When Stuck, Flip a Coin: New Algorithms for Large-Scale Tasks

THÈSE N^{O} 8580 (2018)

PRÉSENTÉE LE 15 JUIN 2018

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE THÉORIE DES COMMUNICATIONS

PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Slobodan MITROVIC

acceptée sur proposition du jury:

Prof. E. Telatar, président du jury
Prof. R. Urbanke, Prof. A. Madry, directeurs de thèse
Dr S. Lattanzi, rapporteur
Prof. S. Khanna, rapporteur
Prof. M. Ghaffari, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2018

If you find a path with no obstacles,
it probably doesn't lead anywhere.
— Frank A. Clark

To my parents and my fiance . . .

Acknowledgements

I was fortunate to have three amazing advisors: Aleksander Mądry, Bernard Moret and Rüdiger Urbanke. I would like to thank Aleksander for his guidance and for introducing me to many of my collaborators. I am also grateful for his help in arranging my visits to other universities. Aleksander taught me that being independent is crucial in becoming a researcher. When Aleksander moved to MIT, Bernard and Rüdiger became my advisors at EPFL. I am immensely grateful for their support and their availability whenever it was needed. In particular, they gave me complete freedom in choosing my collaborators and topics of research.

I also want to thank the jury members of my thesis defense, Emre Telatar, Silvio Lattanzi, Sanjeev Khanna and Mohsen Ghaffari, for providing valuable feedback on my thesis and for all the inspiring discussions that we had during and after the defense. I am grateful to Olivier Leveque for proofreading the French version of my abstract.[1]

I want to thank all the EPFL members who helped me deal with bureaucracy and conference trips. Muriel Bardet was impressive in this regard and set the bar really high.

I am grateful to Piotr Sankowski, Monika Henzinger and Ronitt Rubinfeld for hosting me during my studies. Collaborating with them was an amazing experience that greatly influenced my work. Ronitt easily qualifies as one of the most positive people that I have ever met.

I want to thank my collaborators for, above all, being great teachers. I am very grateful to Krzysztof Onak for fitting many of our online discussions into his busy schedule.

In the absence of Aleksander, Ola Svensson, Michael Kapralov, Nisheeth Vishnoi and their groups made me feel like a part of the family. I had a pleasure to collaborate with Ashkan and Jakub for a significant part of my studies. Also many thanks to Abbas, Aida, Amir, Buddhima, Christos, Damian, Farah, Navid and Sagar for their great company and collaboration.

I want to express my gratefulness to Aca, Ana, Cristina, Danica, Dragan, Dragiša, Goran, Nebojša, Nikola, Radivoje, Sara, Siniša and Viktor for all the friendly words they shared with me. Moreover, I was lucky to know Mihailo who acted like a "big brother" to me. With Peđa I got to share many stories over coffee and contemplate about after-PhD life.

I spent nearly one year at MIT as part of the theory group and met so many great people there. Special thanks to G (aka Gautam) for acting as a social bond between me and the group. Their vibrant atmosphere provided me a unique research experience. I also had many great discussions with Ivan and an occasional recollection of our master's studies at EPFL.

I am grateful to my parents, brother and sister for their continuous understanding. I am still impressed by their willingness, during my visits home, to adjust their daily schedules to mine and, by doing so, make me feel extremely supported in what I do.

My fiance filled this journey with laughter and caring love and provided selfless support. She was my most persistent collaborator.

[1] Special thanks to Google for assisting my initial translation of the abstract to French.

Abstract

Many modern services need to routinely perform tasks on a large scale. This prompts us to consider the following question:

How can we design efficient algorithms
for large-scale computation?

In this thesis, we focus on devising a general strategy to address the above question. Our approaches use tools from graph theory and convex optimization, and prove to be very effective on a number of problems that exhibit locality. A recurring theme in our work is to use randomization to obtain simple and practical algorithms.

The techniques we developed enabled us to make progress on the following questions:

- **Parallel Computation of Approximately Maximum Matchings.** We put forth a new approach to computing $O(1)$-approximate maximum matchings in the Massively Parallel Computation (MPC) model. In the regime in which the memory per machine is $\Theta(n)$, i.e., linear in the size of the vertex-set, our algorithm requires only $O((\log\log n)^2)$ rounds of computations. This is an almost exponential improvement over the barrier of $\Omega(\log n)$ rounds that all the previous results required in this regime.

- **Parallel Computation of Maximal Independent Sets.** We propose a simple randomized algorithm that constructs maximal independent sets in the MPC model. If the memory per machine is $\Theta(n)$ our algorithm runs in $O(\log\log n)$ MPC-rounds. In the same regime, all the previously known algorithms required $O(\log n)$ rounds of computation.

- **Network Routing under Link Failures.** We design a new protocol for stateless message-routing in k-connected graphs. Our routing scheme has two important features: (1) each router performs the routing decisions based only on the local information available to it; and, (2) a message is delivered successfully even if arbitrary $k-1$ links have failed. This significantly improves upon the previous work of which the routing schemes tolerate only up to $k/2-1$ failed links in k-connected graphs.

- **Streaming Submodular Maximization under Element Removals.** We study the problem of maximizing submodular functions subject to cardinality constraint k, in the context of streaming algorithms. In a regime in which up to m elements can be removed from the stream, we design an algorithm that provides a constant-factor approximation for this problem. At the same time, the algorithm stores only $O(k\log^2 k + m\log^3 k)$ elements. Our algorithm improves quadratically upon the prior work, that requires storing $O(k \cdot m)$ many elements to solve the same problem.

- **Fast Recovery for the Separated Sparsity Model.** In the context of compressed sensing, we put forth two recovery algorithms of nearly-linear time for the separated sparsity signals (that naturally model neural spikes). This improves upon the previous algorithm that had a quadratic running time. We also derive a refined version of the natural dynamic programming (DP) approach to the recovery of the separated sparsity signals. This DP approach leads to a recovery algorithm that runs in linear time for an important class of separated sparsity signals. Finally, we consider a generalization of these signals into two dimensions, and we show that computing an exact projection for the two-dimensional model is NP-hard.

Key words: Large-Scale Tasks, Maximum Matching, Maximal Independent Set, Parallel Computation, Network Routing, Submodular Maximization, Compressed Sensing, Separated Sparsity

Résumé

De nombreux services modernes doivent effectuer régulièrement des tâches à grande échelle. Cela nous amène à considérer la question suivante :

Comment pouvons-nous concevoir des algorithmes efficaces pour le calcul à grande échelle ?

Dans cette thèse, nous nous concentrons sur la conception une stratégie générale pour aborder la question ci-dessus. Notre approche utilise des outils issus de la théorie des graphes et de l'optimisation convexe, et se révèle très efficace sur un certain nombre des problèmes qui exposent de localité. Un thème récurrent dans nos résultats consiste à tirer parti de la randomisation pour obtenir des algorithmes simples et pratiques.

