, and it is an interesting open question whether a poly$(N, \frac{1}{\varepsilon})$-lower bound can be shown on the number of rounds, or whether one can achieve efficient ε-approximations in rounds independent of N.

7.2. Description of our Lower Bound Functions

We begin by formally defining partition submodular functions and some properties of such functions. We then describe in detail the lower bound functions that we use in the proof of Theorem 7.1.

7.2.1. Partition Submodular Functions

Let U be a universe of elements and $\mathcal{P} = (P_1, \ldots, P_r)$ be a partition of the elements of U. Let $h : \mathbb{Z}^r_{\geq 0} \to \mathbb{R}$ be a function whose domain is the r-dimensional non-negative integer hypergrid. Given $(\mathcal{P}, h)$, one can define a set-function $f_{\mathcal{P}} : 2^U \to \mathbb{R}$ as follows:

$$f_{\mathcal{P}}(S) = h\left(|S \cap P_1|, \ldots, |S \cap P_r|\right) \tag{7.1}$$

In plain English, the value of $f_{\mathcal{P}}(S)$ is a function only of the *number* of elements of each part that is present in S. We say that $f_{\mathcal{P}}$ is induced by the partition P and h. A ***partition submodular function*** is a submodular function which is induced by some partition P and some hypergrid function h.

A function defined by (P, h) is submodular if and only if h satisfies the same decreasing marginal property as f. To make this precise, let us settle on some notation. Throughout the paper, for any integer k, we use $[k]$ to denote the set $\{0, 1, \ldots, k\}$. First, note that the domain of h is the r-dimensional hypergrid $[|P_1|] \times [|P_2|] \times \cdots \times [|P_r|]$. For brevity's sake, we call this $\mathbf{dom}(h)$. We use boldfaced letters like $\vec{x}, \mathbf{y}$ to denote points in $\mathbf{dom}(h)$. When we write $\vec{x} + \mathbf{y}$ we imply coordinate-wise sum. Given $i \in \{1, \ldots, r\}$, we use $\mathbf{e}_i$ to denote the r-dimensional vector having 1 at the ith coordinate and 0 everywhere else. The function h induces r different ***marginal*** functions defined as

$$\text{For } 1 \leq i \leq r, \quad \partial_i h(\vec{x}) := h(\vec{x} + \mathbf{e}_i) - h(\vec{x}) \tag{7.2}$$

The domain of $\partial_i h$ is $[|P_1|] \times [|P_2|] \times \cdots \times [|P_i| - 1] \times \cdots \times [|P_r|]$.

Definition 7.2.1. *We call a function $h : \mathbb{Z}^r \to \mathbb{R}$ defined over an integer hypergrid $\mathbf{dom}(h)$ (hypergrid) submodular if and only if for every $1 \leq i \leq r$, for every $\vec{x} \in \mathbf{dom}(h)$ with $\vec{x}_i < |P_i|$, and every $1 \leq j \leq r$, we have*

$$\partial_j h(\vec{x}) \geq \partial_j h(\vec{x} + \mathbf{e}_i) \tag{7.3}$$

Lemma 7.2.2. *A set function $f_{\mathcal{P}}$ defined by a partition P and hypergrid function h as in (7.1) is (partition) submodular if and only if h is (hypergrid) submodular.*

Proof. Let $A \subseteq U$ and let $\vec{x}$ be the r-dimensional integer vector with $\vec{x}_i := |A \cap P_i|$. Pick elements $e, e' \in U \setminus A$. Let $e \in P_i$ and $e' \in P_j$ for $1 \leq i, j \leq r$. Note that j could be the same as i. Then $f_{\mathcal{P}}$ is submodular is equivalent to $f_{\mathcal{P}}(A+e') - f_{\mathcal{P}}(A) \geq f_{\mathcal{P}}(A+e+e') - f_{\mathcal{P}}(A+e)$, which is equivalent to (7.3). □

The following lemma shows that minima of partition submodular functions can be assumed to take all or nothing of each part.

Lemma 7.2.3. *Let $f_{\mathcal{P}}$ be a partition submodular function induced by a partition $P = (P_1, \ldots, P_r)$ and hypergrid function h. Let O be a maximal by inclusion minimizer of f. Then, $O \cap P_i \neq \emptyset$ implies $O \cap P_i = P_i$.*

Proof. Let $\vec{x} \in \mathbf{dom}(h)$ be the vector induced by O, that is, $\vec{x}_i = |O \cap P_i|$ for all $1 \leq i \leq r$. For the sake of contradiction, assume $0 < \vec{x}_i < |P_i|$. Let e_1 and e_2 be two arbitrary elements in $O \cap P_i$ and $P_i \setminus O$ respectively. Since O is the minimizer, $f_{\mathcal{P}}(O) - f_{\mathcal{P}}(O - e_1) \leq 0$. Now note that the LHS is precisely $\partial_i h(\vec{x} - \mathbf{e}_i)$. And this is also equal to $f(O - e_1 + e_2) - f(O - e_1)$ and thus this is also ≤ 0. By submodularity, however, $f(O + e_2) - f(O) \leq f(O - e_1 + e_2) - f(O - e_1)$, and thus we obtain $f(O + e_2) \leq f(O)$ which contradicts that O was an inclusion-wise maximal minimizer. □

7.2.2. Suffix Functions

The lower bound functions we construct are partition submodular functions defined with respect to a partition $\mathcal{P} = (P_1, \ldots, P_r)$ of the universe U of N elements into r parts. The number of parts r is an odd integer whose value will be set to be $\tilde{\Theta}(N^{1/3})$. Each part P_i has the same size n, where n is an even positive integer such that $nr = N$. The hypergrid submodular function $h : [n]^r \to \mathbb{Z}$ which define the partition submodular function are themselves defined using *suffix* functions, which we describe below.

Let g be an integer which is divisible by 4 and which is $\tilde{\Theta}(\sqrt{n})$. That is, $\left(\frac{n}{2} - g\right)$ is "many standard deviations" away from $\frac{n}{2}$, and in particular, any random subset of an n-universe set has cardinality within $\pm g$ of the expected value with all but inverse polynomial probability. As described in the previous informal discussion, the following linear suffix functions play a key role in the description of the marginals. Define

$$\text{For any } 1 \leq t \leq r, \quad \ell_t(\vec{x}) := \sum_{s=t}^{r} \left(\vec{x}_s - \left(\frac{n}{2} - g\right)\right) - \frac{gr}{4} \tag{7.4}$$

Given $\vec{x}$, let $a := a(\vec{x}) \in [r]$ be the *odd*-coordinate $t \in [r]$ with the largest $\ell_t(\vec{x})$, breaking ties towards smaller indices in case of ties. Let $b := b(\vec{x}) \in [r]$ be the *even*-coordinate $t \in [r]$

with the largest $\ell_t(\vec{x})$, breaking ties towards smaller indices in case of ties. We call $\{a, b\}$ the largest odd-even index of $\vec{x}$.

Now we are ready to describe our lower bounding functions. First define the function $h : [n]^r \to \mathbb{Z}$ as follows

$$h(\vec{x}) = \|\vec{x}\|_1 - \Big(\max(0, \ell_a(\vec{x})) + \max(0, \ell_b(\vec{x}))\Big) \tag{7.5}$$

The above function contains the seed of the hardness, and satisfies (P1) and (P3). However, the above function, for the precise choice of g we will finally choose, will in fact be non-negative. To obtain the lower bounding functions which treats P_r specially, we define

$$h^*(\vec{x}) = \begin{cases} h(\vec{x}) & \text{if } \vec{x}_r \leq \frac{n}{2} - \frac{g}{4} \\ h(\vec{x}_\downarrow) - \left(\vec{x}_r - \left(\frac{n}{2} - \frac{g}{4}\right)\right) & \text{otherwise} \end{cases} \tag{7.6}$$

$$\text{where, } \vec{x}_\downarrow := \Big(\vec{x}_1, \ldots, \vec{x}_{r-1}, \min(\vec{x}_r, \frac{n}{2} - \frac{g}{4})\Big)$$

In Section 7.2.3, for completeness sake, we give a direct proof that both the functions, h and h^* are hypergrid submodular. However, as we show in Section 7.4, these functions arise as sum of rank functions of particular nested matroids, and thus give a more principled reason why these functions are submodular. In Section 7.2.4, we show that the function h is non-negative, while $h^*(0, 0, \ldots, 0, n)$ attains a negative value of $-g/2$. In Section 7.2.5, we show that i-balanced vectors, for $i < r/2$, cannot distinguish between h and h^*. This, in turn, is used in Section 7.3 to prove the lower bound for parallel SFM.

7.2.3. Submodularity

We first prove that $h : [n]^r \to \mathbb{Z}$ is submodular, and then use this to prove that $h^* : [n]^r \to \mathbb{Z}$ is submodular. We need to prove

Lemma 7.2.4. *Fix $\vec{x}$ and a coordinate $1 \leq i \leq r$. Let $\mathbf{y} := \vec{x} + \mathbf{e}_i$. Let j be any arbitrary*

coordinate. Then,

$$\partial_j h(\vec{x}) \geq \partial_j h(\mathbf{y}) \tag{7.7}$$

The high-level reason why h is submodular is when one moves from $\vec{x}$ to $\mathbf{y} = \vec{x} + \mathbf{e}_i$, the odd-even index $\{a, b\}$ of $\mathbf{y}$ can only "move to the left", that is, become smaller. Formally,

Claim 7.2.5. *Let $\vec{x}$ be any point and let $\mathbf{y} := \vec{x} + \mathbf{e}_i$. Suppose a is the odd coordinate t with the largest $\ell_t(\vec{x})$ breaking ties towards smaller indices. Suppose a' is the odd coordinate t with the largest $\ell_t(\mathbf{y})$ breaking ties towards smaller indices. If $a' \neq a$, then (i) $a' \leq i < a$, and (ii) $\ell_{a'}(\mathbf{y}) = \ell_a(\mathbf{y})$. A similar statement is true for even coordinates.*

Proof. First from the definition, observation that $\ell_t(\mathbf{y}) = \ell_t(\vec{x})$ if $t > i$ and $\ell_t(\mathbf{y}) = \ell_t(\vec{x}) + 1$ if $t \leq i$. Thus, if $a' \neq a$, we must have that $a' \leq i < a$, establishing (i). Furthermore, since $a' < a$, we must have $\ell_a(\vec{x}) \geq \ell_{a'}(\vec{x}) + 1$ for otherwise a' would've been chosen with respect to $\vec{x}$. Since $\ell_{a'}(\mathbf{y}) \geq \ell_a(\mathbf{y})$, again by the observation of the first line, we establish (ii). □

To see how the claim helps in proving Lemma 7.2.4, it is instructive to first establish how the marginals of the function defined in (7.5) look like. To this end, define the following indicator functions. For any $1 \leq t \leq n$ and for any $1 \leq i \leq n$, define

$$\mathsf{C}_t(\vec{x}) = \begin{cases} -1 & \text{if } \ell_t(\vec{x}) \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \mathsf{C}_t^i(\vec{x}) = \mathsf{C}_t(\vec{x}) \cdot \mathbf{1}_{\{i \geq t\}}$$

where $\mathbf{1}_{\{i \geq t\}}$ is the indicator function taking the value 1 if $i \geq t$ and 0 otherwise. Using these notations, we can describe the r different marginals at $\vec{x}$ succinctly as

Lemma 7.2.6. *Fix $\vec{x}$ in the domain of h. Let $\{a, b\}$ be largest odd-even index of $\vec{x}$. Then,*

$$\forall 1 \leq i \leq r, \quad \partial_i h(\vec{x}) = 1 + \mathsf{C}_a^i(\vec{x}) + \mathsf{C}_b^i(\vec{x}) \tag{Marginals}$$

In plain English, given a point $\vec{x}$, one first finds the largest odd-even index $\{a, b\}$ of $\vec{x}$. If

any of these function values are negative, throw them away from consideration: the suffixes aren't large enough. Next, given a coordinate i, the marginal $\partial_i h(\vec{x})$ depends on where i lies in respect to a and b (if they are still in consideration). If i is smaller than both, then the marginal is 1, if i is smaller than one, then the marginal is 0, if i is greater than or equal to both, the marginal is -1. Given this understanding of how the marginals look like, it is perhaps clear why Claim 7.2.5 implies submodularity : as $\{a, b\}$ move left the the marginal of any coordinate j can only decrease when one moves to $\mathbf{y}$.

Proof of Lemma 7.2.6. Fix an $\vec{x}$ and a coordinate i. Let $\mathbf{y} = \vec{x} + \mathbf{e}_i$. Let's consider $h(\mathbf{y}) - h(\vec{x})$ using (7.5), and then show it is precisely as asserted in (Marginals). First note that we can rewrite

$$h(\vec{x}) = \|\vec{x}\|_1 + \mathsf{C}_a(\vec{x})\ell_a(\vec{x}) + \mathsf{C}_b(\vec{x})\ell_b(\vec{x}) \tag{7.8}$$

Consider the expression $\mathsf{C}_a(\mathbf{y})\ell_a(\mathbf{y}) - \mathsf{C}_a(\vec{x})\ell_a(\vec{x})$. If $i < a$, then $\ell_a(\mathbf{y}) = \ell_a(\vec{x})$, and thus $\mathsf{C}_a(\mathbf{y}) = \mathsf{C}_a(\vec{x})$, and thus the expression evaluates to 0. If $i \geq a$, then $\ell_a(\mathbf{y}) = \ell_a(\vec{x}) + 1$. For the expression to contribute anything non-zero, we must have $\ell_a(\mathbf{y}) \geq 1$ implying $\ell_a(\vec{x}) \geq 0$, or in other words, $\mathsf{C}_a(\vec{x}) = \mathsf{C}_a(\mathbf{y}) = -1$. And in that case, we get $\mathsf{C}_a(\mathbf{y})\ell_a(\mathbf{y}) - \mathsf{C}_a(\vec{x})\ell_a(\vec{x}) = -1$. To summarize,

$$\mathsf{C}_a(\mathbf{y})\ell_a(\mathbf{y}) - \mathsf{C}_a(\vec{x})\ell_a(\vec{x}) = \begin{cases} 0 & \text{if } i < a \text{ or if } \ell_a(\vec{x}) < 0, \text{ that is, } \mathsf{C}_a(\vec{x}) = 0 \\ -1 & \text{otherwise, that is, if } i \geq a \text{ and } \mathsf{C}_a(\vec{x}) = -1 \end{cases}$$

In other words,

$$\mathsf{C}_a(\mathbf{y})\ell_a(\mathbf{y}) - \mathsf{C}_a(\vec{x})\ell_a(\vec{x}) = \mathsf{C}^i_a(\vec{x}) \tag{7.9}$$

Now suppose $\{a', b'\}$ are the odd-even index of $\mathbf{y}$. The above discussion proves the claim when $\{a', b'\} = \{a, b\}$. Indeed, plugging (7.9) into (7.8), we get

$$h(\mathbf{y}) - h(\vec{x}) = \underbrace{(\|\mathbf{y}\|_1 - \|\vec{x}\|_1)}_{=1} + \mathsf{C}^i_a(\vec{x}) + \mathsf{C}^i_b(\vec{x})$$

A little more care is needed to take care of the case when $\{a', b'\} \neq \{a, b\}$. Suppose $a \neq a'$. Then, by Claim 7.2.5, we get that $a' < i \leq a$ and $\ell_{a'}(\mathbf{y}) = \ell_a(\mathbf{y})$. Thus, $\mathsf{C}_{a'}(\mathbf{y})\ell_{a'}(\mathbf{y}) - \mathsf{C}_a(\vec{x})\ell_a(\vec{x}) = \mathsf{C}_a(\mathbf{y})\ell_a(\mathbf{y}) - \mathsf{C}_a(\vec{x})\ell_a(\vec{x})$ and the proof follows as in the $a' = a$ case. The case $b' \neq b$ is similar. □

Proof of Lemma 7.2.4. Let $\{a_1, b_1\}$ be the odd-even index of $\vec{x}$. Let $\{a_2, b_2\}$ be the odd-even index of $\succ_{\mathbf{y}}$. From the definition of the marginals, what we need to show is

$$\mathsf{C}^j_{a_1}(\vec{x}) + \mathsf{C}^j_{b_1}(\vec{x}) \geq \mathsf{C}^j_{a_2}(\mathbf{y}) + \mathsf{C}^j_{b_2}(\mathbf{y}) \tag{7.10}$$

We will show this term by term, and focus on a_1, a_2. For any $1 \leq t \leq r$, observe that $\ell_t(\mathbf{y}) \geq \ell_t(\vec{x})$, and thus $\mathsf{C}_t(\vec{x}) \geq \mathsf{C}_t(\mathbf{y})$. Thus if $a_1 = a_2$, we are done.