Les techniques développées nous ont permis de progresser sur les questions suivantes :

- **Calcul parallèle de couplages appromixativement maximaux.** Nous proposons une nouvelle approche pour calculer des couplages $O(1)$-approximatifs maximaux dans le modèle du Calcul Massivement Parallèle (MPC). Dans le régime où la mémoire disponible par machine est $\Theta(n)$, notre algorithme ne nécessite que $O((\log\log n)^2)$ cycles de calculs. C'est une amélioration presque exponentielle par rapport à la barrière de $\Omega(\log n)$ que tous les résultats précédents requièrent dans ce régime.

- **Calcul parallèle d'ensembles indépendants maximaux.** Nous proposons un algorithme aléatoire simple qui construit des ensembles indépendants maximaux dans le modèle MPC. Si la mémoire disponible par machine est $\Theta(n)$, notre algorithme s'exécute en $O(\log\log n)$ cycles de calculs. Dans le même régime, tous les algorithmes précédemment connus nécessitaient $O(\log n)$ cycles de calculs.

- **Routage réseau avec échecs de connexions.** Nous proposons un nouveau protocole pour le routage des messages dans les graphes k-connectés. Notre schéma de routage a deux caractéristiques importantes : (1) chaque routeur travaille avec pour seule base les informations disponibles localement et (2) un message est transmis avec succès même si $k-1$ connexions arbitraires ont échoué. Ceci améliore considérablement le résultat précédent dont le schéma de routage ne tolérait que jusqu'à $k/2-1$ connexions défaillantes dans un graphe k-connecté.

- **Algorithmes de streaming pour maximisation sous-modulaire avec suppression d'éléments.** Nous étudions le problème de la maximisation des fonctions sous-modulaires sujettes à contrainte de cardinalité k, dans le contexte des algorithmes de streaming. Dans un régime dans lequel jusqu'à m éléments peuvent être retirés du stream, nous

concevons un algorithme qui fournit un facteur constant d'approximation pour ce problème. En même temps, l'algorithme stocke seulement $O(k\log^2 k + m\log^3 k)$ éléments. Notre algorithme représente une nette amélioration par rapport aux résultats précédents, qui nécessitaient de stocker $O(km)$ éléments pour résoudre le même problème.

- **Récupération rapide pour le modèle épars séparé.** Dans le contexte de l'acquisition comprimée, nous proposons deux algorithmes de récupération presque linéaires pour des signaux épars séparés (qui modélisent de manière naturelle les pics neuronaux). Ceci améliore l'algorithme précédemment connu dont le temps de récupération était quadratique. Nous proposons également une version améliorée de notre algorithme basée sur la programmation dynamique (DP) pour la récupération de signaux épars séparés. Cette approche DP conduit à un algorithme de récupération qui s'exécute en temps linéaire pour une classe importante de ces signaux. Finalement, nous considérons une généralisation de ces signaux à deux dimensions, et montrons que calculer une projection exacte pour le modèle bidimensionnel est NP-difficile.

Mots clefs : tâches à grande échelle, couplages maximaux, ensembles indépendants maximaux, calcul parallèle, routage réseau, maximisation sous-modulaire, acquisition comprimée, modèle épars séparé

Contents

List of Figures

List of Tables

1 Introduction

The development of the Internet, combined with the availability of inexpensive storage, has led to the accumulation of an immense amount of data. For example, in 2016, Google Knowledge Graph encompassed 70 billion facts [Goo16]. Similarly, Amazon offers more than half a billion products [Scr18], with an objective of recommending a personalized list of items to each of more than 300 million active users [Sta16]. These two examples, along with many other modern data-processing tasks, are performed on *a large scale*, and all have one property in common – the data of interest *does not fit on one machine*. This gave rise to various models of computation that naturally capture settings in which these tasks are performed. The design of these models is affected by the way data is accessed, by the amount of memory available at each computational unit, whether the result of computation is used only temporarily or stored permanently in the memory, and by many other properties. To exploit their power, we need to design efficient algorithms for these models. In this context, we explore four topics: streaming algorithms, parallel computation, network routing, and compressed sensing.

Low-memory algorithms – the streaming setting

Streaming algorithms attempt to aggregate statistics of data by accessing each data item only once. These algorithms were studied already in the 1980s [MP80, FM85], and they were popularized by the work of Alon, Matias and Szegedy [AMS99]. Streaming algorithms are designed to operate with *a small memory* and in scenarios in which the data is produced so rapidly that it cannot even be stored. As they need only a small memory, these algorithms can also be used in tasks for simultaneously aggregating many statistics (e.g., for active users) of a collection of items (e.g., available products).

Low round complexity in parallel computation

In many large-scale scenarios, it is difficult (or even impossible) to construct the desired data structure (e.g., the Knowledge Graph) by accessing the data only once and using a small memory. One way to approach these types of tasks is to gradually build the corresponding structure. This is done in multiple steps (also called *rounds*), where each round consists of processing — *in parallel* — the data across a number of machines. Over the last two decades, a number of frameworks have been developed for the purpose of large-scale parallel computation; examples include MapReduce [DG04], Hadoop [Whi12], Dryad [IBY+07], or Spark [ZCF+10]. Due to their natural approach to processing massive data, these frameworks have gained

great popularity and found a variety of applications. They are some of the key components in computational operations carried out in the tech industry, big-data processing, machine learning, clinical-data analysis, bioinformatics, weather forecasting, etc. As communication is the bottleneck in parallel computation, it is desirable that algorithms designed for this setting perform a few rounds.

Network routing with local decisions and under link failures

A proper choice of a model of computation (e.g., streaming, parallel) and the design of an adequate algorithm are some of the defining features of efficient methods for solving large-scale tasks. Nevertheless, an integral part of many large-scale computation systems is *network routing* [MR17, Kur05]. For instance, parallel computation is carried by a number of machines that communicate during this process, and the task of network routing is to ensure the delivery of messages sent between machines. We also need the routing decisions made at every routing device to be based only on *the local information* available to it. This type of protocol leads to relatively small routing tables, that are usually necessary for obtaining fast routing, which leads to overall efficient computation. Furthermore, communication links are prone to failures, which has to be addressed. This setting brings additional challenges in the design of routing protocols as, in addition to having access only to the local state of the network, they have to tolerate link failures.

Recovery algorithms for structured compressed sensing

In the aforementioned examples, as a consequence of their limited memory, the computational devices have constrained access to data. Nevertheless, there are scenarios in which the data can be accessed (or *measured*) only in a very specific way, regardless of the available memory. These scenarios gave rise to the area of *compressed sensing*. The purpose of compressed sensing is to perform a small number of linear measurements of data (usually a sparse signal), then to recover the data from the acquired measurements. Seminal results [Don06, CRT06, FR13] show that if the measurements satisfy certain regularity conditions, such as the restricted isometry property (RIP), then from only $O(k \log n/k)$ linear observations it is possible to recover a k-sparse n-dimensional vector in polynomial time. This work found applications for medical imaging [LDP07], data-stream algorithms [GI10], and sparse linear regression [HTW15].

A natural question is how to extend the notion of sparsity in order to model more complex structures present in real-world data. For instance, wavelet coefficients can naturally be arranged as a tree and, for many classes of images, the large wavelet coefficients form a subtree of the wavelet coefficient tree. Ideally, utilizing such structure beyond "'standard"' sparsity improves the statistical efficiency of a task of interest, e.g., by leading to $o(k \log n/k)$ number of linear observations needed for sparse recovery. Indeed, there is now a large body of work on structured sparsity, and there are several sparsity models that yield a provably better sample complexity, both in theory and in practice (among others, [YL06, EM09, BCDH10, MJOB11, RRN12, NRWY12, BBC14, HIS15a]). However, these improvements in sample complexity require the recovery algorithms to optimize over a set with more complex structure. This typically leads to recovery algorithms that have a worse running time than their "standard sparsity" counterparts.

1.1 Thesis Motivation

The aforementioned scenarios prompt us to consider the following question:

How can we efficiently solve large-scale problems?

In this thesis, we mainly focus on developing new combinatorial tools and techniques that can be applied in the context of memory-constrained computation. In light of this objective, we address the following fundamental problems:

- Constructing approximate maximum matchings and maximal independent sets in models of parallel computation.
- Designing stateless routing protocols that efficiently deliver messages, even when many links in the network have failed.
- Maximizing submodular functions over large datasets when the elements are presented in the streaming fashion and, in addition, elements can be retracted from the stream once they have been seen.
- In the context of compressed sensing, designing fast recovery procedures for *separated sparsity* signals (a classic model that naturally corresponds to neural spikes).

Most of the algorithms we design are randomized and stem from simple deterministic approaches. Each of these deterministic algorithms either does not always solve the underlying task or, if it does solve the task, it is not efficient. Our methods remedy this situation by making randomized decisions at the steps where, informally speaking, their deterministic counterparts get stuck. We elaborate on this strategy in Section 1.3.

1.2 Overview of Our Contributions

In this section we give an overview of our results. In Section 1.3 we present the common theme of our approaches and later, in separate chapters, explain each of the approaches in detail.

1.2.1 Parallel Computation of Approximately Maximum Matchings

Maximum matchings have been the cornerstone of algorithmic research since the 1950s and their study inspired many important ideas, including the complexity class P [Edm65]. We study this problem in the context of the Massively Parallel Computation (MPC) model. In the MPC model, we have m machines at our disposal and each of them has S words of space. Initially, each machine receives its share of the input. In our case, the input is a collection of edges. The computation proceeds in synchronous *rounds*. In each round, the data is split across the machines and processed locally. At the end, all the machines output their results, and this output is used as the input in the next round. As communication is the bottleneck in parallel computation, for given S, the main measure of complexity in this setting is *the number of rounds* the computation requires.

Even though the maximum matching problem has been studied in models of parallel computation for more than 30 years [Lub86, IS86, II86, LMSV11, AG15, AK17], it seems that all

known techniques fail to improve upon the $\Omega(\log n)$ MPC-round complexity when the memory per machine is $S = \Theta(n)$ (on graphs of vertex-size n). We broke this barrier by designing an algorithm that constructs a $(1+\varepsilon)$-approximate maximum matching in $O\left(\left(\log\log n\right)^2\right)$ rounds, for any constant $\varepsilon > 0$.

1.2.2 Parallel Computation of Maximal Independent Sets

Maximal independent set (MIS) is one of the most fundamental problems in algorithmic graph theory. In the context of parallel computation, this problem has been studied since the 1980s [Lub86, ABI86, II86, IS86]. MIS was also studied in models of distributed computation, such as LOCAL and CONGESTED-CLIQUE (e.g., [BEPS12, Gha17, CHPS17, BFS12, FN18]).

In the context of the MPC computation when the memory per machine is $\Theta(n)$ (on graphs of vertex-size n), the prior work implies $O(\log n)$ round complexity for constructing MIS. We design an algorithm that constructs MIS in $O(\log\log n)$ MPC-rounds of computation. If the maximum degree of the input graph is bounded by Δ, then our algorithm constructs an MIS in $O(\log\log\Delta)$ rounds. We show that the same algorithm can be executed in $O(\log\log\Delta)$ rounds of CONGESTED-CLIQUE.

1.2.3 Network Routing under Link Failures

In the context of network routing, we study the task of computing routing tables in the setting where

- Links used to pass messages through the network can fail. Each router detects which of the links incident to it have failed.
- Each router performs stateless routing based only on the local information available to it. That is, if a router receives a message through link e, then it forwards the message through a link that is chosen as a function of the message destination, the link e, and the set of failed links incident to the router.

Prior work shows how to design this type of routing tables that in k-connected networks guarantee successful delivery if up to $k/2-1$ links have failed [EGR14]. We improve upon this result and design a randomized routing protocol that in k-connected networks delivers messages, even if arbitrary $k-1$ links have failed. This result is optimal in the parameter k, because a failure of k links in a k-connected network could entirely disconnect a message from its destination. Furthermore, our protocol performs randomized decisions only in case the routing encounters a failed link, otherwise it is deterministic.

1.2.4 Streaming Submodular Maximization under Element Removals

For completeness, we first give a definition of submodularity.

Definition 1.1 (Submodular functions)**.** *Let V be a potentially large universe of elements. A set function $f : 2^V \to \mathbb{R}_{\geq 0}$ is* monotone *if for any two sets $X \subseteq Y \subseteq V$ we have $f(X) \leq f(Y)$. The function f is said to be* submodular *if for any two sets $X \subseteq Y \subseteq V$ and any element $e \in V \setminus Y$ it holds that*

$$f(X \cup \{e\}) - f(X) \geq f(Y \cup \{e\}) - f(Y).$$

The task of *data summarization* is to select a small representative subset (e.g., to recommend a personalized list of items) out of a large dataset. This task, along with many other real-world problems, can be viewed as *monotone submodular function maximization*.

Often, in these applications elements can be removed, e.g., from the recommended list of items, a user removes items from the list that he has already bought. This gave rise to a task of constructing a set of elements that retains a large value even after its small subset is removed. Motivated by this scenario, we study the problem of maximizing submodular functions subject to cardinality constraint k, in the streaming setting. Furthermore, we consider the regime in which *up to* m items can be retracted from the stream. It is known there is a constant-factor approximation for this problem if the algorithm is enabled to store $O(k \cdot m)$ elements [MKK17]. We design an algorithm that quadratically improves upon this result. Specifically, we show an algorithm that solves the same problem while storing only $O(k \log^2 k + m \log^3 k)$ elements.