If $a_1 \neq a_2$, then by Claim 7.2.5 $a_2 \leq i < a_1$ and $\ell_{a_2}(\mathbf{y}) = \ell_{a_1}(\mathbf{y}) \geq \ell_{a_1}(\vec{x})$. This implies $\mathsf{C}_{a_1}(\vec{x}) \geq \mathsf{C}_{a_2}(\mathbf{y})$. Since $a_2 < a_1$, we get that $\mathbf{1}_{\{j \geq a_2\}} \geq \mathbf{1}_{\{j \geq a_1\}}$. Since C is non-positive, we get $\mathsf{C}^j_{a_1}(\vec{x}) = \mathbf{1}_{\{j \geq a_1\}} \cdot \mathsf{C}_{a_1}(\vec{x}) \geq \mathbf{1}_{\{j \geq a_2\}} \cdot \mathsf{C}_{a_2}(\mathbf{y}) = \mathsf{C}^j_{a_2}(\mathbf{y})$. □

Lemma 7.2.7. *The function h^* as defined in (7.6) is submodular*

Proof. We recall the definition.

$$h^*(\vec{x}) = \begin{cases} h(\vec{x}) & \text{if } \vec{x}_r \leq \frac{n}{2} - \frac{g}{4} \\ h(\vec{x}_\downarrow) - \left(\vec{x}_r - \left(\frac{n}{2} - \frac{g}{4}\right)\right) & \text{otherwise} \end{cases} \quad \text{where, } \vec{x}_\downarrow := \left(\vec{x}_1, \ldots, \vec{x}_{r-1}, \min(\vec{x}_r, \frac{n}{2} - \frac{g}{4})\right)$$

Observe,

- If $j \neq r$, then $\partial_j h^*(\vec{x}) = \partial_j h(\vec{x}_\downarrow)$.
- If $j = r$, then $\partial_r h^*(\vec{x}) = -1$ if $\vec{x}_r \geq \frac{n}{2} - \frac{g}{4}$, else $\partial_r h^*(\vec{x}) = \partial_r h(\vec{x})$.

Now pick $\vec{x} \in [n]^r$, $\mathbf{y} := \vec{x} + \mathbf{e}_i$. Since $\vec{x}_\downarrow$ is coordinate wise dominated by $\mathbf{y}_\downarrow$, we get that if $j \neq r$,

$$\partial_j h^*(\vec{x}) = \partial_j h(\vec{x}_\downarrow) \underbrace{\geq}_{\text{Lemma 7.2.4}} \partial_j h(\mathbf{y}_\downarrow) = \partial_j h^*(\mathbf{y})$$

If $j = r$, then either $\mathbf{y}_r \geq \frac{n}{2} - \frac{g}{4}$ and then $\partial_r h^*(\vec{x}) \geq \partial_r h^*(\mathbf{y})$ since the RHS is -1 and the LHS is at least that. Or, both $\vec{x}_r, \mathbf{y}_r < \frac{n}{2} - \frac{g}{4}$, and thus $\partial_r h^*(\vec{x}) = \partial_r h(\vec{x}) \underbrace{\geq}_{\text{Lemma 7.2.4}} \partial_r h(\mathbf{y}) = \partial_r h^*(\mathbf{y})$. □

7.2.4. Minimizers

Lemma 7.2.8. *Suppose the parameters n, g and r chosen such that $5gr \leq n$. Let $P = (P_1, \ldots, P_r)$ be any partition with $|P_i| = n$ for all i. Let $f_{\mathcal{P}}$ be the partition submodular function induced by $(P; h)$ and let f_P^* be the partition submodular function induced by $(P; h^*)$. Then, $\emptyset$ is the unique minimizer of $f_{\mathcal{P}}$ achieving the value 0, and*[a] *$f_P^*(P_r) \leq -\frac{g}{2}$.*

[a] In fact, one can show P_r is the unique minimizer of f_P^*, but that is not needed for the lower bound.

Proof. It is obvious that $f_{\mathcal{P}}(\emptyset) = f_P^*(\emptyset) = h(0, 0, \ldots, 0) = 0$. Next, observe that

$$f_P^*(P_r) = h^*(0, 0, \ldots, n) = h\left(0, 0, \ldots, 0, \frac{n}{2} - \frac{g}{4}\right) - \left(\frac{n}{2} + \frac{g}{4}\right)$$

If we let $\mathbf{z} = (0, 0, \ldots, 0, \frac{n}{2} - \frac{g}{4})$, then just using $h(\mathbf{z}) \leq \|\mathbf{z}\|_1$, we get $f_P^*(P_r) \leq -\frac{g}{2}$. Indeed, when $r \geq 3$, this is an equality since then $\ell_t(\mathbf{z}) \leq 0$ for all t and $h(\mathbf{z}) = \|\mathbf{z}\|_1$.

Next, we establish that if $5gr \leq n$, then the minimum value $f_{\mathcal{P}}$ takes is indeed 0.

From Lemma 7.2.3, we know that the maximal minimizer of h is a vector $\vec{x}^*$ where $\vec{x}_i^* \in \{0, n\}$ for $1 \leq i \leq r$. Now fix an arbitrary $\vec{x}$ with $\vec{x}_i \in \{0, n\}$ which is different from the all zeros vector. We claim that $h(\vec{x}) > 0$, which would prove the lemma. Let the number of i's with $\vec{x}_i = n$ among the coordinates $\{1, 2, \ldots, r\}$ be $k \geq 1$.

Note that for any $t \leq r$,

$$\ell_t(\vec{x}) = \sum_{i \geq t} \left(\vec{x}_i - \left(\frac{n}{2} - g \right) \right) - \frac{gr}{4} \leq (k - t + 1) \cdot \left(\frac{n}{2} + g \right) - \frac{gr}{4}$$

Therefore, if $\{a, b\}$ are the odd-even index of $\vec{x}$, we get that these ℓ_t values are at most $k \cdot \left(\frac{n}{2} + g \right) - \frac{gr}{4}$ and $(k-1) \cdot \left(\frac{n}{2} + g \right) - \frac{gr}{4}$, respectively, since a and b are distinct (and occurs when $a = 1$ and $b = 2$). Thus,

$$h^*(\vec{x}) = h(\vec{x}) > kn - \max\left(0, k \cdot \left(\frac{n}{2} + g \right) - \frac{gr}{4} \right) - \max\left(0, (k-1) \cdot \left(\frac{n}{2} + g \right) - \frac{gr}{4} \right)$$

If both the max terms in the expression for h turn out to be 0, then since $k \geq 1$, we get $h(\vec{x}) > n$. If only one of them is 0, then we get $h(\vec{x}) > k \left(\frac{n}{2} - g \right) + \frac{gr}{4} > 0$. Otherwise, we get that

$$h^*(\vec{x}) = h(\vec{x}) > kn - (2k-1) \cdot \left(\frac{n}{2} + g \right) - \frac{gr}{2} \underbrace{\geq}_{\text{using } k \leq r} \frac{n}{2} - \frac{5gr}{2} + g \underbrace{>}_{\text{if } 5gr \leq n} 0 \qquad \square$$

7.2.5. Suffix Indistinguishability

We now establish the key property about h and h^* which allows us to prove a polynomial lower bound on the rounds of adaptivity. To do so, we need a definition.

Definition 7.2.9. *For $1 \leq i < r$, a point $\vec{x} \in [n]^r$ is called i-**balanced** if $\vec{x}_i - \frac{g}{8} \leq \vec{x}_j \leq \vec{x}_i + \frac{g}{8}$ for all $j > i$.*

Suffix Indistinguishability asserts that two points $\vec{x}$ and $\vec{x}'$ which are i-balanced, have the same norm, and which agree on the first i coordinates have the same function value. More precisely,

Lemma 7.2.10 (Suffix Indistinguishability)**.** *Let $i < \frac{r}{2}$. If $\vec{x}$ and $\vec{x}'$ are two i-balanced points with $\vec{x}_j = \vec{x}'_j$ for $j \leq i$ and $\|\vec{x}\|_1 = \|\vec{x}'\|_1$, then $h^*(\vec{x}) = h^*(\vec{x}') = h(\vec{x}) = h(\vec{x}')$.*

Proof. We first prove Suffix Indistinguishability for h, and then show that if $i < \frac{r}{2}$, then h and h^* take the same value on i-balanced points, which implies Suffix Indistinguishability for h^* as well (for $i < \frac{r}{2}$).

Claim 7.2.11. *Let $i \leq r-2$. If $\vec{x}$ and $\vec{x}'$ are two i-balanced points with $\vec{x}_j = \vec{x}'_j$ for $j \leq i$ and $\|\vec{x}\|_1 = \|\vec{x}'\|_1$, then $h(\vec{x}) = h(\vec{x}')$.*

Proof. First note that for any $t \in \{1, 2, \ldots, i+1\}$, $\ell_t(\vec{x}) = \ell_t(\vec{x}')$; this follows from the fact that $\|\vec{x}\|_1 = \|\vec{x}'\|_1$ and that $\vec{x}$ and $\vec{x}'$ agree on the first i-coordinates.

Case 1: $\vec{x}_i = \vec{x}'_i < \frac{n}{2} - \frac{7g}{8}$. Since $\vec{x}$ and $\vec{x}'$ are both i-balanced, we have $\vec{x}_j, \vec{x}'_j < \frac{n}{2} - \frac{7g}{8} + \frac{g}{8} = \frac{n}{2} - \frac{3g}{4}$ for all $j \geq i$. This, in turn, implies that for any $t \geq i$, $\ell_t(\vec{x}), \ell_t(\vec{x}')$ are both $\leq \frac{gr}{4} - \frac{gr}{4} = 0$, since each summand in the definition (7.4) contributes at most $\frac{g}{4}$. So the largest odd (similarly, even) indexed $\ell_t(\vec{x})$ is either negative in which case it contributes 0 to $h(\vec{x})$, or $t \in \{1, \ldots, i+1\}$ in which case it subtracts $\ell_t(\vec{x}) = \ell_t(\vec{x}')$ from $\|\vec{x}\|_1 = \|\vec{x}'\|_1$. Furthermore, in the latter case, the same t is the maximize for $\vec{x}'$ as well. Therefore, in either case, $h(\vec{x}) = h(\vec{x}')$.

Case 2: $\vec{x}_i = \vec{x}'_i \geq \frac{n}{2} - \frac{7g}{8}$. Since $\vec{x}$ and $\vec{x}'$ are both i-balanced, we have $\vec{x}_j, \vec{x}'_j \geq \frac{n}{2} - g$ for all $j \geq i$. Thus each term in the summands of (7.4) is ≥ 0. This, in turn implies that both the odd and the even maximizers of $\ell_t(\vec{x}), \ell_t(\vec{x}')$, lie in $\{1, 2, \ldots, i+1\}$. Since $\ell_t(\vec{x}) = \ell_t(\vec{x}')$ for all such t's and $\|\vec{x}\|_1 = \|\vec{x}'\|_1$, we get that $h(\vec{x}) = h(\vec{x}')$. □

Next, we prove that when i is bounded way from r, for any i-balanced vector $\vec{x}$, we have $h^*(\vec{x}) = h(\vec{x})$. This lemma is useful to prove the indistinguishability of h^* and h.

Claim 7.2.12. *If $i < \frac{r}{2}$ and $\vec{x}$ is i-balanced, then $h^*(\vec{x}) = h(\vec{x})$.*

Proof. If $\vec{x}_r \leq \frac{n}{2} - \frac{g}{4}$, we have $h^*(\vec{x}) = h(\vec{x})$ by definition. So we only need to consider the case when $\vec{x}_r \geq \frac{n}{2} - \frac{g}{4}$. Let $k := \vec{x}_r - \left(\frac{n}{2} - \frac{g}{4}\right)$, by definition $\|\vec{x}\|_1 = \|\vec{x}_{\downarrow}\|_1 + k$ and

$h^*(\vec{x}) = h(\vec{x}_\downarrow) - k$. For any $1 \le t \le r$, we have $\ell_t(\vec{x}) = \ell_t(\vec{x}_\downarrow) + k$, which means that the odd (respectively, even) index t with largest $\ell_t(\vec{x})$ is the same for $\ell_t(\vec{x}_\downarrow)$. That is the odd-even index $\{a, b\}$ is the same for $\vec{x}$ and $\vec{x}_\downarrow$.

Since $\vec{x}$ is i-balanced and $\vec{x}_r \ge \frac{n}{2} - \frac{g}{4}$, we have $\vec{x}_i \ge \frac{n}{2} - \frac{3g}{8}$, and thus, for any $j \ge i$, $\vec{x}_j \ge \frac{n}{2} - \frac{g}{2}$. Thus, all summands in (7.4) for $j \ge i$ give non-negative contribution. This means both a and b lie in $\{1, 2, \ldots, i+1\}$. On the other hand, both $\ell_i(\vec{x}_\downarrow)$ and $\ell_{i+1}(\vec{x}_\downarrow)$ are at least $(r-i-1)\frac{g}{2} - \frac{gr}{4} \ge 0$ since $i \le \frac{r}{2} - 1$. So both $\ell_a(\vec{x}_\downarrow)$ and $\ell_b(\vec{x}_\downarrow)$ are at least 0, which implies that both $\ell_a(\vec{x})$ and $\ell_b(\vec{x})$ are at least k (we only need they are ≥ 0). Therefore, we have

$$h^*(\vec{x}) = h(\vec{x}_\downarrow) - k = (\|\vec{x}_\downarrow\|_1 - \ell_a(\vec{x}_\downarrow) - \ell_b(\vec{x}_\downarrow)) - k = \|\vec{x}\|_1 - \ell_a(\vec{x}) - \ell_b(\vec{x}) = h(\vec{x}). \quad \square$$

Claim 7.2.11 and Claim 7.2.12 implies the Suffix Indistinguishability property of h^* and h. $\square$

7.3. Parallel SFM Lower bound : Proof of Theorem 7.1

We now prove lower bounds on the rounds-of-adaptivity for algorithms which make $\le N^c$ queries per round for some $1 \le c \le N^{1-\delta}$ where $\delta > 0$ is a constant. Let n be an even integer and g be an integer divisible by 4 such that $800\sqrt{cn \log n} \ge g \ge 200\sqrt{cn \log n}$. Let r be the largest odd integer such that $5gr \le n$. Finally, let $N = nr$. Note that $g = \Theta(N^{1/3}(c \log N)^{2/3})$, $r = \Theta\left(\frac{N^{1/3}}{(c \log N)^{1/3}}\right)$, and $n = \Theta(N^{2/3}(c \log N)^{1/3})$. Since $c \le N^{1-\delta}$, we get $n > cN^{2\delta/3} > c \log N$ and thus $g \ge 200c \log n$.

Remark 7.3.1. It is perhaps worth reminding that we are allowing the algorithm to query $N^{N^{1-\delta}}$ sets. A reader may wonder with these many queries available won't one be able to find the minimizer by brute force even in a single round. In the "hard functions" we construct, the minimizer has $n \approx N^{1-\frac{\delta}{3}} \gg N^{1-\delta}$ elements. And thus $N^{N^{1-\delta}}$ queries would not be able to find the minimizer by enumeration over $\approx N^n$ sets.

Let $\mathcal{P} = (P_1, \ldots, P_r)$ be a random equipartition of a universe U of N elements into parts of size n. Given a subset S, let the r-dimensional vector $\vec{x}$ defined as $\vec{x}_i := |S \cap P_i|$ be the signature of S with respect to $\mathcal{P}$. We say a query S is *i-balanced* with respect to $\mathcal{P}$ if the associated signature $\vec{x}$ is i-balanced. We use the following simple property of a random equipartition.

Lemma 7.3.2. *For any integer $i \in [1, \ldots, (r-1)]$, let $P_1, P_2, ..., P_{i-1}$ be a sequence of $(i-1)$ sets each of size n such that for $1 \leq j \leq (i-1)$, the set P_j is generated by choosing uniformly at random n elements from $U \setminus (P_1 \cup P_2 \cup ...P_{j-1})$. Let $S \subset U$ be any query that is chosen with possibly complete knowledge of $P_1, P_2, ..., P_{i-1}$. Then if we extend $P_1, P_2, ..., P_{i-1}$ to a uniformly at random equipartition $(P_1, ..., P_r)$ of U, with probability at least $1 - 1/n^{2c+3}$, the query S is i-balanced with respect to the partition $(P_1, P_2, ..., P_r)$; here the probability is taken over the choice of $P_i, P_{i+1}, ..., P_r$.*

Proof. Let $V = U \setminus (P_1 \cup P_2 \cup ... \cup P_{i-1})$. For $i \leq j \leq r$, let X_j be the random variable whose value equals $|S \cap P_j|$, and let $\mu = E[X_j] = |S \cap V|/(r - i + 1) \leq n$. To prove the assertion of the lemma, it is sufficient to show that with probability at least $1 - 1/n^{2c+3}$, we have $|X_j - \mu| \leq g/16$ for any j.

Note that each X_j is a sum of $|V|$ negatively correlated 0/1 random variables. By Chernoff bound for negatively correlated random variables [104, 154], the probability that X_j deviates from its expectation μ by more than $g/16$ is at most $2e^{\max\{-\frac{(g/16)^2}{3\mu}, -(g/16)\}} \leq 2e^{-10c \log n} \leq 1/n^{2c+4}$. By taking a union bound over all $i \leq j \leq r$, with probability at least $1 - 1/n^{2c+3}$, we have $|X_j - \mu| \leq g/16$ for all such j. □

Now we are ready to prove Theorem 7.1 which we restate below for convenience.

Theorem 7.1. *For any constant $\delta > 0$ and any $1 \leq c \leq N^{1-\delta}$, any randomized algorithm for SFM on an N element universe making $\leq N^c$ evaluation oracle queries per round and succeeding with probability $\geq 2/3$ must have $\Omega\left(\frac{N^{1/3}}{(c \log N)^{1/3}}\right)$ rounds-of-adaptivity. This is*

true even when the range of the submodular function is $\{-N, -N+1, \ldots, N-1, N\}$*, and even if the algorithm is only required to output the value of the minimum.*

Proof. We use Yao's minimax lemma. The distribution over hard functions is as follows. First, we sample a random equipartition $\mathcal{P}$ of the U into r parts each of size n. Given $\mathcal{P}$ and a subset S, let $f_{\mathcal{P}}(S) := h(\vec{x})$ and $f^*_{\mathcal{P}}(S) := h^*(\vec{x})$, where $\vec{x}$ is the signature of S with respect to $\mathcal{P}$. Select one of $f_{\mathcal{P}}$ and $f^*_{\mathcal{P}}$ uniformly at random. This fixes the distribution over the functions, and this distribution is offered to a deterministic algorithm. We now prove that any s-round deterministic algorithm with $s < \frac{r}{2}$ fails to return the correct answer with probability $> 1/3$, and this would prove Theorem 7.1. In fact, we prove that with probability $\geq 1 - 1/n$, over the random equipartition $\mathcal{P}$, the deterministic algorithm cannot distinguish between $f_{\mathcal{P}}$ and $f^*_{\mathcal{P}}$, that is, the answers to all the queries made by the algorithm is the same on both functions. This means that the deterministic algorithm errs with probability $\geq \frac{1}{2} \cdot (1 - \frac{1}{n}) > \frac{1}{3}$.

An s-round deterministic algorithm performs a collection of queries $\mathsf{Q}^{(\ell)}$ at every round $1 \leq \ell \leq s$ with $|\mathsf{Q}^{(\ell)}| \leq N^c \leq n^{2c}$. Let $\mathsf{Ans}^{(\ell)}$ denote the answers to the queries in $\mathsf{Q}^{(\ell)}$. The subsets queried in $\mathsf{Q}^{(\ell)}$ is a deterministic function of the answers given in $\mathsf{Ans}^{(1)}, \ldots, \mathsf{Ans}^{(\ell-1)}$. After receiving the answers to the sth round of queries, that is $\mathsf{Ans}^{(s)}$, the algorithm must return the minimizing set S. We now prove that when $\mathcal{P}$ is a random equipartition of U, then with probability $1 - \frac{1}{n}$, the answers $\mathsf{Ans}^{(\ell)}$ given to $\mathsf{Q}^{(\ell)}$ are the same for $f_{\mathcal{P}}$ and $f^*_{\mathcal{P}}$, if $s < \frac{r}{2}$.

We view the process of generating the random equipartition as a game between an adversary and the algorithm where the adversary reveals the parts one-by-one. Specifically, the process of generating the random equipartition will be such that at the start of any round $\ell \in [1, \ldots, s]$, the adversary has only chosen and revealed to the algorithm the parts $P_1, P_2, ..., P_{\ell-1}$, and at this stage, $P_\ell, P_{\ell+1}, ..., P_r$ are equally likely to be any equipartition of $U \setminus (P_1 \cup P_2 \cup ... \cup P_{\ell-1})$ into $(r - \ell + 1)$ parts. By the end of round ℓ, the adversary has committed and revealed to the algorithm the part P_ℓ, and the game continues with

one caveat. In each round, there will be a small probability (at most $1/n^2$) with which the adversary may "fail". This occurs at a round ℓ if any query made by the algorithm *on or before round* ℓ turns out to be not ℓ-balanced with respect to the sampled partition at round ℓ. In that case, the adversary reveals all remaining parts to the algorithm (consistent with the answers given thus far), and the game terminates in the current round ℓ itself with the algorithm winning the game (that is, the algorithm can distinguish between $f_{\mathcal{P}}$ and $f^*_{\mathcal{P}}$). The probability of this failure event can be bound by $s/n^2 \leq 1/n$, summed over all rounds. In absence of this failure event, by Lemma 7.2.10, we know that the answers will be the same for $f_{\mathcal{P}}$ and $f^*_{\mathcal{P}}$ at the end of the algorithm, concluding the proof. We now formally describe this process.

At the start of round 1, the adversary samples a uniformly at random equipartition of U, say, $\Gamma^{(1)} = (P_1^{(1)}, P_2^{(1)}, ..., P_r^{(1)})$. The algorithm reveals its set of queries for round 1, namely, $\mathsf{Q}^{(1)}$. The adversary answers all queries in $\mathsf{Q}^{(1)}$ in accordance with the partition $\Gamma^{(1)}$. By Lemma 7.3.2, since $|\mathsf{Q}^{(1)}| \leq n^{2c}$, every query in $\mathsf{Q}^{(1)}$ is 1-balanced with respect to the partition $\Gamma^{(1)}$, with probability at least $1 - 1/n^3$. If this event occurs, the adversary reveals $P_1^{(1)}$ to the algorithm, and continues to the next round. Otherwise, the adversary reveals the entire partition $\Gamma^{(1)}$ to the algorithm and the game terminates.

At the start of round 2, the adversary samples another uniformly at random equipartition of U, say, $\Gamma^{(2)} = (P_1^{(2)}, P_2^{(2)}, ..., P_r^{(2)})$ subject to the constraint $P_1^{(2)} = P_1^{(1)}$. Note that $\Gamma^{(2)}$ is a uniformly at random equipartition of U since $P_1^{(1)}$ was chosen uniformly at random. The algorithm reveals its set of queries for round 2, namely, $\mathsf{Q}^{(2)}$. Again by Lemma 7.3.2, we have that (i) every query in $\mathsf{Q}^{(1)}$ is 1-balanced with respect to the partition $\Gamma^{(2)}$, with probability at least $1 - 1/n^3$, and (ii) every query in $\mathsf{Q}^{(2)}$ is 2-balanced with respect to the partition $\Gamma^{(2)}$, with probability at least $1 - 1/n^3$. If this event occurs, the adversary answers all queries in $\mathsf{Q}^{(2)}$ in accordance with the partition $\Gamma^{(2)}$, and the game proceeds to the next round. The key insight here is that by Lemma 7.2.10, if a query $S \in \mathsf{Q}^{(i)}$ is i-balanced w.r.t. some partition $(P_1, ..., P_r)$, then the function value on the query S is completely determined

by $P_1, P_2, ..., P_i$ and $|S|$, and does not require knowledge of $P_{i+1}, ..., P_r$. Furthermore, the value of $f_{\mathcal{P}}(S)$ *and* $f^*_{\mathcal{P}}(S)$ are the same. In other words, the function value on query S remains unchanged, for both f and f^*, if we replace $P := (P_1, ..., P_i, P_{i+1}, ..., P_r)$ with any other partition $P' := (P_1, ..., P_i, P'_{i+1}, .., P'_r)$ such that S remains i-balanced with respect to P'. So answers to all queries in $\mathsf{Q}^{(1)}$ are the same under both partitions $\Gamma^{(1)}$ and $\Gamma^{(2)}$. On the other hand, if either (i) or (ii) above does not occur, the adversary terminates the game and reveals the entire partition $\Gamma^{(1)}$ to the algorithm.

In general, if the game has successfully reached round $\ell \leq s$, then at the start of round ℓ, the adversary samples a uniformly at random equipartition of U, say, $\Gamma^{(\ell)} = (P_1^{(\ell)}, P_2^{(\ell)}, ..., P_r^{(\ell)})$ subject to the constraints $P_1^{(\ell)} = P_1^{(1)}, P_2^{(\ell)} = P_2^{(2)}, ..., P_{\ell-1}^{(\ell)} = P_{\ell-1}^{(\ell-1)}$. Once again, note that $\Gamma^{(\ell)}$ is a uniformly at random equipartition of U since $P_1^{(1)}$ was chosen uniformly at random, $P_2^{(2)}$ was chosen uniformly at random having fixed $P_1^{(1)}$, and so on. The algorithm now reveals its set of queries for round ℓ, namely, $\mathsf{Q}^{(\ell)}$. By Lemma 7.3.2, we have that for any fixed $i \in [1, \ldots, \ell]$, all queries in $\mathsf{Q}^{(i)}$ are i-balanced with respect to the partition $\Gamma^{(\ell)}$ with probability at least $1 - 1/n^3$ each. Thus with probability at least $1 - \ell/n^3$, for every $i \in [1, \ldots, \ell]$, all queries in $\mathsf{Q}^{(i)}$ are i-balanced with respect to the partition $\Gamma^{(\ell)}$. If this event occurs, the adversary answers all queries in $\mathsf{Q}^{(\ell)}$ with respect to the partition $\Gamma^{(\ell)}$, and once again, by Lemma 7.2.10, answers to all queries in $\mathsf{Q}^{(1)}, \mathsf{Q}^{(2)}, ..., \mathsf{Q}^{(\ell-1)}$ remain unchanged if we answer them using the partition $\Gamma^{(\ell)}$. The game then continues to the next round. Otherwise, with probability at most $\ell/n^3 \leq 1/n^2$, the game terminates and the adversary reveals the entire partition $\Gamma^{(\ell-1)}$ to the algorithm.

Summing up over all rounds 1 through $s \leq \frac{r}{2} - 1$, the probability that the game reaches round s is at least $1 - s/n^2 \geq 1 - 1/n$. This, in turn, implies that with probability $\geq 1 - \frac{1}{n}$, the random equipartition $\mathcal{P}$ satisfies the following property : all the queries in $\mathsf{Q}^{(i)}$ are i-balanced with respect to $\mathcal{P}$ for all $i \in [1..s]$. Now, since $s \leq \frac{r}{2}$, by Claim 7.2.12 we get that the answers $\mathsf{Ans}^{(1)}, \ldots, \mathsf{Ans}^{(s)}$ given to these queries are the same for $f_{\mathcal{P}}$ and $f^*_{\mathcal{P}}$. Hence the algorithm cannot distinguish between these two cases. This completes the proof

of Theorem 7.1.

□

7.3.1. Modification to boost gap : $\Omega(1/\varepsilon)$-lower bound for ε-approximate SFM

An inspection of the proof of Theorem 7.1 shows us that the minimum values of $f_{\mathcal{P}}$ and $f^*_{\mathcal{P}}$ are 0 and $-\frac{g}{2}$ for all $\mathcal{P}$'s (by Lemma 7.2.8). That is, any polynomial query algorithm making fewer than $\tilde{\Omega}(N^{1/3})$ rounds of adaptivity cannot distinguish between the case when the minimum value is 0 and minimum value is $-g/2$. Since $g = \Theta(N^{1/3}(c\log N)^{2/3})$, we also rule out *additive* $O(N^{1/3})$-approximations for submodular functions whose range is $\{-N, -N+1, \ldots, N\}$. Scaling such that the range is $[-1, +1]$, we in fact obtain an $\widetilde{\Omega}(1/\sqrt{\varepsilon})$-dept lower bound to obtain ε-additive approximation algorithms.

In this section we show how a small modification leads to indistinguishability between functions with minimum value 0 and those with minimum value $-\Theta(N^{2/3})$ thus proving an $\widetilde{\Omega}(\frac{1}{\varepsilon})$ lower bound on the depth required for polynomial query ε-additive approximation algorithms for SFM.

The difference is in the definition of h^*; we redefine it such that the minimizer is not just P_r (or rather $(0, 0, \ldots, 0, n)$) but $P_{\frac{2r}{3}} \cup P_{\frac{2r}{3}+1} \cup \cdots \cup P_r$, and the minimum value becomes $-\frac{gr}{6} = -\Theta(N^{2/3})$. However, it still remains indistinguishable from h if the number of rounds is $< r/2$, and thus the proof of Theorem 7.1 carries word-to-word.