1.2.5 Fast Recovery for Separated Sparsity Signals

In the context of compressed sensing, we study recovery algorithms for the *separated sparsity model*. Separated sparsity has been proposed as a model for neuronal spike trains, i.e., time series data in which we record the activity of a single neuron (or a set of neurons) [HDC09, DDJB10, HB11, DSRB13, FMN15]. An important aspect of neuronal activity patterns is that neurons have a minimum *refractory period* between consecutive activity spikes. The separated sparsity model formalizes this fact by assuming that the signals are not only k-sparse but also that each non-zero entry in the signal vector has to be separated by at least a given number Δ of zero entries. Prior work establishes a tight sample-complexity bound of $O(k \log(n/k - \Delta))$ for this setting, but the best known recovery algorithm has a time complexity of $\tilde{O}(n^2)$ [FMN15] that quickly becomes impractical on large data sets.

In our work, we address this issue and provide new, faster algorithms that achieve the same improved sample complexity while running in *nearly-linear* time. Our new algorithms essentially match the time complexity of standard sparse recovery and show that we can utilize separated sparsity without a significant increase in time complexity.

An outline of our results in context of the separated sparsity model is given below:

- We design both a randomized and a deterministic nearly-linear time recovery algorithm for the separated sparsity model.

- We derive a refined version of the natural dynamic programming (DP) approach to the recovery of the separated sparsity model. The time complexity of the resulting DP-based algorithm improves from $O(n^2)$ to $O(k(n-(k-1)\Delta))$. Consequently, in the regime of $n/k - \Delta = O(1)$ of the separated sparsity model that requires $O(k)$ measurements, our algorithm has a time complexity of $O(n + k^2)$.

- We analyze the time complexity of the separated sparsity model in *two dimensions*; it is a model in which the non-zeros in a matrix are separated by a certain minimum amount. We show that, in contrast to the one-dimensional case, computing an exact projection for the two-dimensional model is NP-hard.

1.3 A General Strategy

At the core of this thesis is the following question:

How can we efficiently solve large-scale problems?

Many of the problems studied in the context of large-scale computation already have efficient *centralized* algorithms, i.e., algorithms that are efficient if the entire computation is performed on a single machine that has no memory restriction. However, large-scale computations are necessarily performed across many machines and/or with memory restriction. As a result, translating efficient centralized algorithms to this setting is challenging.

In this thesis, we formulate a general strategy for performing such translation and then demonstrate its effectiveness on a number of problems.

In a broad sense, our strategy focuses on translating centralized algorithms that perform *local* computation into efficient memory-constrained algorithms. We show that in some cases it is possible to simulate these local computations by using "small memory". For instance, consider a centralized graph algorithm that processes each vertex by considering only its neighborhood, i.e., for each vertex it has to know only local information. Then, instead of considering the whole neighborhood of a vertex, it could be possible to select a small but representative vertex-neighborhood by a proper sampling. This would allow the simulated algorithm to perform computation by using much smaller memory than the total input size. However, due to the sampling, the resulting simulated algorithm behaves as the input instance was perturbed. If the centralized algorithm is not "stable enough", this simulation on a perturbed input can output a solution which is far from the one corresponding to the actual input. For instance, the centralized algorithm could make significantly different decisions depending on whether the degree of a vertex is d or $d-1$. To circumvent this, we demonstrate on a number of problems how to regularize constraints leading to such erratic behavior. The regularization we apply depends on the problem we consider.

In the subsequent sections, we illustrate that this strategy improves state of the art for the following problems:

- **Maximal matching**: Our starting point is a centralized algorithm that iteratively matches vertices whose degree is above certain *sharp threshold*. To efficiently simulate this algorithm in parallel setting, we apply our strategy as follows. We first show how to sample neighborhoods of vertices so that the sample is small but also carries sufficient information about the degrees of vertices. Then, instead of applying a sharp degree-threshold to decide which vertices to match, we use a carefully chosen smooth one which affects the final solution only by a little.

- **Maximal independent set**: We start from an algorithm that iteratively removes a maximum-degree vertex and its neighborhood. Detecting such vertices requires too much communication in a parallel setting. Instead of requiring to remove *only* maximum-degree vertices, we allow removal of a relatively small number of other vertices as well. This regularized constraint enables us to design a significantly more efficient algorithm for parallel computation than directly simulating the starting one.

- **Network routing**: Some routing schemes address link failures by computing new routing paths that are failure-free. In large networks, computing such paths usually leads to

significant overhead in communication. We obtain an efficient routing scheme by applying our strategy as follows. At a high level, instead of requiring that new routing paths are *always* failure-free we require that they are *often* failure-free. Namely, some of the new routing paths might still contain failed links, but we show that it happens rarely.

- **Compressed sensing**: We demonstrate that our strategy is also applicable in designing fast algorithms. Namely, we consider a Lagrangian relaxation of the separated sparsity projection problem. However, in some cases this relaxation *does not* provide the intended solution. Intuitively, this happens as a certain structure of the problem is very "unstable" under small perturbations of the corresponding Lagrangian parameter. We apply our strategy by regularizing this structure. This, in turn, stabilizes the instance and enables us to apply Lagrangian relaxation leading to a simple and efficient algorithm.

1.4 Thesis Outline

We present our results in separate chapters. Each of the chapters is self-contained, and the reader can use the following condensed content-list to locate topics of interest:
In **Chapter 2**, we describe our result on constructing approximate maximum matchings
In **Chapter 3**, we describe our result on constructing maximal independent sets in parallel computation
In **Chapter 4**, we present our work on network routing
In **Chapter 5**, we are devoted to describing our approach to streaming submodular maximization
In **Chapter 6**, we present our results on the separated sparsity model

Each of the chapters begins with an introduction and a brief history about the underlying problem. This content is followed by the definition of the model we consider or by the problem setup. Next, we state our contributions and provide a section with related work. In the remaining content, we first overview our approach and techniques. Then, we describe our ideas in detail and present our proofs.

In **Chapter 7**, we provide some directions for future work that we find interesting.

2 Approximate Maximum Matchings in Parallel Computation

This chapter is based on a joint work with Artur Czumaj, Jakub Łącki, Aleksander Mądry, Krzysztof Onak, and Piotr Sankowski. It has been accepted to the 50th ACM Symposium on Theory of Computing (**STOC**) 2018 [CŁM+18] under the title

Round Compression for Parallel Matching Algorithms.

This paper has also been **invited to the special issue** of SIAM Journal on Computing.

2.1 Introduction

Over the last decade, massive parallelism became a major paradigm in computing, and we have witnessed the deployment of a number of very successful massively parallel computation frameworks, such as MapReduce [DG04, DG08], Hadoop [Whi12], Dryad [IBY+07], or Spark [ZCF+10]. This paradigm and the corresponding models of computation are rather different from classical parallel algorithms models considered widely in literature, such as the PRAM model. In particular, in this chapter, we study the *Massive Parallel Computation (MPC)* model (also known as Massively Parallel Communication model) that was abstracted out of capabilities of existing systems, starting with the work of Karloff, Suri, and Vassilvitskii [KSV10, GSZ11, BKS13, ANOY14, BKS14]. The main difference between this model and the PRAM model is that the MPC model allows for much more (in principle, unbounded) local computation. This enables it to capture a more "coarse–grained," and thus, potentially, more meaningful aspect of parallelism. It is often possible to simulate one clock step of PRAM in a constant number of rounds on MPC [KSV10, GSZ11]. This implies that algorithms for the PRAM model usually give rise to MPC algorithms without incurring any asymptotic blow up in the number of parallel rounds. As a result, a vast body of work on PRAM algorithms naturally translates to the new model.