Define $\vec{x}_{\downarrow} := \left(\vec{x}_1, \ldots, \vec{x}_{\frac{2r}{3}-1}, \min(\vec{x}_{\frac{2r}{3}}, \frac{n}{2} - \frac{g}{4}), \min(\vec{x}_{\frac{2r}{3}+1}, \frac{n}{2} - \frac{g}{4}) \ldots, \min(\vec{x}_r, \frac{n}{2} - \frac{g}{4})\right)$. Then,

$$h^{**}(\vec{x}) = h(\vec{x}_{\downarrow}) - \sum_{i=\frac{2r}{3}}^{r} \max\left(0, \vec{x}_i - \left(\frac{n}{2} - \frac{g}{4}\right)\right) \tag{7.11}$$

Below we note the relevant changes. Let $f^{**}_{\mathcal{P}}$ be the partition submodular function induced by a partition $P = (P_1, \ldots, P_r)$ with $|P_i| = n$, and h^{**}.

- The proof of Lemma 7.2.7 generalizes to prove h^{**} is partition submodular. The

two cases are $j < \frac{2r}{3}$ and $j \geq \frac{2r}{3}$. In the former case, $\partial_j h^{**}(\vec{x}) = \partial_j h(\vec{x}_\downarrow)$ and $\partial_j h^{**}(\mathbf{y}) = \partial_j h(\mathbf{y}_\downarrow)$, and submodularity follows from submodularity of h. If $j \geq \frac{2r}{3}$ and $\mathbf{y}_j \geq \frac{n}{2} - \frac{g}{4}$, then $\partial_j h^{**}(\mathbf{y}) = -1$ which implies it's $\leq \partial_j h^{**}(\vec{x})$. Otherwise, both $\vec{x}_j, \mathbf{y}_j < \frac{n}{2} - \frac{g}{4}$, and then submodularity again follows from that of h.

- In Lemma 7.2.8, we can now assert $f^{**}_{\mathcal{P}}(P_{\frac{2r}{3}} \cup \cdots \cup P_r) = -\frac{g}{2} \cdot \frac{r}{3} = -\frac{gr}{6}$.

- We assert that Lemma 7.2.10 still holds. To see this, note that the only changes are in the proof of Claim 7.2.12 (not the statement), and we sketch this below. Let $k := \sum_{i=\frac{2r}{3}}^{r} \max\left(0, \vec{x}_{\frac{2r}{3}} - \left(\frac{n}{2} - \frac{g}{4}\right)\right)$; we (still) have $\|\vec{x}\|_1 = \|\vec{x}_\downarrow\|_1 + k$ and $h^{**}(\vec{x}) = h(\vec{x}_\downarrow) - k$. Furthermore, for any $1 \leq t \leq \frac{2r}{3}$, we have $\ell_t(\vec{x}) = \ell_t(\vec{x}_\downarrow) + k$, and so if the odd-even index $\{a, b\}$ of $\vec{x}$ is in $\{1, \ldots, \frac{2r}{3}\}$, then $\{a, b\}$ is also the odd-even index for $\vec{x}_\downarrow$.

  Now, if $\vec{x}_t \leq \frac{n}{2} - \frac{g}{4}$ for all $\frac{2r}{3} \leq t \leq r$, then $\vec{x}_\downarrow = \vec{x}$ and $k = 0$ and $h^{**}(\vec{x}) = h(\vec{x})$. So, we may assume that some $\vec{x}_t > \frac{n}{2} - \frac{g}{4}$. And since $\vec{x}$ is i-balanced (for $i < t$), we get (just as in the previous proof) $\vec{x}_j \geq \frac{n}{2} - \frac{g}{2}$ for all $j \geq i$. And thus, the odd-even index $\{a, b\}$ of $\vec{x}$ lies in $\{1, 2, \ldots, i+1\}$. The rest of the proof now proceeds exactly as in Claim 7.2.12.

7.4. Suffix Functions, Nested Matroids, and Parallel Matroid Intersection

In this section we explain how our suffix functions, and as a result our partition submodular functions, arise in the context of matroid intersection. This is then used to prove Theorem 7.2 which states that any efficient matroid intersection algorithm, even with access to *rank* functions to the two matroids, must proceed in polynomially many rounds.

Matroids. A matroid $\mathcal{M} = (U, \mathcal{I})$ is a set-system over a universe U satisfying the following two axioms

- $I \in \mathcal{I}$ and $J \subseteq I$ implies $J \in \mathcal{I}$.
- For any $I, J \in \mathcal{I}$ with $|I| < |J|$, there exists $x \in J \setminus I$ such that $I + x \in \mathcal{I}$.

The sets in $\mathcal{I}$ are called *independent* sets of the matroid. A maximal independent set is called a *base*. It is well-known that all bases have the same cardinality. There are two usual oracles to access matroids. The first is the ***independence oracle*** which given a subset $S \subseteq U$ returns whether S is independent or not. The second stronger oracle, and we assume an algorithm has access to this, is the ***rank oracle*** which given a subset S returns $\mathsf{rk}_{\mathcal{M}}(S)$ which is the cardinality of the largest independent subset of S. It is well known that $\mathsf{rk}(S)$ is a submodular function whose marginals are in $\{0, +1\}$.

Nested Matroids. Let $\mathcal{C} = \{U = C_1 \supseteq C_2 \supseteq \cdots \supseteq C_r\}$ be a collection of nested subsets of the universe U. Let each set C_i have an associated non-negative integer capacity cap_i. Let $\vec{\mathsf{cap}} = (\mathsf{cap}_1, \ldots, \mathsf{cap}_r)$ be the capacity vector. Then $(\mathcal{C}, \vec{\mathsf{cap}})$ defines the following set family which is a matroid. Such matroids are called *nested matroids* (see, for example, [119]) and are a special class of laminar matroids.

$$\mathcal{M}_{\mathcal{C}} := \{I \subseteq U \ : \ |I \cap C_t| \leq \mathsf{cap}_t, \quad 1 \leq t \leq r\} \qquad \text{(Nested Matroids)}$$

Given the nested family $\mathcal{C}$, there is an obvious associated partition $\mathcal{P} := (P_1, P_2, \ldots, P_r)$ of the universe U defined as $P_r := C_r$, the minimal subset in $\mathcal{C}$, and $P_j := C_j \setminus C_{j+1}$ for all $1 \leq j < r$. Similarly, we define "thresholds" for each part of the partition $\mathcal{P}$ as $\tau_r := \mathsf{cap}_r$, and $\tau_j := \mathsf{cap}_j - \mathsf{cap}_{j+1}$. We use $\vec{\tau}$ to denote the threshold vector $(\tau_1, \ldots, \tau_r)$.

Observe that these definitions are interchangeable : given $(\mathcal{P}, \vec{\tau})$ one gets the nested matroid defined by $(\mathcal{C}, \vec{\mathsf{cap}})$, where $C_j = \bigcup_{t \geq j} P_t$ for all $1 \leq j \leq r$, and $\mathsf{cap}_j = \sum_{t \geq j} \tau_t$.

Rank of a Nested Matroid. Given a nested matroid $\mathcal{M}$, let $\mathcal{P} = (P_1, \ldots, P_r)$ be the associated partition with thresholds τ_1 to τ_r. For simplicity, let us assume $|P_i| = n$ for all $1 \leq i \leq r$. Given a subset $S \subseteq U$, let $\vec{x} \in \mathbb{Z}^r_{\geq 0}$ be the signature of S where $\vec{x}_i := |P_i \cap S|$. Define

$$\text{for any } 1 \leq t \leq r, \quad \ell_t(\vec{x}) := \sum_{s=t}^{r} (\vec{x}_s - \tau_s) \tag{7.12}$$

Note that a set S is independent if and only if $\ell_t(\vec{x}) \leq 0$ for all $1 \leq t \leq r$. Also note the connection with (7.4) when we set $\tau_1 = \cdots = \tau_{r-1} = \left(\frac{n}{2} - g\right)$ and $\tau_r = \left(\frac{n}{2} - g\right) + \frac{gr}{4}$. The next lemma shows how these functions define the rank of a nested matroid.

Lemma 7.4.1 (Rank of a Nested Matroid)**.**

Let $\mathcal{M}$ be a nested matroid defined by $(\mathcal{P} = (P_1, \ldots, P_r); \vec{\tau} = (\tau_1, \ldots, \tau_r))$ where $\tau_i \geq 0$ for all i. Given any subset $S \subseteq U$ with signature $\vec{x}$, the rank of S is

$$\mathsf{rk}_{\mathcal{M}}(S) = \|\vec{x}\|_1 - \max\left(0, \max_{1 \leq a \leq r} \ell_a(\vec{x})\right)$$

where $\ell_t(\vec{x})$ is as defined in (7.12).

Proof. The rank $\mathsf{rk}_{\mathcal{M}}(S)$, which we also denote as $\mathsf{rk}_{\mathcal{M}}(\vec{x})$, is the cardinality of the largest independent subset of S. This value can be found by the following linear program, which is integral because the constraint matrix is totally unimodular.

$$\begin{aligned}
\mathsf{rk}(\vec{x}) := \max \sum_{i=1}^{r} \mathbf{y}_i & & & & \min \sum_{i=1}^{r} \eta_i \vec{x}_i + \sum_{t=1}^{r} \mathbf{z}_t \cdot \left(\sum_{i \geq t} \tau_i\right) & \\
\mathbf{y}_i \leq \vec{x}_i, & \quad \forall i \in [r] & \overset{=}{\underset{\text{Duality}}{}} & & \sum_{t \leq i} \mathbf{z}_t + \eta_i = 1, & \quad \forall i \in [r] \\
\sum_{i \geq t} \mathbf{y}_i \leq \sum_{i \geq t} \tau_i, & \quad \forall t \in [r] & & & \mathbf{z}, \eta \geq 0 &
\end{aligned}$$

We do not impose non-negativity constraints on the $\mathbf{y}_i$ variables in the primal because the maximizing solution will indeed have non-negative $\mathbf{y}_i$'s. To see this, suppose $\mathbf{y}_j < 0$ and let $t \leq j$ be the largest index such that $\sum_{i \geq t} \mathbf{y}_i = \sum_{i \geq t} \tau_i$. That is, the largest indexed constraint, among the ones containing $\mathbf{y}_j$, which is tight. There must be such a t for otherwise we could increase the objective by incrementing $\mathbf{y}_j$. Furthermore, $\mathbf{y}_t > 0$ for otherwise $\sum_{i \geq t+1} \mathbf{y}_i = \sum_{i \geq t+1} \tau_i$ and our t won't be largest; this argument uses $\tau_t \geq 0$. Now, increasing $\mathbf{y}_j$ and decreasing $\mathbf{y}_t$ by the same amount gives a feasible solution with the same optimum, and continuing the above procedure, we will get to a non-negative $\mathbf{y}$.

We can massage the dual as follows. Let $\mathsf{pref}_i(\mathbf{z}) := \sum_{t \le i} \mathbf{z}_t$. Thus, we can rewrite $\eta_i = 1 - \mathsf{pref}_i(\mathbf{z})$, and since $\eta_i \ge 0$, we get all $\mathsf{pref}_i(\mathbf{z})$'s, and in particular which is equivalent to, by the non-negativity of $\mathbf{z}$, the constraint $\|\mathbf{z}\|_1 \le 1$. Therefore, we can eliminate η's and get

$$\mathsf{rk}(\vec{x}) = \min_{\mathbf{z}:\|\mathbf{z}\|_1 \le 1} \sum_{i=1}^{r} \vec{x}_i \cdot (1 - \mathsf{pref}_i(\mathbf{z})) + \sum_{t=1}^{r} \mathbf{z}_t \left(\sum_{i \ge t} \tau_i \right)$$

Next, using the observation that $\sum_{t=1}^{r} \mathbf{z}_t \left(\sum_{i \ge t} \tau_i \right) = \sum_{i=1}^{r} \mathsf{pref}_i(\mathbf{z}) \cdot \tau_i$, we can further simplify to get

$$\mathsf{rk}(\vec{x}) = \min_{\mathbf{z}:\|\mathbf{z}\|_1 \le 1} \sum_{i=1}^{r} \vec{x}_i - \sum_{i=1}^{r} \mathsf{pref}_i(\mathbf{z}) \cdot (\vec{x}_i - \tau_i) = \|\vec{x}\|_1 - \max_{\mathbf{z}:\|\mathbf{z}\|_1 \le 1} \sum_{t=1}^{r} \mathbf{z}_t \underbrace{\left(\sum_{i \ge t} (\vec{x}_i - \tau_i) \right)}_{\ell_t(\vec{x})}$$

The last summand $\max_{\mathbf{z}:\|\mathbf{z}\|_1 \le 1} \sum_{t=1}^{r} \mathbf{z}_t \ell_t(\vec{x})$ is 0 if all $\ell_t(\vec{x}) \le 0$ (by setting $\mathbf{z} \equiv \mathbf{0}$), and otherwise, it is $\max_{1 \le a \le t} \ell_a(\vec{x})$. This completes the proof. □

The reader should notice the similarity with (7.5). We will now make the connection more precise. Before doing so, we need another well known definition.

Duals of Matroids. Given a matroid $\mathcal{M}$, the dual matroid $\mathcal{M}^*$ is defined as follows

$$\mathcal{I}^* := \{S \subseteq U \ : \ U \setminus S \ \text{ contains a base of } \ \mathcal{M}\}$$

It is not too hard to check this is a matroid. The rank of any set in the dual matroid can be computed using the rank of the original matroid as follows.

Lemma 7.4.2 (e.g., Theorem 39.3 in [237])**.** *Let $\mathcal{M}$ be a matroid with rank function rk. Let $\mathcal{M}^*$ be its dual with corresponding rank function rk^*. Then,*

$$\forall S \subseteq U : \quad \mathsf{rk}^*(S) = \mathsf{rk}(U \setminus S) + |S| - \mathsf{rk}(U)$$

It is not too hard to see that the dual of a nested matroid is another nested matroid whose nesting is from the "other end". More formally, one can prove the following.

Lemma 7.4.3. *Let $\mathcal{M}$ be a nested matroid defined by the partition $\mathcal{P} = (P_1, \ldots, P_r)$ and thresholds $\vec{\tau} := (\tau_1, \ldots, \tau_r)$. Then, $\mathcal{M}^*$ is another nested matroid defined by the* reverse *partition $\mathcal{P}' = (P_r, P_{r-1}, \ldots, P_2, P_1)$ and thresholds $\vec{\tau}' := (n_r - \tau_r, n_{r-1} - \tau_{r-1}, \ldots, n_1 - \tau_1)$, where $n_i := |P_i|$.*

Proof. Let S be a subset with signature $\vec{x}$ with respect to the original partition $\mathcal{P}$. S is independent in $\mathcal{M}^*$ if and only if $U \setminus S$ contains a base of $\mathcal{M}$. Equivalently, $\mathsf{rk}_{\mathcal{M}}(U \setminus S) = \mathsf{rk}_{\mathcal{M}}(U)$. Now, the latter is precisely $\|\mathbf{n}\|_1 - \ell_1(\mathbf{n})$ where $\mathbf{n} = (n_1, n_2, \ldots, n_r)$ is the signature of the universe U. Let $\mathbf{z}$ be the signature of $U \setminus S$; note that $\mathbf{z}_i = n_i - \vec{x}_i$. Thus, we get that S is independent in $\mathcal{M}^*$ if and only if

$$\|\mathbf{z}\|_1 - \max(0, \max_{1 \le a \le r} \ell_a(\mathbf{z})) = \|\mathbf{n}\|_1 - \ell_1(\mathbf{n}) \underbrace{\Rightarrow}_{\text{Rearranging}} \ell_1(\mathbf{z}) = \max(0, \max_{1 \le a \le r} \ell_a(\mathbf{z}))$$

$\ell_1(\mathbf{z})$ is largest suffix if and only if all the $(r-1)$ *prefix-sums* are non-negative, and $\ell_1(\mathbf{z}) \ge 0$ implies all prefix-sums are non-negative. Thus, we get

$$\forall 1 \le j \le r, \ \sum_{j \le t} (\mathbf{z}_j - \tau_j) \ge 0 \quad \equiv \quad \forall 1 \le j \le r, \ \sum_{j \le t} (\vec{x}_j - (n_j - \tau_j)) \le 0$$

which is precisely the signature of an independent set in the nested matroid defined by $(\mathcal{P}', \vec{\tau}')$. □

The Hard Matroid Intersection Set-up. Let $r = 2k + 1$ be an odd number. Let $\mathcal{P} = (P_1, \ldots, P_r)$ be a partition with $|P_i| = n$. Each part will be associated with a parameter τ_i. These will be set to $\tau_1 = \cdots = \tau_{r-1} = \left(\frac{n}{2} - g\right)$ and $\tau_r = \left(\frac{n}{2} - g\right) + \frac{gr}{4}$, where $g, \frac{gr}{4}$ are as described in Section 7.3.

We define *three* coarsenings of this partition. The first is the *odd* coarsening containing

$(k+1)$ parts defined as follows.

$$\mathcal{P}_{\mathsf{odd}} := (P_1 \cup P_2, \quad P_3 \cup P_4, \quad \ldots, \quad P_{r-2} \cup P_{r-1}, \quad P_r)$$

and the associated τ-values are, as expected, the sum of the relevant τ_j's. More precisely, they are $\vec{\tau}_{\mathsf{odd}} := (\tau_1+\tau_2, \tau_3+\tau_4, \ldots, \tau_{r-2}+\tau_{r-1}, \tau_r)$. Let $\mathcal{M}_{\mathsf{odd}}$ be the nested matroid defined by $(\mathcal{P}_{\mathsf{odd}}, \vec{\tau}_{\mathsf{odd}})$. The rank of $\mathcal{M}_{\mathsf{odd}}$ is given by Lemma 7.4.1 as follows; we only consider the odd indices since r is odd.

Claim 7.4.4. *Let $S \subseteq U$. Let $\vec{x}$ be the signature of S with respect to the $(2k+1)$-part partition $\mathcal{P}$. Then,*

$$\mathsf{rk}_{\mathcal{M}_{\mathsf{odd}}}(\vec{x}) := \mathsf{rk}_{\mathcal{M}_{\mathsf{odd}}}(S) = \|\vec{x}\|_1 - \max\left(0, \max_{1 \le a \le r,\ a\ odd} \ell_a(\vec{x})\right)$$

The second coarsening is the *even* coarsening containing $(k+1)$-parts defined as

$$\mathcal{P}_{\mathsf{even}} := (P_1, \quad P_2 \cup P_3, \quad P_4 \cup P_5, \quad \ldots, \quad P_{r-1} \cup P_r)$$

The associated τ-values are slightly different in that the first part is effectively "ignored". The vector of τ's are $\vec{\tau}_{\mathsf{even}} := (n, \tau_2+\tau_3, \tau_4+\tau_5, \ldots, \tau_{r-1}+\tau_r)$. Let $\mathcal{M}_{\mathsf{even}}$ be the corresponding nested matroid defined by $(\mathcal{P}_{\mathsf{even}}, \vec{\tau}_{\mathsf{even}})$. Note that any base of $\mathcal{M}_{\mathsf{even}}$ must contain the whole set P_1. Again using Lemma 7.4.1, the rank of this matroid is given as follows.

Claim 7.4.5. *Let $S \subseteq U$. Let $\vec{x}$ be the signature of S with respect to the $(2k+1)$-part partition $\mathcal{P}$. Then,*

$$\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}) := \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(S) = \|\vec{x}\|_1 - \max\left(0, \max_{1 \le a \le r,\ a\ even} \ell_a(\vec{x})\right)$$

The reason the first part does not count is because $(\vec{x}_1 - n)$ is ≤ 0, and this cannot be the maximizer when we apply Lemma 7.4.1. And otherwise, it corresponds to an even index in

the original partition.

Finally, the third coarsening is a refinement of $\mathcal{P}_{\mathsf{even}}$ where the last part $P_{r-1} \cup P_r$ is divided into two. That is,

$$\mathcal{P}'_{\mathsf{even}} := (P_1, \quad P_2 \cup P_3, \quad P_4 \cup P_5, \quad \ldots, \quad P_{r-1}, \quad P_r)$$

The associated τ vector is $\vec{\tau}'_{\mathsf{even}} := (n, \tau_2 + \tau_3, \tau_4 + \tau_5, \ldots, \tau_{r-1} + \tau_r - \theta, \theta)$ for some parameter θ, which is set to $\left(\frac{n}{2} - \frac{g}{4}\right)$. Let $\mathcal{M}'_{\mathsf{even}}$ be the nested matroid defined by $(\mathcal{P}'_{\mathsf{even}}, \vec{\tau}'_{\mathsf{even}})$.

Claim 7.4.6. *Let $S \subseteq U$. Let $\vec{x}$ be the signature of S with respect to the $(2k+1)$-part partition $\mathcal{P}$. Then,*

$$\mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x}) := \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(S) = \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}_{\downarrow})$$

where, $\vec{x}_{\downarrow} = (\vec{x}_1, \ldots, \vec{x}_{r-1}, \min(\vec{x}_r, \theta))$.

Proof. First observe that for any t, $\ell_t(\vec{x}) = \ell_t(\vec{x}_{\downarrow}) + \max(0, (\vec{x}_r - \theta))$. Therefore, for any $\vec{x}$, the t maximizing $\ell_t(\vec{x})$ also is the one maximizing $\ell_t(\vec{x}_{\downarrow})$.

When computing $\mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x})$ as $\|\vec{x}\| - \max(0, \max_a \ell_a(\vec{x}))$, the maximization over a is over all even indices and also r. This leads to two cases.

Case 1: This maximizer is at $a = r$, that is, $\mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x}) = \|\vec{x}\|_1 - \max(0, \vec{x}_r - \theta)$. In that case, we have $\ell_a(\vec{x}) \leq (\vec{x}_r - \theta)$ for all other a's. Which implies $\ell_a(\vec{x}_{\downarrow}) \leq 0$. Therefore, $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}_{\downarrow}) = \|\vec{x}_{\downarrow}\|_1 = \|\vec{x}\|_1 - \max(0, \vec{x}_r - \theta) = \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x})$.

Case 2: This maximizer at $a \neq r$, that is, $\mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x}) = \|\vec{x}\|_1 - \max(0, \ell_a(\vec{x}))$ for some even a. Note that this a is also the maximizer when computing $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}_{\downarrow})$. Therefore,

$$\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}_{\downarrow}) = \|\vec{x}_{\downarrow}\|_1 - \max(0, \ell_a(\vec{x}_{\downarrow})) = \|\vec{x}_{\downarrow}\|_1 - \max(0, \ell_a(\vec{x}) - \max(0, \underbrace{(\vec{x}_r - \theta)}_{\ell_r(\vec{x})}))$$

If $\vec{x}_r \leq \theta$, we get $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}_{\downarrow}) = \|\vec{x}_{\downarrow}\|_1 - \max(0, \ell_a(\vec{x})) = \|\vec{x}\|_1 - \max(0, \ell_a(\vec{x})) = \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x})$,

where the second equality follows because $\vec{x}_\downarrow = \vec{x}$ when $\vec{x}_r \leq \theta$.

If $\vec{x}_r > \theta$, then $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}_\downarrow) = \|\vec{x}_\downarrow\|_1 - (\ell_a(\vec{x}) - (\vec{x}_r - \theta))$ since $\ell_a(\vec{x}) \geq \ell_r(\vec{x}) \geq 0$ as a is the maximizer. Now observe that $\|\vec{x}_\downarrow\|_1 = \|\vec{x}\|_1 - (\vec{x}_r - \theta)$, and so $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}_\downarrow) = \|\vec{x}\| - \ell_a(\vec{x}) = \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x})$. □

Claim 7.4.7. $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(U) = \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(U)$.

Proof. Let $\mathbf{n}$ be the $(n, n, \ldots, n)$ vector. $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(U) = \|\mathbf{n}\|_1 - \ell_2(\mathbf{n})$, and $\mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(U) = \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\mathbf{n}_\downarrow)$. This, in turn, is $\|\mathbf{n}_\downarrow\|_1 - \ell_2(\mathbf{n}_\downarrow) = (\|\mathbf{n}\| - (n - \theta)) - (\ell_2(\mathbf{n}) - (n - \theta)) = \|\mathbf{n}\|_1 - \ell_2(\mathbf{n})$. □

The following lemma connects matroid intersection with submodular function minimization for the functions described in Section 7.2.

Lemma 7.4.8. *The size of the largest cardinality independent set in $\mathcal{M}_{\mathsf{odd}} \cap \mathcal{M}^*_{\mathsf{even}}$ is precisely $C + \min_{S \subseteq U} f(S)$ where $C = |U| - \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(U)$ and $f(S) = h(\vec{x})$ with*

$$h(\vec{x}) = \|\vec{x}\|_1 - \max\left(0, \max_{1 \leq a \leq r,\ a\ odd} \ell_a(\vec{x})\right) - \max\left(0, \max_{1 \leq a \leq r,\ a\ even} \ell_a(\vec{x})\right)$$

and the size of the largest cardinality independent set in $\mathcal{M}_{\mathsf{odd}} \cap (\mathcal{M}'_{\mathsf{even}})^*$ is precisely $C + \min_{S \subseteq U} f^*(S)$ where $C = |U| - \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(U) = |U| - \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(U)$ and $f^*(S) = h^*(\vec{x})$ with

$$h^*(\vec{x}) = \begin{cases} h(\vec{x}) & \text{if } \vec{x}_r \leq \theta \\ h(\vec{x}_\downarrow) - (\vec{x}_r - \theta) & \text{otherwise} \end{cases} \quad \text{where, } \vec{x}_\downarrow := (\vec{x}_1, \ldots, \vec{x}_{r-1}, \min(\vec{x}_r, \theta))$$

Proof. From Edmond's theorem [107], we know that for any two matroids $\mathcal{M}_1$ and $\mathcal{M}_2$, one has

$$\max_{I \in \mathcal{M}_1 \cap \mathcal{M}_2} |I| \;=\; \min_{S \subseteq U} \left(\mathsf{rk}_{\mathcal{M}_1}(S) + \mathsf{rk}_{\mathcal{M}_2}(U \setminus S)\right)$$

Fix a set S with signature $\vec{x}$ with respect to the $(2k+1)$-part partition $\mathcal{P}$. By Claim 7.4.4,

we have $\mathsf{rk}_{\mathcal{M}_{\mathsf{odd}}}(S) = \mathsf{rk}_{\mathcal{M}_{\mathsf{odd}}}(\vec{x}) = \|\vec{x}\|_1 - \max(0, \max_{1\le a\le r,\ a\ \text{odd}} \ell_a(\vec{x}))$. By Lemma 7.4.2, we have $\mathsf{rk}_{\mathcal{M}^*_{\mathsf{even}}}(U \setminus S) = \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(S) + |U| - \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(U) - |S| = \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}) + C - \|\vec{x}\|_1$. By Claim 7.4.5, we have $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(S) = \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}) = \|\vec{x}\|_1 - \max(0, \max_{1\le a\le r,\ a\ \text{even}} \ell_a(\vec{x}))$. And thus,

$$\mathsf{rk}_{\mathcal{M}_{\mathsf{odd}}}(S) + \mathsf{rk}_{\mathcal{M}^*_{\mathsf{even}}}(U \setminus S) = C + h(\vec{x})$$

Similarly, by Lemma 7.4.2, we have $\mathsf{rk}_{(\mathcal{M}'_{\mathsf{even}})^*}(U \setminus S) = \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(S) + |U| - \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(U) - |S| = \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x}) + C - \|\vec{x}\|_1$. By Claim 7.4.6, the RHS equals $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}_\downarrow) + C - \|\vec{x}\|_1$. And so,

$$\begin{aligned}
&\mathsf{rk}_{\mathcal{M}_{\mathsf{odd}}}(S) + \mathsf{rk}_{(\mathcal{M}'_{\mathsf{even}})^*}(U \setminus S) \\
&= C + \|\vec{x}_\downarrow\|_1 - \max\left(0, \max_{1\le a\le r,\ a\ \text{odd}} \ell_a(\vec{x})\right) - \max\left(0, \max_{1\le a\le r,\ a\ \text{even}} \ell_a(\vec{x}_\downarrow)\right)
\end{aligned}$$

When $\vec{x}_r \le \theta$, the RHS is $C + h(\vec{x})$. When $\vec{x}_r > \theta$, we have $\ell_t(\vec{x}) = \ell_t(\vec{x}_\downarrow) + (\vec{x}_r - \theta)$ for all t, and as before, one can argue that

$$\max\left(0, \max_{1\le a\le r,\ a\ \text{odd}} \ell_a(\vec{x})\right) = \max\left(0, \max_{1\le a\le r,\ a\ \text{odd}} \ell_a(\vec{x}_\downarrow)\right) + (\vec{x}_r - \theta).$$

Which implies the RHS is $C + h(\vec{x}_\downarrow) - (\vec{x}_r - \theta)$. In sum, the RHS is $C + h^*(\vec{x})$. □

An Illustration. It is perhaps instructive to illustrate the difference in the two situations described in Lemma 7.4.8 with a concrete example which directly describes why the largest cardinality common independent sets are different in the two different cases. Take $r = 3$. Fix a partition (P_1, P_2, P_3) with each part having n elements each, and the size of the universe is $3n$. The τ values are $(\frac{n}{2} - g, \frac{n}{2} - g, \frac{n}{2} - 0.25g)$.

Let us understand what $\mathcal{M}_{\mathsf{odd}}$ is in this case. This is generated by $(P_1 \cup P_2, P_3)$ and the threshold vector $(n - 2g, \frac{n}{2} - 0.25g)$. So, a subset I is independent in $\mathcal{M}_{\mathsf{odd}}$ iff (a) it contains $\le \frac{n}{2} - 0.25g$ elements from P_3, and (b) $\le \frac{3n}{2} - 2.25g$ elements overall.

Similarly, the matroid $\mathcal{M}_{\mathsf{even}}$ is generated by $(P_1, P_2 \cup P_3)$ with the threshold vector $(n, n - 1.25g)$. We are interested in its dual, which is also a nested matroid which, by Lemma 7.4.3

is generated by the partition $(P_2 \cup P_3, P_1)$ with thresholds $(n+1.25g, 0)$. That is, a subset I is independent in $\mathcal{M}^*_{\mathsf{even}}$ iff (a) it contains 0 elements from P_1, and (b) $\leq n+1.25g$ elements overall.