It is thus natural to wonder: Are the MPC parallel round bounds "inherited" from the PRAM model tight? In particular, which problems can be solved in significantly *smaller* number of MPC rounds than what the lower bounds established for the PRAM model suggest?

In this chapter, we focus on one such problem, which is also one of the most central graph problems both in sequential and parallel computations: maximum matching. In the PRAM model we can compute $(1+\epsilon)$-approximate matching in $O(\log n)$ rounds [LPP15] using randomization. Deterministically, a $(2+\epsilon)$-approximation can be computed in $O\left(\log^2 n\right)$

rounds [FG17]. We note that these results hold in a distributed message passing setting, where processors are located at graph nodes and can communicate only with neighbors. In such a distributed setting, $\Omega\left(\sqrt{\log n/\log\log n}\right)$ time lower bound is known for computing any constant approximation to maximum matching [KMW06].

So far, in the MPC setting, the prior results are due to Lattanzi, Moseley, Suri, and Vassilvitskii [LMSV11], Ahn and Guha [AG15] and Assadi and Khanna [AK17]. Lattanzi et al. [LMSV11] put forth algorithms for several graph problems, such as connected components, minimum spanning tree, and maximum matching problem, that were based on a so-called *filtering technique*. In particular, using this technique, they have obtained an algorithm that can compute a 2-approximation to maximum matching in $O(1/\delta)$ MPC rounds, provided that the space per machine is $S = \Omega\left(n^{1+\delta}\right)$, for any constant $\delta \in (0,1)$. Later on, Ahn and Guha [AG15] provided an improved algorithm that computes a $(1+\epsilon)$-approximation in $O(1/(\delta\epsilon))$ rounds, provided $S = \Omega\left(n^{1+\delta}\right)$, for any constant $\delta > 0$. Both these results, however, crucially require that the space per machine is significantly superlinear in n. In fact, if the space S is linear in n, which is a very natural setting for massively parallel graph algorithms, the performance of both these algorithms degrades to $O(\log n)$ parallel rounds, which matches what was known for the PRAM model. Recently, Assadi and Khanna [AK17] showed how to construct randomized composable coresets of size $\tilde{O}(n)$ that give an $O(1)$-approximation for maximum matching. Their techniques apply to the MPC model only if the space per machine is $\tilde{O}(n\sqrt{n})$.

We also note that the known PRAM maximal independent set and maximal matching algorithms [Lub86, ABI86, II86] can be used to find a maximal matching (i.e., 2-approximation to maximum matching) in $O(\log n)$ MPC rounds as long as space per machine is at least $n^{\Omega(1)}$ (i.e., $S \geq n^c$ for some constant $c > 0$). We omit further details here, except mentioning that a more or less direct simulation of those algorithms is possible via an $O(1)$-round sorting subroutine [GSZ11].

The above results give rise to the following fundamental question: Can the maximum matching be (approximately) solved in $o(\log n)$ parallel rounds in $O(n)$ space per machine? The main result of this chapter is an affirmative answer to that question. We show that, for any $S = \Omega(n)$, one can obtain an $O(1)$-approximation to maximum matching using $O\left((\log\log n)^2\right)$ parallel MPC rounds. So, not only do we break the existing $\Omega(\log n)$ barrier, but also provide an exponential improvement over the previous work. Our algorithm can also provide a $(2+\epsilon)$, instead of $O(1)$-approximation, at the expense of the number of parallel rounds increasing by a factor of $O(\log(1/\epsilon))$. Finally, our approach can also provide algorithms that have $o(\log n)$ parallel round complexity also in the regime of S being (mildly) sublinear. For instance, we obtain $O\left((\log\log n)^2\right)$ MPC rounds even if space per machine is $S = n/(\log n)^{O(\log\log n)}$. The exact comparison of our bounds with previous results is given in Table 2.1.

2.1.1 Model

In this work, we adopt a version of the model introduced by Karloff, Suri, and Vassilvitskii [KSV10] and refined in later works [GSZ11, BKS13, ANOY14]. We call it *massive parallel computation* (MPC), which is a mutation of the name proposed by Beame et al. [BKS13].

In the MPC model, we have m machines at our disposal and each of them has S words of space. Initially, each machine receives its share of the input. In our case, the input is a collection E of edges and each machine receives approximately $|E|/m$ of them.

The computation proceeds in *rounds*. During the round, each of the machines processes

its local data without communicating with other machines. At the end of each round, machines exchange messages. Each message is sent only to a single machine specified by the machine that is sending the message. All messages sent and received by each machine in each round have to fit into the machine's local memory. Hence, their total length is bounded by S.[1] This in particular implies that the total communication of the MPC model is bounded by $m \cdot S$ in each round. The messages are processed by recipients in the next round.

At the end of the computation, machines collectively output the solution. The data output by each machine has to fit in its local memory. Hence again, each machine can output at most S words.

The range of values for S and m. If the input is of size N, one usually wants S sublinear in the N, and the total space across all the machines to be at least N—so the input fits onto the machines—and ideally not much larger. Formally, one usually considers $S \in \Theta\left(N^{1-\epsilon}\right)$, for some $\epsilon > 0$.

In this chapter, the focus is on graph algorithms. If n is the number of vertices in the graph, the input size can be as large as $\Theta\left(n^2\right)$. Our parallel algorithm requires $\Theta(n)$ space per machine (or even slightly less), which is polynomially less than the size of the input for dense graphs.

Communication vs. computation complexity. The main focus of this work is the number of (communication) rounds required to finish computation. Also, even though we do not make an effort to explicitly bound it, it is apparent from the design of our algorithms that every machine performs $O(S \operatorname{polylog} S)$ computation steps locally. This in particular implies that the overall work across all the machines is $O(r N \operatorname{polylog} S)$, where r is the number of rounds and N is the input size (i.e., the number of edges).

2.1.2 Our Results

In our work, we focus on computing an $O(1)$-approximate maximum matching in the MPC model. We collect our results and compare to the previous work in Table 2.1. The table presents two interesting regimes for our algorithms. On the one hand, when the space per machine is $S = O(n)$, we obtain an algorithm that requires $O((\log\log n)^2)$ rounds. This is the first known algorithm that, with linear space per machine, breaks the $O(\log n)$ round barrier. On the other hand, in the mildly sublinear regime of space per machine, i.e., when $S = O(n/f(n))$, for some function $f(n)$ that is $n^{o(1)}$, we obtain an algorithm that still requires $o(\log n)$ rounds. This, again is the first such result in this regime. In particular, we prove the following result.