Notice that any set I^* which contains $\frac{n}{2} - 0.25g$ elements from P_3, $\frac{n}{2} + 1.5g$ elements from P_2, and 0 elements from P_1 is a ***base*** of $\mathcal{M}_{\mathsf{even}}$ which is independent in $\mathcal{M}_{\mathsf{odd}}$. All that is needed is that $1.5g \leq \frac{n}{2}$ so that there are enough items in P_2 to pick from.

Finally, let us consider the matroid $(\mathcal{M}'_{\mathsf{even}})$ and its dual. The former is a nested matroid generated by (P_1, P_2, P_3) with thresholds $(n, \frac{n}{2} - g, \frac{n}{2} - 0.25g)$. Which, in turn, implies that its dual is a nested matroid generated by (P_3, P_2, P_1) with thresholds $(\frac{n}{2} + 0.25g, \frac{n}{2} + g, 0)$. That is, an independent set cannot contain more than $\frac{n}{2} + g$ elements from P_2, thus ruling out the I^* described in the previous paragraph. Indeed, since $\mathcal{M}_{\mathsf{odd}}$ forces at most $\frac{n}{2} - 0.25g$ elements from P_3, the largest common independent set in $\mathcal{M}_{\mathsf{odd}}$ and $(\mathcal{M}'_{\mathsf{even}})^*$ is at most of size $n + 0.75g$ elements. Which is exactly $-g/2$ less, as predicted by Lemma 7.4.8 and Lemma 7.2.8. Note, however, that the size of the largest independent set in $(\mathcal{M}'_{\mathsf{even}})^*$ is the same as that in $\mathcal{M}^*_{\mathsf{even}}$, that is $n + 2.75g$; that set picks more elements from P_3. It is the intersection with $\mathcal{M}_{\mathsf{odd}}$ which prevents picking such a base of $(\mathcal{M}'_{\mathsf{even}})^*$.

Now we are ready to prove Theorem 7.2.

Theorem 7.2. *For any constant $\delta > 0$ and any $1 \leq c \leq N^{1-\delta}$, any randomized algorithm for matroid intersection on an N element universe making $\leq N^c$* rank-oracle *queries per round and succeeding with probability $\geq 2/3$ must have $\Omega\left(\frac{N^{1/3}}{(c \log N)^{1/3}}\right)$ rounds-of-adaptivity. This is true even when the two matroids are* nested matroids*, a special class of laminar matroids, and also when the algorithm is only required to output the value of the optimum.*

Proof of Theorem 7.2. To complete the proof of Theorem 7.2, we need one more thing. In SFM, we have access to evaluation oracle for the function. In particular, if $\vec{x}$ is the signature of a set S with respect to a partition, then we have access to $h(\vec{x})$. In the matroid intersection problem, we have access to the individual ranks of each matroid. Therefore,

we need to establish suffix-indistinguishability for each of the individual ranks. Since the rank of the dual matroid can be simulated by the rank of the original matroid, the suffix indistinguishability of both matroids is established by the following lemma whose proof is very similar to that of Lemma 7.2.10.

Lemma 7.4.9. *A signature $\vec{x}$ (with respect to the original $(2k+1)$-part partition) is i-balanced if $\vec{x}_i - \frac{g}{8} \leq \vec{x}_j \leq \vec{x}_i + \frac{g}{8}$. Let $i < \frac{r}{2}$. If $\vec{x}$ and $\vec{x}'$ are two i-balanced points with $\vec{x}_j = \vec{x}'_j$ for $j \leq i$ and $\|\vec{x}\|_1 = \|\vec{x}'\|_1$, then (a) $\mathsf{rk}_{\mathcal{M}_{\mathsf{odd}}}(\vec{x}) = \mathsf{rk}_{\mathcal{M}_{\mathsf{odd}}}(\vec{x}')$, and (b) $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}) = \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}') = \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x}) = \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x}')$*

Proof. As in the proof of Lemma 7.2.10, we proceed in two claims. First, we claim that for any $i \leq r-2$, if $\vec{x}$ and $\vec{x}'$ are i-balanced, then $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}) = \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}')$. If $\vec{x}_i = \vec{x}'_i < \frac{n}{2} - \frac{7g}{8}$, then just as in Claim 7.2.11, all $\vec{x}_j, \vec{x}'_j$, for $j \geq i$, are $\leq \frac{n}{2} - \frac{3g}{4}$, implying that the even-index with the largest $\ell_t(\cdot)$ must lie in $\{1, 2, \ldots, i+1\}$. And this, due to the premise of the lemma, implies (using Claim 7.4.5) $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}) = \mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}')$. A similar argument using odd-index and Claim 7.4.4 proves part (a).

The proof of the second and third equality in part(b) follows as in Claim 7.2.12. We have $\theta = \frac{n}{2} - \frac{g}{4}$. If $\vec{x}_r \leq \theta$, then the two ranks are the same by Claim 7.4.6. If $\vec{x}_r > \theta$, then since $\vec{x}$ is i-balanced, all $\vec{x}_j \geq \frac{n}{2} - \frac{g}{2}$ for $j \geq i$. This means the even index with the largest $\ell_t(\vec{x})$ lies in $\{1, \ldots, i+1\}$. And since $i \leq r/2$, which implies that both $\ell_i(\vec{x}_\downarrow)$ and $\ell_{i+1}(\vec{x}_\downarrow)$ (we look at both for we don't know which is even, but one of them is) are ≥ 0. Therefore, $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}) = \|\vec{x}\|_1 - \ell_a(\vec{x})$ for some even $a \leq i+1$, and $\mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x}) = \|\vec{x}_\downarrow\|_1 - \ell_a(\vec{x}_\downarrow)$ for the same a. Since $a \leq i+1$, we get that $\|\vec{x}_\downarrow\|_1 = \|\vec{x}\|_1 - k$ and $\ell_a(\vec{x}_\downarrow) = \ell_a(\vec{x}) - k$, where $k = \vec{x}_r - \theta$. In sum, we get $\mathsf{rk}_{\mathcal{M}_{\mathsf{even}}}(\vec{x}) = \mathsf{rk}_{\mathcal{M}'_{\mathsf{even}}}(\vec{x})$, and this, together with the previous paragraph, implies part (b). □

The proof of Theorem 7.2 then follows almost word-to-word as the proof of Theorem 7.1. The hard distributions over the pairs of matroids are as follows. First one samples a random equipartition P of U into $(2k+1)$ parts. Given P, the "odd" matroid $\mathcal{M}_{\mathsf{odd}}$ is one nested

matroid. The other nested matroid is either $\mathcal{M}^*_{\mathsf{even}}$ or $(\mathcal{M}'_{\mathsf{even}})^*$. Note that by Lemma 7.4.3, these duals are also nested matroids. We give the algorithm rank-oracle access to these two matroids. As in the proof of Theorem 7.1, armed with Lemma 7.4.9, one can show that for any s-round deterministic algorithm for $s \leq \frac{r}{2} - 1$, with probability $\geq 1 - \frac{1}{n}$, the answers given in the case of $(\mathcal{M}_{\mathsf{odd}}, \mathcal{M}^*_{\mathsf{even}})$ and the answers given in the case of $(\mathcal{M}_{\mathsf{odd}}, (\mathcal{M}'_{\mathsf{even}})^*)$ are exactly the same. Since the *sizes* of the largest common independent sets in both cases are different, one gets the proof of Theorem 7.2. □

CHAPTER 8

Conclusion and Open Problems

In this thesis, we studied several combinatorial problems in different sublinear settings, where the standard algorithms are hard to implement.

In particular, we first considered two classic graph problems: $(\Delta+1)$-coloring problem and the graphical traveling salesman problem. We prove the upper and lower bound for these two problems in different models. For both problems, we first prove a structure-property for the problem, then design sublinear algorithms in different models based on the properties.

We then consider the hypergraph cut sparsifier problem. We prove the existence of a near-linear size cut sparsifier for any hypergraph. We also consider the problem in the sublinear case and give a query algorithm that runs in polynomial time in the number of vertices and is independent of the number of hyperedges in the graph.

We then considered a communication problem we call hidden pointer chasing and gave a communication lower bound. By reduction from this problem, we prove lower bounds on the number of passes we needed to compute maximum flow and lexicographically-first maximal independent set problem using sublinear space.

Finally, we studied the round complexity of the submodular function minimization problem and proved a polynomial lower bound on the number of rounds we need to compute the minimum value of a submodular function in a polynomial number of queries.

At the end of this thesis, we list several open problems that are related to the problems and models we considered in this thesis.

8.1. Sublinear Algorithms for Graph Problems

In Chapter 4, we gave sublinear algorithms that $(2-\varepsilon)$-approximates the cost of a graphical traveling salesman problem. An immediate question to ask is whether there is such an

algorithm for the general metric TSP problem. Since the technique we used in Chapter 4 is heavily relying on the property that the metric is defined by an underlying unweighted graph, we might need completely different ideas to general our algorithms to the general case.

Problem 1. *Are there a constant $\varepsilon > 0$ and a query algorithm that estimates the cost of metric TSP to within a factor of $(2-\varepsilon)$ by performing $o(n^2)$ queries?*

As we discussed in Chapter 4, the cost of a metric minimum spanning tree can be estimated to within a factor of $(1+\varepsilon)$ for any $\varepsilon > 0$ in $\tilde{O}(n)$ queries. Similar to the metric TSP problem, the cost of MST also gives a 2-approximation for the metric Steiner tree problem. Designing sublinear algorithms for breaking the barrier of 2 for the metric TSP problem and the metric Steiner tree problem might be closely related.

Problem 2. *Are there a constant $\varepsilon > 0$ and a query algorithm that estimates the cost of the metric Steiner tree to within a factor of $(2-\varepsilon)$ by performing $o(n^2)$ queries?*

8.2. Linear Size Hypergraph Sparsifier

In Chapter 5, we prove that any hypergraph admits a cut sparsifier with $O(n \log n)$ hyperedges, and it is easy to prove that there are hypergraphs that any sparsifier contains $\Omega(n)$ hyperedges. Thus, the question is, can we prove that any hypergraph admits a cut sparsifier with $O(n)$ hyperedges?

In the case of a normal graph, it is shown that any graph has a cut sparsifier with $O(n)$ edges [41, 195]. This is shown by considering the more generalized spectral sparsifier problem. Thus, to answer the question of the existence of linear size hypergraph cut sparsifier, we might need to consider the hypergraph spectral sparsifier problem. Following our work, it is shown that any hypergraph also has a spectral sparsifier with $\tilde{O}$ hyperedges [168].

Problem 3. *Does every hypergraph have a cut/spectral sparsifier with $O(n)$ hyperedges?*

8.3. Query Complexity of Submodular Function Minimization

In Chapter 7, we proved that if the algorithm runs in a small number of rounds, we need exponential queries to solve submodular function minimization and matroid intersection. However, we can design an easy $\tilde{O}(N)$ query algorithm for the instance if we allow an unlimited round of queries. There are still huge gaps between the upper bound and lower bound for these two problems in the fully adaptive regime. For submodular function minimization, the current best-known algorithms are $O(N^3)$ queries in polynomial time and $\tilde{O}(N^2)$ Queries in exponential time [162]. For matroid intersection, the current best algorithm in the independence query (query if a set is independent) model uses $\tilde{O}(N^{9/5})$ independence queries [59] and the best algorithm in the rank query model (query the rank of a set) uses $\tilde{O}(N^{1.5})$ rank queries [79]. However, the best lower bounds for both problems are only $2N$ [136, 149].

Problem 4. *Can we solve submodular function minimization/matroid intersection in $\tilde{O}(N)$ queries? Or can we prove that any algorithm that solves submodular function minimization/matroid intersection requires $\Omega(N^{1+\varepsilon})$ queries?*

BIBLIOGRAPHY

[1] A. Abboud, K. Censor-Hillel, S. Khoury, and A. Paz. Smaller cuts, higher lower bounds. *CoRR*, abs/1901.01630, 2019. 11

[2] F. M. Ablayev. Lower bounds for one-way probabilistic communication complexity. In *Automata, Languages and Programming, 20nd International Colloquium, ICALP93, Lund, Sweden, July 5-9, 1993, Proceedings*, pages 241–252, 1993. 10

[3] A. Adamaszek, M. Mnich, and K. Paluch. New approximation algorithms for (1, 2)-tsp. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 98

[4] K. J. Ahn and S. Guha. Graph sparsification in the semi-streaming model. In *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, pages 328–338, 2009. 9

[5] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 459–467. SIAM, 2012. 28, 30

[6] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14, 2012. 9

[7] K. J. Ahn, S. Guha, and A. McGregor. Spectral sparsification in dynamic graph streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, pages 1–10, 2013. 9

[8] N. Alon and S. Assadi. Palette sparsification beyond (Δ+1) vertex coloring. In J. Byrka and R. Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 6:1–6:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 34

[9] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986. 28, 33, 191

[10] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29. ACM, 1996. 10, 11

[11] H. An, R. D. Kleinberg, and D. B. Shmoys. Improving christofides' algorithm for the s-t path TSP. *J. ACM*, 62(5):34:1–34:28, 2015. 8

[12] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 574–583, 2014. 4, 89

[13] S. Assadi. Simple round compression for parallel vertex cover. *CoRR*, abs/1709.04599, 2017. 30

[14] S. Assadi. Tight space-approximation tradeoff for the multi-pass streaming set cover problem. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 321–335, 2017. 11

[15] S. Assadi, M. Bateni, A. Bernstein, V. S. Mirrokni, and C. Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1616–1635. SIAM, 2019. 4, 30

[16] S. Assadi, A. Chen, and G. Sun. Deterministic graph coloring in the streaming model. *CoRR*, abs/2109.14891, 2021. 34

[17] S. Assadi, Y. Chen, and S. Khanna. Polynomial pass lower bounds for graph streaming algorithms. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 265–276, 2019. 12

[18] S. Assadi, Y. Chen, and S. Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786, 2019. 6, 8, 120, 121, 134, 191

[19] S. Assadi, S. Khanna, and Y. Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 698–711, 2016. 6

[20] S. Assadi, S. Khanna, and Y. Li. On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1723–1742. SIAM, 2017. 100

[21] S. Assadi, S. Khanna, Y. Li, and G. Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364, 2016. 28

[22] S. Assadi and S. Solomon. When algorithms for maximal independent set and maximal matching run in sublinear time. In *46th International Colloquium on Automata, Languages and Programming, ICALP 2019, Patras, Greece*, 2019. 34

[23] B. Axelrod, Y. P. Liu, and A. Sidford. Near-optimal approximate discrete and continuous submodular function minimization. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 837–853, 2020. 12

[24] L. Babai, P. Frankl, and J. Simon. Complexity classes in communication complexity theory (preliminary version). In *27th Annual Symposium on Foundations of Computer Science, 27-29 October 1986*, pages 337–347, 1986. 11

[25] E. Balkanski, A. Rubinstein, and Y. Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 283–302, 2019. 3, 5, 244, 247

[26] E. Balkanski and Y. Singer. Minimizing a submodular function from samples. In *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, pages 814–822, 2017. 245

[27] E. Balkanski and Y. Singer. The adaptive complexity of maximizing a submodular function. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 1138–1151, 2018. 3, 5, 244, 247

[28] E. Balkanski and Y. Singer. A lower bound for parallel submodular minimization. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 130–139, 2020. 13, 244, 245, 246

[29] N. Bansal, O. Svensson, and L. Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 910–928, 2019. 9, 143, 148

[30] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Proceedings*, pages 209–218, 2002. 11, 22, 25, 188, 191, 192, 206

[31] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 623–632, 2002. 11

[32] B. Barak, M. Braverman, X. Chen, and A. Rao. How to compress interactive communication. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, 5-8 June 2010*, pages 67–76, 2010. 21, 22, 191, 194

[33] L. Barba, J. Cardinal, M. Korman, S. Langerman, A. van Renssen, M. Roeloffzen, and S. Verdonschot. Dynamic graph coloring. *Algorithmica*, 81(4):1319–1341, 2019. 33

[34] L. Barenboim. Deterministic (Δ + 1)-coloring in sublinear (in Δ) time in static, dynamic and faulty networks. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 345–354, 2015. 33

[35] L. Barenboim and M. Elkin. Distributed deterministic edge coloring using bounded neighborhood independence. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 129–138, 2011. 30

[36] L. Barenboim and M. Elkin. **Distributed Graph Coloring: Fundamentals and Recent Developments**. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013. 33

[37] L. Barenboim, M. Elkin, and U. Goldenberg. Locally-iterative distributed ($\Delta + 1$)-coloring below szegedy-vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 437–446, 2018. 33

[38] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The locality of distributed symmetry breaking. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 321–330, 2012. 33

[39] L. Barenboim and T. Maimon. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. In *International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland*, pages 89–98, 2017. 33

[40] H. Bast, K. Mehlhorn, G. Schäfer, and H. Tamaki. Matching algorithms are fast in sparse random graphs. In *STACS 2004, 21st Annual Symposium on Theoretical Aspects of Computer Science, Montpellier, France, March 25-27, 2004, Proceedings*, pages 81–92, 2004. 75, 76

[41] J. D. Batson, D. A. Spielman, and N. Srivastava. Twice-ramanujan sparsifiers. *SIAM J. Comput.*, 41(6):1704–1721, 2012. 9, 277

[42] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 273–284, 2013. 4, 89

[43] R. Becker, A. Karrenbauer, S. Krinninger, and C. Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 7:1–7:16, 2017. 3, 6

[44] S. Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 873–884. IEEE, 2021. 4, 98, 99, 111

[45] S. Behnezhad, S. Brandt, M. Derakhshan, M. Fischer, M. Hajiaghayi, R. M. Karp, and J. Uitto. Massively parallel computation of matching and MIS in sparse graphs. In P. Robinson and F. Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 481–490. ACM, 2019. 4, 5

[46] S. Behnezhad, M. Derakhshan, and M. Hajiaghayi. Brief announcement: Semi-mapreduce meets congested clique. *CoRR*, abs/1802.10297, 2018. 30

[47] S. Behnezhad, L. Dhulipala, H. Esfandiari, J. Lacki, and V. S. Mirrokni. Near-optimal massively parallel graph connectivity. In D. Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1615–1636. IEEE Computer Society, 2019. 4

[48] S. Behnezhad, M. Hajiaghayi, and D. G. Harris. Exponentially faster massively parallel maximal matching. In D. Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1637–1649. IEEE Computer Society, 2019. 4, 30

[49] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3446^n)$: A No-MIS algorithm. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 444–452, 1995. 33

[50] A. A. Benczúr and D. R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 47–55, 1996. 9, 141, 142, 143, 144, 145, 162, 165

[51] A. A. Benczúr and D. R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. 145

[52] S. K. Bera and A. Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 11:1–11:14, 2017. 6, 11

[53] S. K. Bera, A. Chakrabarti, and P. Ghosh. Graph coloring via degeneracy in streaming

and other space-conscious models. *Manuscript*, 2019. 34

[54] S. K. Bera and P. Ghosh. Coloring in graph streams. *CoRR*, abs/1807.07640, 2018. 33, 34

[55] B. Berger and J. Rompel. A better performance guarantee for approximate graph coloring. *Algorithmica*, 5(3):459–466, 1990. 33

[56] P. Berman and M. Karpinski. 8/7-approximation algorithm for (1, 2)-tsp. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 641–648. Society for Industrial and Applied Mathematics, 2006. 98

[57] S. Bhattacharya, D. Chakrabarty, M. Henzinger, and D. Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20, 2018. 33

[58] G. E. Blelloch, J. T. Fineman, and J. Shun. Greedy sequential maximal independent set and matching are parallel on average. In *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 308–317, 2012. 191

[59] J. Blikstad, J. van den Brand, S. Mukhopadhyay, and D. Nanongkai. Breaking the quadratic barrier for matroid intersection. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 421–432, 2021. 3, 278

[60] A. Bogdanov, K. Obata, and L. Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 93–102. IEEE, 2002. 99, 114, 117, 118

[61] B. Bollobás. **Random Graphs**. Cambridge university press, 2001. 40, 61

[62] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(9):1124 – 1137, 2004. 12

[63] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 23(11):1222 – 1239, 2001. 12

[64] M. Braverman, F. Ellen, R. Oshman, T. Pitassi, and V. Vaikuntanathan. A tight bound for set disjointness in the message-passing model. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 668–677, 2013. 22, 191

[65] M. Braverman, A. Garg, D. Pankratov, and O. Weinstein. From information to exact communication. In *Symposium on Theory of Computing Conference, STOC'13, June 1-4, 2013*, pages 151–160, 2013. 188, 192, 206

[66] M. Braverman and A. Moitra. An information complexity approach to extended formulations. In *Symposium on Theory of Computing Conference, STOC'13, June 1-4, 2013*, pages 161–170, 2013. 188, 192, 206

[67] M. Braverman and A. Rao. Information equals amortized communication. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, October 22-25, 2011*, pages 748–757, 2011. 22, 24, 191, 194

[68] J. Brody, A. Chakrabarti, R. Kondapally, D. P. Woodruff, and G. Yaroslavtsev. Beyond set disjointness: the communication complexity of finding the intersection. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 106–113, 2014. 188, 191

[69] R. L. Brooks. On colouring the nodes of a network. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 37, pages 194–197. Cambridge University Press, 1941. 33

[70] S. Bubeck, Q. Jiang, Y.-T. Lee, Y. Li, and A. Sidford. Complexity of highly parallel non-smooth convex optimization. In *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, pages 13900–13909, 2019. 5, 248

[71] N. Buchbinder, M. Feldman, J. S. Naor, and R. Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing (SICOMP)*, 44(5):1384–1402, 2015. 5, 247

[72] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10(7):673–693, 1999. 9

[73] Ü. V. Çatalyürek, E. G. Boman, K. D. Devine, D. Bozdag, R. T. Heaphy, and L. A. Riesen. A repartitioning hypergraph model for dynamic load balancing. *J. Parallel Distributed Comput.*, 69(8):711–724, 2009. 9

[74] A. Chakrabarti, G. Cormode, and A. McGregor. Robust lower bounds for communication and stream computation. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, May 17-20, 2008*, pages 641–650, 2008. 10

[75] A. Chakrabarti, Y. Shi, A. Wirth, and A. C. Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001*, pages 270–278, 2001. 22

[76] D. Chakrabarty, P. Jain, and P. Kothari. Provable submodular minimization using Wolfe's algorithm. In *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, pages 802–809, 2014. 12

[77] D. Chakrabarty, Y. T. Lee, A. Sidford, S. Singla, and S. C. Wong. Faster matroid intersection. In *Proc., IEEE Symposium on Foundations of Computer Science (FOCS)*, 2019. To appear. 3

[78] D. Chakrabarty, Y. T. Lee, A. Sidford, and S. C. Wong. Subquadratic submodular function minimization. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 1220–1231, 2017. 12

[79] D. Chakrabarty, Y. T. Lee, A. Sidford, and S. C. Wong. Subquadratic submodular function minimization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1220–1231, 2017. 278

[80] Y. Chang, W. Li, and S. Pettie. An optimal distributed (Δ+1)-coloring algorithm? In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 445–456, 2018. 32, 33, 35, 39, 43

[81] Y.-J. Chang, M. Fischer, M. Ghaffari, J. Uitto, and Y. Zheng. The complexity of (Δ+1) coloring in congested clique, massively parallel computation, and centralized local computation. *CoRR*, abs/1808.08419, 2018. 34

[82] A. Chattopadhyay and S. Mukhopadhyay. Tribes is hard in the message passing model. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 224–237, 2015. 192

[83] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005. 4, 97

[84] C. Chekuri and K. Quanrud. Approximating the held-karp bound for metric TSP in nearly-linear time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 789–800, 2017. 101

[85] C. Chekuri and K. Quanrud. Fast approximations for metric-TSP via linear programming. *CoRR*, abs/1802.01242, 2018. 101

[86] C. Chekuri and K. Quanrud. Parallelizing greedy for submodular set function maximization in matroids and beyond. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 78–89, 2019. 5, 244, 247

[87] C. Chekuri and K. Quanrud. Submodular function maximization in parallel via the multilinear relaxation. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 303–322, 2019. 5, 244, 247

[88] C. Chekuri and C. Xu. Minimum cuts and sparsification in hypergraphs. *SIAM J. Comput.*, 47(6):2118–2156, 2018. 9, 138, 168

[89] L. Chen, M. Feldman, and A. Karbasi. Unconstrained submodular maximization with constant adaptive complexity. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 102–113, 2019. 5, 247

[90] Y. Chen, S. Kannan, and S. Khanna. Sublinear algorithms and lower bounds for metric TSP cost estimation. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 30:1–30:19, 2020. 9

[91] Y. Chen, S. Khanna, and A. Nagda. Near-linear size hypergraph cut sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 61–72, 2020. 10

[92] Y. Chen, S. Khanna, and A. Nagda. Sublinear time hypergraph sparsification via cut and edge sampling queries. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, pages 53:1–53:21, 2021. 10

[93] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–21, 1985. 191

[94] G. Cormode, J. Dark, and C. Konrad. Approximating the caro-wei bound for independent sets in graph streams. In *Combinatorial Optimization - 5th International Symposium, ISCO 2018, Marrakesh, Morocco, April 11-13, 2018, Revised Selected Papers*, pages 101–114, 2018. 191

[95] G. Cormode, J. Dark, and C. Konrad. Independent sets in vertex-arrival streams. *CoRR*, abs/1807.08331, 2018. 191

[96] G. Cormode and H. Jowhari. A second look at counting triangles in graph streams (corrected). *Theor. Comput. Sci.*, 683:22–30, 2017. 11

[97] T. M. Cover and J. A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006. 18

[98] W. H. Cunningham. On submodular function minimization. *Combinatorica*, 5:185 – 192, 1985. 12

[99] A. Czumaj, J. Lacki, A. Madry, S. Mitrovic, K. Onak, and P. Sankowski. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, June 25-29, 2018*, pages 471–484, 2018. 4, 30

[100] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009. 4, 8, 97

[101] A. Czumaj and C. Sohler. Sublinear time approximation of the cost of a metric k-nearest neighbor graph. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2973–2992. SIAM, 2020. 4

[102] D. Dadush, L. A. Végh, and G. Zambelli. Geometric rescaling algorithms for submodular function minimization. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 832–848, 2018. 12

[103] G. A. Dirac. Some theorems on abstract graphs. *Proceedings of the London Mathematical Society*, 3(1):69–81, 1952. 132

[104] D. P. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *Random Struct. Algorithms*, 13(2):99–124, 1998. 259

[105] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2):674–701, 2012. 5, 248

[106] P. Duris, Z. Galil, and G. Schnitger. Lower bounds on communication complexity. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 81–91, 1984. 10

[107] J. Edmonds. Submodular Functions, Matroids, and Certain Polyhedra. In R. Guy, H. Hanam, and J. Schonheim, editors, *Combinatorial structures and their applications*, pages 69–85, New York, 1970. Gordon and Breach. 271

[108] S. Eggert, L. Kliemann, and A. Srivastav. Bipartite graph matchings in the semi-streaming model. In *Algorithms - ESA 2009, 17th Annual European Symposium, September 7-9, 2009. Proceedings*, pages 492–503, 2009. 3

[109] M. Elkin, S. Pettie, and H. Su. (2Δ - l)-edge-coloring is much easier than maximal matching in the distributed setting. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 355–370, 2015. 32, 39, 43

[110] T. Emden-Weinert, S. Hougardy, and B. Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Combinatorics, Probability & Computing*,

7(4):375–386, 1998. 33

[111] A. Ene and H. L. Nguyen. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 274–282, 2019. 5, 244, 247

[112] A. Ene, H. L. Nguyen, and A. Vladu. Submodular maximization with matroid and packing constraints in parallel. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 90–101, 2019. 5, 244, 247

[113] D. Eppstein. Small maximal independent sets and faster exact graph coloring. *J. Graph Algorithms Appl.*, 7(2):131–140, 2003. 33

[114] M. Fahrbach, V. Mirrokni, and M. Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 255–273, 2019. 5, 247

[115] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci.*, 57(2):187–199, 1998. 33

[116] U. Feige, V. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. *SIAM Journal on Computing (SICOMP)*, 40(4):1133 – 1153, 2011. 247

[117] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. 3

[118] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 2008. 6, 10, 11

[119] T. Fife and J. Oxley. Laminar matroids. *European Journal of Combinatorics*, 62:206–216, 2017. 265

[120] M. Fischer, M. Ghaffari, and F. Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 180–191, 2017. 30

[121] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18(01n02):3–28, 2008. 85

[122] P. Fraigniaud, M. Heinrich, and A. Kosowski. Local conflict coloring. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 625–634, 2016. 33

[123] W. S. Fung, R. Hariharan, N. J. A. Harvey, and D. Panigrahi. A general framework for graph sparsification. *SIAM J. Comput.*, 48(4):1196–1223, 2019. 16

[124] Z. Gao. On the metric s-t path traveling salesman problem. *SIAM Review*, 60(2):409–426, 2018. 8

[125] D. Gavinsky, J. Kempe, I. Kerenidis, R. Raz, and R. de Wolf. Exponential separations for one-way quantum communication complexity, with applications to cryptography. *STOC*, pages 516–525, 2007. 10

[126] M. Ghaffari, T. Gouleakis, C. Konrad, S. Mitrovic, 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, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 129–138, 2018. 3, 5, 6, 30, 191

[127] M. Ghaffari, D. R. Karger, and D. Panigrahi. Random contractions and sampling for hypergraph and hedge connectivity. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1101–1114, 2017. 142

[128] M. Ghaffari and J. Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1636–1653. SIAM, 2019. 4

[129] S. Ghosh, R. Gurjar, and R. Raj. A deterministic parallel reduction from weighted matroid intersection search to decision. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page to appear, 2022. 246

[130] A. Goel, M. Kapralov, and I. Post. Single pass sparsification in the streaming model with edge deletions. *CoRR*, abs/1203.4900, 2012. 9

[131] O. Goldreich. **Introduction to Property Testing**. Cambridge University Press, 2017. 29, 87

[132] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. 4

[133] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. 4

[134] V. Goncharov. Some facts from combinatorics. *Izvestia Akad. Nauk. SSSR, Ser. Mat*, 8:3–48, 1944. 133

[135] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, pages 374–383, 2011. 4, 89