Theorem 2.1

There exists an MPC algorithm that constructs an $O(1)$-approximation to maximum matching with constant probability in $O\left((\log\log n)^2 + \max\left(\log \frac{n}{S}, 0\right)\right)$ rounds, where $S = n^{\Omega(1)}$ is the amount of space on each machine.

[1] This for instance allows a machine to send a single word to $S/100$ machines or $S/100$ words to one machine, but not $S/100$ words to $S/100$ machines if $S = \omega(1)$, even if the messages are identical.

Source	Approx.	Space	Rounds	Remarks
[LMSV11]	2	$n^{1+\Omega(1)}$	$O(1)$	Maximal matching
		$O(n)$	$O(\log n)$	
[AG15]	$1+\epsilon$	$O\left(n^{1+1/p}\right)$	$O(p/\epsilon)$	$p>1$
	2	$n^{\Omega(1)}$	$O(\log n)$	Maximal matching Simulate [Lub86, ABI86, II86]
here	$O(1)$	$O(n)$	$O\left((\log\log n)^2\right)$	$\epsilon \in (0,1/2)$ $2 \le f(n) = O\left(n^{1/2}\right)$
	$2+\epsilon$		$O\left((\log\log n)^2 \cdot \log(1/\epsilon)\right)$	
	$O(1)$	$O(n)/f(n)$	$O\left((\log\log n)^2 + \log f(n)\right)$	
	$2+\epsilon$		$O\left((\log\log n)^2 + \log f(n)\right) \cdot \log(1/\epsilon)$	

Table 2.1 – Comparison of our results for computing approximate maximum size matchings to the previous results for the MPC model.

As a corollary, we obtain the following result that provides nearly 2-approximate maximum matching.

Corollary 2.2. *For any $\epsilon \in (0, \frac{1}{2})$, there exists an MPC algorithm that constructs a $(2+\epsilon)$-approximation to maximum matching with 99/100 probability in $O\left((\log\log n)^2 + \max\left(\log \frac{n}{S}, 0\right)\right) \cdot \log(1/\epsilon)$ rounds, where $S = n^{\Omega(1)}$ is the amount of space on each machine.*

Assadi et al. [ABB+17] observe that one can use a technique of McGregor [McG05] to extend the algorithm to compute a $(1+\epsilon)$-approximation in $O((\log\log n)^2) \cdot (1/\epsilon)^{O(1/\epsilon)}$ rounds.

It should also be noted that (as pointed out to us by Seth Pettie) any $O(1)$-approximation algorithm for unweighted matchings can be used to obtain a $(2+\epsilon)$-approximation algorithm for weighted matchings (see Section 4 of his paper with Lotker and Patt-Shamir [LPP15] for details). In our setting this implies that Theorem 2.1 yields an algorithm that computes a $(2+\epsilon)$-*approximation* to maximum weight matching in $O((\log\log n)^2 \cdot (1/\epsilon))$ rounds and $O(n \log n)$ space per machine.

2.1.3 Related Work

We note that there were efforts at modeling MapReduce computation [FMS+10] before the work of Karloff et al. Also a recent work [RVW16] investigates the complexity of the MPC model.

In the *filtering* technique, introduced by Lattanzi et al. [LMSV11], the input graph is iteratively sparsified until it can be stored on a single machine. For the matching problem, the sparsification is achieved by first obtaining a small sample of edges, then finding a maximal matching in the sample, and finally removing all the matched vertices. Once a sufficiently small graph is obtained, a maximal matching is computed on a single machine. In the $S = \Theta(n)$ regime, the authors show that their approach reduces the number of edges by a constant factor in each iteration. Despite this guarantee, until the very last step, each iteration may make little progress towards obtaining even an approximate maximal matching, resulting in a $O(\log n)$ round complexity of the algorithm. Similarly, the results of Ahn and Guha [AG15] require $n^{1+\Omega(1)}$ space per machine to compute a $O(1)$-approximate maximum weight matching in a constant number of rounds and do not imply a similar bound for the case of linear space.

We note that the algorithm of Lattanzi et al. [LMSV11] cannot be turned easily into a fast approximation algorithm when space per machine is sublinear. Even with $\Theta(n)$ space, their method is able to remove only a constant fraction of edges from the graph in each iteration, so $\Omega(\log n)$ rounds are needed until only a matching is left. When $S = \Theta(n)$, their algorithm works as follows: sample uniformly at random $\Theta(n)$ edges of the graph, find maximal matching on the sampled set, remove the matched vertices, and repeat. We do not provide a formal proof here, but on the following graph this algorithm requires $\tilde{\Omega}(\log n)$ rounds, even to discover a constant factor approximation. Consider a graph consisting of t separate regular graphs of degree 2^i, for $0 \leq i \leq t-1$, each on 2^t vertices. This graph has $t2^t$ nodes and the algorithm requires $\tilde{\Omega}(t)$ rounds even to find a constant approximate matching. The algorithm chooses edges uniformly at random, and few edges are selected each round from all but the densest remaining subgraphs. Thus, it takes multiple rounds until a matching of significant size is constructed for sparser subgraphs. This example emphasizes the weakness of direct edge sampling and motivates our vertex sampling scheme that we introduce in our work.

Similarly, Ahn and Gupta [AG15] build on the filtering approach of Lattanzi et al. and design a primal-dual method for computing a $(1+\epsilon)$-approximate weighted maximum matching. They show that each iteration of their distributed algorithm either makes large progress in the dual, or they can construct a large approximate matching. Regardless of their new insights, their approach is inherently edge-sampling based and does not break the $O(\log n)$ round complexity barrier when $S = O(n)$.

Despite the fact that MPC model is rather new, computing matching is an important problem in this model, as the above mentioned two papers demonstrate. This is further witnessed by the fact that the distributed and parallel complexity of maximal matching has been studied for many years already. The best deterministic PRAM maximal matching algorithm, due to Israeli and Shiloach [IS86], runs in $O\left(\log^3 n\right)$ rounds. Israeli and Itai [II86] gave a randomized algorithm for this problem that runs in $O(\log n)$ rounds. Their algorithm works as well in CONGEST, a distributed message-passing model with a processor assigned to each vertex and a limit on the amount of information sent along each edge per round. A more recent paper by Lotker, Patt-Shamir, and Pettie [LPP15] gives a $(1+\epsilon)$-approximation to maximum matching in $O(\log n)$ rounds also in the CONGEST model, for any constant $\epsilon > 0$. On the deterministic front, in the LOCAL model, which is a relaxation of CONGEST that allows for an arbitrary amount of data sent along each edge, a line of research initiated by Hańćkowiak, Karoński, and Panconesi [HKP01, HKP99] led to an $O\left(\log^3 n\right)$-round algorithm by Fischer and Ghaffari [FG17].