[136] A. Graur, T. Pollner, V. Ramaswamy, and S. M. Weinberg. New query lower bounds for submodular function minimization. In *Proc., Innovations in Theoretical Computer Science (ITCS)*, pages 64:1–64:16, 2020. 278

[137] M. Grötschel, László Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169 – 197, 1981. 3, 12

[138] S. Guha and A. McGregor. Tight lower bounds for multi-pass stream computation via pass elimination. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 760–772, 2008. 10

[139] S. Guha, A. McGregor, and D. Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 241–247, 2015. 139

[140] R. Gurjar and R. Rathi. Linearly representable submodular functions: An algebraic algorithm for minimization. In *Proc., International Colloquium on Automata, Languages and Programming (ICALP)*, pages 61:1–61:15, 2020. 245

[141] V. Guruswami and K. Onak. Superlinear lower bounds for multipass graph processing. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 287–298, 2013. 10, 11, 190

[142] B. V. Halldórsson, M. M. Halldórsson, E. Losievskaja, and M. Szegedy. Streaming algorithms for independent sets. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, pages 641–652, 2010. 191

[143] B. V. Halldórsson, M. M. Halldórsson, E. Losievskaja, and M. Szegedy. Streaming algorithms for independent sets in sparse hypergraphs. *Algorithmica*, 76(2):490–501, 2016. 191

[144] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Inf. Process. Lett.*, 45(1):19–23, 1993. 33

[145] M. M. Halldórsson, X. Sun, M. Szegedy, and C. Wang. Streaming and communication complexity of clique approximation. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings,*

Part I, pages 449–460, 2012. 11, 191

[146] S. Har-Peled, P. Indyk, S. Mahabadi, and A. Vakilian. Towards tight bounds for the streaming set cover problem. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 371–383, 2016. 3, 6

[147] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities (Cambridge Mathematical Library)*. Cambridge University Press, 1988. 15

[148] D. G. Harris, J. Schneider, and H.-H. Su. Distributed $(\Delta+1)$-coloring in sublogarithmic rounds. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 465–478. ACM, 2016. 32, 33, 35, 39, 43

[149] N. J. A. Harvey. Matroid intersection, pointer chasing, and young's seminormal representation of S_n. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 542–549, 2008. 278

[150] N. J. A. 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, SPAA 2018, July 16-18, 2018*, pages 43–52, 2018. 30

[151] J. Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001. 121

[152] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. 75, 77

[153] Y. Huang, Q. Liu, and D. N. Metaxas. Video object segmentation by hypergraph cut. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1738–1745, 2009. 9

[154] R. Impagliazzo and V. Kabanets. Constructive proofs of concentration bounds. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, September 1-3, 2010. Proceedings*, pages 617–631, 2010. 259

[155] P. Indyk. Sublinear time algorithms for metric space problems. In J. S. Vitter, L. L. Larmore, and F. T. Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 428–434. ACM, 1999. 4

[156] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, 48(4):761–777, 2001. 12

[157] S. Iwata and J. B. Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1230–1237, 2009. 12

[158] R. Iyer and J. A. Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, 2013. 12

[159] R. Iyer, S. Jegelka, and J. A. Bilmes. Fast semidifferential-based submodular function optimization. In *Proc., International Conference on Machine Learning (ICML)*, pages 855–863, 2013. 12

[160] R. Jain, J. Radhakrishnan, and P. Sen. A direct sum theorem in communication complexity via message compression. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, June 30 - July 4, 2003. Proceedings*, pages 300–315, 2003. 11

[161] T. S. Jayram, R. Kumar, and D. Sivakumar. Two applications of information complexity. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 673–682, 2003. 192, 206

[162] H. Jiang. Minimizing convex functions with integral minimizers. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 976–985, 2021. 3, 12, 278

[163] D. Johnson. Worst case behavior of graph coloring algorithms. In *Proceedings of the 5th Southeast Conference on Combinatorics, Graph Theory, and Computing, 1974*, pages 513–527, 1974. 33

[164] H. Jowhari and M. Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, pages 710–716, 2005. 11

[165] H. Jowhari, M. Sağlam, and G. Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 49–58. ACM, 2011. 85

[166] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992. 11, 188

[167] M. Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In D. Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1874–1893. SIAM, 2021. 6

[168] M. Kapralov, R. Krauthgamer, J. Tardos, and Y. Yoshida. Spectral hypergraph spar-

sifiers of nearly linear size. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1159–1170. IEEE, 2021. 277

[169] M. Kapralov, R. Krauthgamer, J. Tardos, and Y. Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 598–611, 2021. 10, 139

[170] M. Kapralov, Y. T. Lee, C. Musco, C. Musco, and A. Sidford. Single pass spectral sparsification in dynamic streams. *SIAM J. Comput.*, 46(1):456–477, 2017. 9

[171] M. Kapralov, S. Mitrović, A. Norouzi-Fard, and J. Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1753–1772. SIAM, 2020. 4, 98

[172] M. Kapralov, A. Mousavifar, C. Musco, C. Musco, N. Nouri, A. Sidford, and J. Tardos. Fast and space efficient spectral sparsification in dynamic streams. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1814–1833. SIAM, 2020. 9

[173] M. Kapralov, J. Nelson, J. Pachocki, Z. Wang, D. P. Woodruff, and M. Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 475–486, 2017. 85

[174] D. R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, SODA*, pages 21–30, 1993. 9, 142, 145

[175] D. R. Karger. Random sampling in cut, flow, and network design problems. *Math. Oper. Res.*, 24(2):383–413, 1999. 9

[176] D. R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. 147

[177] D. R. Karger and R. Motwani. An NC algorithm for minimum cuts. *SIAM J. Comput.*, 26(1):255–272, 1997. 245

[178] A. R. Karlin, N. Klein, and S. O. Gharan. A (slightly) improved approximation algorithm for metric TSP. In S. Khuller and V. V. Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 32–45. ACM, 2021. 8

[179] H. J. 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, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948, 2010. 4, 89

[180] R. M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972. 33

[181] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986. 245

[182] R. M. Karp, E. Upfal, and A. Wigderson. The complexity of parallel search. *J. Comput. System Sci.*, 36(2):225–253, 1988. 246

[183] M. Karpinski, M. Lampis, and R. Schmied. New inapproximability bounds for TSP. *J. Comput. Syst. Sci.*, 81(8):1665–1677, 2015. 8

[184] R. Kiveris, S. Lattanzi, V. S. Mirrokni, V. Rastogi, and S. Vassilvitskii. Connected components in mapreduce and beyond. In *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, November 03 - 05, 2014*, pages 18:1–18:13, 2014. 4

[185] D. Kogan and R. Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS*, pages 367–376, 2015. 9, 138, 139, 142, 146, 148, 168, 169

[186] P. Kohli, M. P. Kumar, and P. H. S. Torr. P3 and beyond: Move making algorithms for solving higher order functions. *IEEE Trans. Pattern Anal. and Machine Learning*, 31:1–8, 2008. 12

[187] C. Konrad. MIS in the congested clique model in o(log log Δ) rounds. *CoRR*, abs/1802.07647, 2018. 5, 30

[188] C. Konrad, F. Magniez, and C. Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 231–242, 2012. 3

[189] I. Kremer, N. Nisan, and D. Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999. 10, 135

[190] K. Kutzkov and R. Pagh. Triangle counting in dynamic graph streams. In *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen,*

Denmark, July 2-4, 2014. Proceedings, pages 306–318, 2014. 6

[191] S. Lacoste-Julien and M. Jaggi. On the global linear convergence of Frank-Wolfe optimization variants. In *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, 2015. 12

[192] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 85–94, 2011. 28, 30

[193] E. L. Lawler. A note on the complexity of the chromatic number problem. *Inf. Process. Lett.*, 5(3):66–67, 1976. 33

[194] Y. T. Lee, A. Sidford, and S. C. Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1049–1065, 2015. 12, 17

[195] Y. T. Lee and H. Sun. An sdp-based algorithm for linear-sized spectral sparsification. In H. Hatami, P. McKenzie, and V. King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 678–687. ACM, 2017. 9, 277

[196] W. Li, P. Liu, and J. Vondrák. A polynomial lower bound on adaptive complexity of submodular maximization. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 140–152, 2020. 3, 5, 244, 247

[197] H. Lin. Reducing directed max flow to undirected max flow. *Unpublished manuscript*, 2009. 240

[198] J. Lin. Divergence measures based on the shannon entropy. *IEEE Trans. Information Theory*, 37(1):145–151, 1991. 21

[199] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975. 33

[200] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. 28, 33, 191

[201] A. McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, pages 170–181, 2005. 3, 6

[202] A. McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014. 5, 139, 190

[203] A. McGregor, D. Tench, S. Vorotnikova, and H. T. Vu. Densest subgraph in dynamic graph streams. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 472–482, 2015. 85

[204] M. Mnich and T. Mömke. Improved integrality gap upper bounds for traveling salesperson problems with distances one and two. *European Journal of Operational Research*, 266(2):436–457, 2018. 8

[205] M. Molloy and B. Reed. **Graph Colouring and the Probabilistic Method**, volume 23. Springer Science & Business Media, 2013. 16, 33, 45

[206] M. Molloy and B. A. Reed. A bound on the total chromatic number. *Combinatorica*, 18(2):241–280, 1998. 32, 35, 39, 43

[207] M. Molloy and B. A. Reed. Asymptotically optimal frugal colouring. *J. Comb. Theory, Ser. B*, 100(2):226–246, 2010. 32, 35

[208] M. Molloy and B. A. Reed. Colouring graphs when the number of colours is almost the maximum degree. *J. Comb. Theory, Ser. B*, 109:134–195, 2014. 32, 33, 35

[209] T. Mömke and O. Svensson. Approximating graphic TSP by matchings. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 560–569, 2011. 103

[210] T. Mömke and O. Svensson. Removing and adding edges for the traveling salesman problem. *Journal of the ACM (JACM)*, 63(1):2, 2016. 8, 97, 104

[211] R. Motwani. Expanding graphs and the average-case analysis of algorithms for matchings and related problems. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 550–561, 1989. 75, 76

[212] M. Mucha. $\frac{13}{9}$-approximation for graphic tsp. *Theory of computing systems*, 55(4):640–657, 2014. 97

[213] M. Naor and L. J. Stockmeyer. What can be computed locally? In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 184–193, 1993. 26

[214] H. Narayanan, H. Saran, and V. V. Vazirani. Randomized parallel algorithms for matroid union and intersection, with applications to arborescences and edge-disjoint

spanning trees. *SIAM J. Comput.*, 23(2):387–397, 1994. 245, 246

[215] G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978. 247

[216] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Math. Programming*, 14(1):265–294, 1978. 247

[217] A. Nemirovski. On parallel complexity of nonsmooth convex optimization. *Journal of Complexity*, 10(4):451–463, 1994. 5, 247

[218] H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 327–336, 2008. 4, 98

[219] N. Nisan and A. Wigderson. Rounds in communication complexity revisited. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 419–429, 1991. 10, 191

[220] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1123–1131. Society for Industrial and Applied Mathematics, 2012. 4, 98

[221] J. B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Math. Programming*, 118(2):237–251, 2009. 12

[222] A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001. 30

[223] C. H. Papadimitriou and M. Sipser. Communication complexity. *J. Comput. Syst. Sci.*, 28(2):260–269, 1984. 10

[224] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993. 98, 114

[225] M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007. 33, 98

[226] M. Parter. $(\Delta+1)$ coloring in the congested clique model. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 160:1–160:14, 2018. 29, 30, 32, 33, 35

[227] M. Parter and H. Su. Randomized (Δ+1)-coloring in $O(\log^* \Delta)$ congested clique rounds. In *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, pages 39:1–39:18, 2018. 33

[228] S. Ponzio, J. Radhakrishnan, and S. Venkatesh. The communication complexity of pointer chasing: Applications of entropy and sampling. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 602–611, 1999. 10

[229] V. Rastogi, A. Machanavajjhala, L. Chitnis, and A. D. Sarma. Finding connected components in map-reduce in logarithmic rounds. In *29th IEEE International Conference on Data Engineering, ICDE 2013, April 8-12, 2013*, pages 50–61, 2013. 4

[230] A. A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992. 11, 188

[231] B. Reed. ω, Δ, and χ. *Journal of Graph Theory*, 27(4):177–212, 1998. 32, 35

[232] B. A. Reed. A strengthening of brooks' theorem. *J. Comb. Theory, Ser. B*, 76(2):136–149, 1999. 32, 35

[233] V. Rödl. On a packing and covering problem. *Eur. J. Comb.*, 6(1):69–78, 1985. 32

[234] A. Rubinstein, T. Schramm, and S. M. Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 39:1–39:16, 2018. 3, 5, 6, 190

[235] J. Schneider and R. Wattenhofer. A new technique for distributed symmetry breaking. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 257–266, 2010. 32, 33

[236] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Combin. Theory Ser. B*, 80(2):346–355, 2000. 12

[237] A. Schrijver. *Combinatorial Optimization*. Springer, New York, 2003. 267

[238] A. Sebö and A. van Zuylen. The salesman's improved paths: A 3/2+1/34 approximation. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 118–127, 2016. 8

[239] A. Sebö and J. Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combina-*

torica, 34(5):597–629, 2014. 8, 97, 104

[240] A. Sidford and K. Tian. Coordinate methods for accelerating ℓ_∞ regression and faster approximate maximum flow. *CoRR*, abs/1808.01278, 2018. 240

[241] D. A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In L. Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC*, pages 81–90. ACM, 2004. 9

[242] V. Traub and J. Vygen. Beating the integrality ratio for s-t-tours in graphs. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 766–777, 2018. 8

[243] V. Traub and J. Vygen. Approaching 3/2 for the s-t-path TSP. *J. ACM*, 66(2):14:1–14:17, 2019. 8

[244] E. Verbin and W. Yu. The streaming complexity of cycle counting, sorting by reversals, and other problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, January 23-25, 2011*, pages 11–25, 2011. 10

[245] J. Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM J. Comput.*, 42(1):265–304, 2013. 247

[246] J. Vygen. New approximation algorithms for the tsp. 2012. 8

[247] O. Weinstein and D. P. Woodruff. The simultaneous communication of disjointness with applications to data streams. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, July 6-10, 2015, Proceedings, Part I*, pages 1082–1093, 2015. 192, 206

[248] D. B. West. *Introduction to graph theory*. Prentice-Hall Inc., 1996. 104

[249] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *J. ACM*, 30(4):729–735, 1983. 33

[250] Y. Yamaguchi, A. Ogawa, A. Takeda, and S. Iwata. Cyber security analysis of power networks by hypergraph cut algorithms. *IEEE Trans. Smart Grid*, 6(5):2189–2199, 2015. 9

[251] A. C. Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213, 1979. 21

[252] A. C. Yao. Lower bounds by probabilistic arguments (extended abstract). In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 420–428, 1983. 192, 222, 224

[253] A. C. Yao. Lower bounds to randomized algorithms for graph properties (extended abstract). In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 393–400, 1987. 92, 128

[254] A. Yehudayoff. Pointer chasing via triangular discrimination. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:151, 2016. 10, 191

[255] Y. Yoshida, M. Yamamoto, and H. Ito. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM Journal on Computing*, 41(4):1074–1093, 2012. 4, 98

[256] M. Zelke. Intractability of min- and max-cut in streaming graphs. *Inf. Process. Lett.*, 111(3):145–150, 2011. 5, 6

[257] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 681–690, 2006. 33