On the negative side, Kuhn, Moscibroda, and Wattenhofer [KMW06] showed that any distributed algorithm, randomized or deterministic, when communication is only between neighbors requires $\Omega\left(\sqrt{\log n / \log\log n}\right)$ rounds to compute a constant approximation to maximum matching. This lower bound applies to all distributed algorithms that have been mentioned above. Our algorithm circumvents this lower bound by loosening the only possible assumption there is to be loosened: single-hop communication. In a sense, we assign subgraphs to multiple machines and allow multi-hop communication between nodes in each subgraph.

Finally, the ideas behind the peeling algorithm that is a starting point for this work can be traced back to the papers of Israeli, Itai, and Shiloach [II86, IS86], that can be interpreted as matching high-degree vertices first in order to reduce the maximum degree. A sample distributed algorithm given in a work of Parnas and Ron [PR07] uses this idea to compute

an $O(\log n)$ approximation for vertex cover. Their algorithm was extended by Onak and Rubinfeld [OR10] in order to provide an $O(1)$-approximation for vertex cover and maximum matching in a dynamic version of the problems. This was achieved by randomly matching high-degree vertices to their neighbors in consecutive phases while reducing the maximum degree in the remaining graph. This approach was further developed in the dynamic graph setting by a number of papers [BHI15, BHN16, BHN17, BCH17]. Ideas similar to those in the paper of Parnas and Ron [PR07] were also used to compute polylogarithmic approximation in the streaming model by Kapralov, Khanna, and Sudan [KKS14]. Our version of the peeling algorithm was directly inspired by the work of Onak and Rubinfeld [OR10] and features important modifications in order to make our analysis go through.

Recent developments In a recent work, Assadi [Ass17] applied the round compression idea to the distributed $O(\log n)$-approximation algorithm for vertex cover of Parnas and Ron [PR07]. Using techniques from his recent work with Khanna [AK17], he gave a simple MPC algorithm that in $O(\log\log n)$ rounds and $n/\operatorname{polylog}(n)$ space per machine computes an $O(\log n)$-approximation to minimum vertex cover.

Second, a new paper by Assadi et al. [ABB+17] provides an MPC algorithm that computes $O(1)$-approximation to both vertex cover and maximum matching in $O(\log\log n)$ rounds and $\tilde{O}(n)$ space per machine (though the space is strictly superlinear). Their result builds on techniques developed originally for dynamic matching algorithms [BS15, BS16] and composable coresets [AK17]. It is worth to note that their construction critically relies on the vertex sampling approach (i.e., random assignment of vertices to machines) introduced in our work.

2.2 Preliminaries

For a graph $G = (V, E)$ and $V' \subseteq V$, we write $G[V']$ to denote the subgraph of G induced by V'. Formally, $G[V'] \stackrel{\text{def}}{=} \big(V', E \cap (V' \times V')\big)$. We also write $N(v)$ to denote the set of neighbors of a vertex v in G.

2.3 Overview and Organization

In this section we present the main ideas and techniques behind our result. The result of this chapter contains two main technical contributions.

First, our algorithm *randomly partitions vertices* across the machines, and on each machine considers only the corresponding induced graph. We prove that it suffices to consider these induced subgraphs to obtain an approximate maximum matching. Note that this approach greatly deviates from previous works, that used edge based partitioning.

Second, we introduce a *round compression* technique. Namely, we start with an algorithm that executes $O(\log n)$ phases and can be naturally implemented in $O(\log n)$ MPC rounds and then demonstrate how to emulate this algorithm using only $o(\log n)$ MPC rounds. The underlying idea is quite simple: each machine independently runs multiple phases of the initial algorithm. This approach, however, has obvious challenges since the machines cannot communicate in a single round of the MPC algorithm. The rest of the section is devoted to describing our approach and illustrating how to overcome these challenges.

2.3.1 Vertex Based Sampling

The algorithms for computing maximal matching in PRAM and their simulations in the MPC model [Lub86, ABI86, IS86, II86] are designed to, roughly speaking, either halve the number of the edges or halve the maximum degree in each round. Therefore, in the worst case those algorithms inherently require $\Omega(\log n)$ rounds to compute a maximal matching.

On the other hand, all the algorithm for the maximal matching problem in the MPC model prior to ours ([LMSV11, AG15, AK17]) process the input graph by discarding edges, and eventually aggregate the remaining edges on a single machine to decide which of them are part of the final matching. It is not known how to design approaches similar to [LMSV11, AG15, AK17] while avoiding a step in which the maximal matching computation is performed on a single machine. This seems to be a barrier for improving upon $O(\log n)$ rounds, if the space available on each machine is $O(n)$.

The starting point of our new approach is alleviating this issue by resorting to a more careful vertex based sampling. Specifically, at each round, we randomly partition the vertex set into vertex sets $V_1, \ldots, V_m$ and consider induced graphs on those subsets independently. Such sampling scheme has the following handy property: the union of matchings obtained across the machines is still a matching. Furthermore, we show that for the appropriate setting of parameters this sampling scheme allows us to handle vertices of a wide range of degrees in a single round, unlike handling only high-degree vertices (that is, vertices with degree within a constant factor of the maximum degree) as guaranteed by [II86, IS86].

2.3.2 Global Algorithm

To design an algorithm executed on machines locally, we start from a sequential peeling algorithm `GlobalAlg` (see Algorithm 1), that is a modified version of an algorithm used by Onak and Rubinfeld [OR10]. The algorithm had to be significantly adjusted in order to make our later analysis of a parallel version possible.

The execution of `GlobalAlg` is divided into $\Theta(\log n)$ *phases*. In each phase, the algorithm first computes a set H of high-degree vertices. Then it selects a set F of vertices, that we call *friends*. Next the algorithm selects a matching $\widetilde{M}$ between H and F, using a simple randomized strategy. F is carefully constructed so that both F and $\widetilde{M}$ are likely to be of order $\Theta(|H|)$. Finally, the algorithm removes all vertices in $H \cup F$, hence reducing the maximum vertex degree in the graph by a constant factor, and proceeds to the next phase. The central property of `GlobalAlg` is that it returns an $O(1)$ approximation to maximum matching with constant probability (Corollary 2.6). A detailed discussion of `GlobalAlg` is given in Section 2.4.

2.3.3 Parallel Emulation of the Global Algorithm (Section 2.5)

The following two ways could be used to execute `GlobalAlg` in the MPC model: (1) place the whole graph on one machine, and trivially execute all the phases of `GlobalAlg` in a single round; or (2) simulate one phase of `GlobalAlg` in one MPC round while using $O(n)$ space per machine, by distributing vertices randomly onto machines (see Section 2.7.1 for details). However, each of these approaches has severe drawbacks. The first approach requires $\Theta(|E|)$ space per machine, which is likely to be prohibitive for large graphs. On the other hand, while the second approach uses $O(n)$ space, it requires $\Theta(\log n)$ rounds of MPC computation. We

Algorithm 1: `GlobalAlg`$(G, \widetilde{\Delta})$
Global matching algorithm

Input: Graph $G = (V, E)$ of maximum degree at most $\widetilde{\Delta}$
Output: A matching in G

$\Delta \leftarrow \widetilde{\Delta}$, $M \leftarrow \emptyset$, $V' \leftarrow V$
while $\Delta \geq 1$ **do**
  `/* Invariant: the maximum degree in` $G[V']$ `is at most` Δ `*/`
  Let $H \subset V'$ be a set of vertices of degree at least $\Delta/2$ in $G[V']$. We call vertices in H *heavy*.
  Create a set F of *friends* by selecting each vertex $v \in V'$ independently with probability $|N(v) \cap H|/4\Delta$.
  Compute a matching $\widetilde{M}$ in $G[H \cup F]$ using `MatchHeavy`(H, F) and add it to M.
  $V' \leftarrow V' \setminus (H \cup F)$, $\Delta \leftarrow \Delta/2$
return M

Algorithm 2: `MatchHeavy`(H, F)
Computing a matching in $G[H \cup F]$

Input: set H of heavy vertices and set F of friends
Output: a matching in $G[H \cup F]$

For every vertex $v \in F$ pick uniformly at random a heavy neighbor $v_\star$ in $N(v) \cap H$.
Independently at random color each vertex in $H \cup F$ either red or blue.
Select the following subset of edges: $E_\star \leftarrow \{(v, v_\star) : v \in F \wedge v \text{ is red} \wedge v_\star \in H \wedge v_\star \text{ is blue}\}$.
For every blue vertex w incident to an edge in $E_\star$, select one such edge and add it to $\widetilde{M}$.
return $\widetilde{M}$

achieve the best of both worlds by showing how to emulate the behavior of *multiple phases* of `GlobalAlg` in a *single MPC round* with each machine using $O(n)$ space, thus obtaining an MPC algorithm requiring $o(\log n)$ rounds. More specifically, we show that it is possible to emulate the behavior of `GlobalAlg` in $O\left((\log\log n)^2\right)$ rounds with each machine using $O(n)$ (or even only $n/(\log n)^{O(\log\log n)}$) space.

Before we provide more details about our parallel multi-phase emulation of `GlobalAlg`, let us mention the main obstacle such an emulation encounters. At the beginning of every phase, `GlobalAlg` has access to the full graph. Therefore, it can easily compute the set of heavy vertices H. On the other hand, machines in our MPC algorithm use $O(n)$ space and thus have access only to a small subgraph of the input graph (when $|E| \gg n$). In the first phase this is not a big issue, as, thanks to randomness, each machine can estimate the degrees of high-degree vertices. However, the degrees of vertices can significantly change from phase to phase. Therefore, after each phase it is not clear how to select high-degree vertices in the next phase without inspecting the entire graph again. Hence, one of the main challenges in designing a multi-phase emulation of `GlobalAlg` is to ensure that machines at the beginning of every phase can estimate *global* degrees of vertices well enough to identify the set of heavy vertices, while each machine still having access only to its local subgraph. This property is achieved using a few modifications to the algorithm.

Preserving Randomness

Our algorithm partitions the vertex set into m disjoint subsets V_i by assigning each vertex independently and uniformly at random. Then the graph induced by each subset V_i is processed on a separate machine. Each machine finds a set of heavy vertices, H_i, by estimating the global degree of each vertex of V_i. It is not hard to argue (using a standard concentration bound) that there is enough randomness in the initial partition so that local degrees in each induced subgraph roughly correspond to the global degrees. Hence, after the described partitioning, sets H and $\bigcup_{i\in[m]} H_i$ have very similar properties. This observation crucially relies on the fact that initially the vertices are distributed *independently* and *uniformly* at random.

However, if one attempts to execute the second phase of `GlobalAlg` without randomly reassigning vertices to sets after the first phase, the remaining vertices are no longer distributed independently and uniformly at random. In other words, after inspecting the neighborhood of every vertex locally and making a decision based on it, the randomness of the initial random partition may significantly decrease.

Let us now make the following thought experiment. Imagine for a moment that there is an algorithm that emulates multiples phases of `GlobalAlg` in parallel and in every phase inspects **only** the vertices that end-up being matched. Then, from the point of view of the algorithm, the vertices that are not matched so far are still distributed independently and uniformly at random across the machines. Or, saying in a different way, if randomness of some vertices is not inspected while emulating a phase, then at the beginning of the next phase those vertices still have the same distribution as in the beginning of that MPC round. But, how does an algorithm learn about vertices that should be matched by inspecting no other vertex? How does the algorithm learn even only about high-degree vertices without looking at their neighborhood?
In the sequel we show how to design an algorithm that looks only "slightly" at the vertices that do not end-up being matched. As we prove, that is sufficient to design a multi-phase emulation of `GlobalAlg`.

We now discuss in more detail how to preserve two crucial properties of our vertex assignments throughout the execution of multiple phases: independent and nearly-uniform distribution.

Independence (Lemma 2.9)

As noted above, it is not clear how to compute vertex degrees without inspecting their local neighborhood. A key, and at first sight counter-intuitive, step in our approach is to *estimate* even *local degrees* of vertices (in contrast to computing them exactly). To obtain the estimates, it suffices to examine only small neighborhoods of vertices and in turn preserve the independent distribution of the intact ones. More precisely, we sample a small set of vertices on each machine, called *reference sets*, and use the set to estimate the local degrees of all vertices assigned to this machine. Furthermore, we show that with a proper adjustments of `GlobalAlg` these estimates are sufficient for capturing high-degree vertices.

Very crucially, all the vertices that are used in computing a matching in one emulated phase (including the reference sets) are discarded at the end of the phase, even if they do not participate in the obtained matching. In this way we disregard the vertices which position is fixed and, intuitively, secure an independent distribution of the vertices across the machines

in the next phase.

We also note, without going into details, that obtaining full independence required modifying how the set of friends is selected, compared to the original approach of Onak and Rubinfeld [OR10]. In their approach, each heavy vertex selected one friend at random. However, as before, in order to select exactly one friend would require examining neighborhood of heavy vertices. This, however, introduces dependencies between vertices that have not been selected. So instead, in our `GlobalAlg`, every vertex selects itself as a friend independently and proportionally to the number of high-degree vertices (found using the reference set), which again secures an independent distribution of the remaining vertices. The final properties of the obtained sets in either approach are very similar.

Uniformity (Lemma 2.10)

A very convenient property in the task of emulating multiple phases of `GlobalAlg` is a uniform distribution of vertices across all the machines at every phase – for such a distribution, we know the expected number of neighbors of each desired type assigned to the same machine. Obtaining perfect uniformity seems difficult—if not impossible in our setting—and we therefore settle for *near* uniformity of vertex assignments. The probability of the assignment of each vertex to each machine is allowed to differ slightly from that in the uniform distribution. Initially, the distribution of each vertex is uniform and with every phase it can deviate more and more from the uniform distribution. We bound the rate of the decay with high probability and execute multiple rounds as long as the deviation from the uniform distribution is negligible. More precisely, in the execution of the entire parallel algorithm, the sufficiently uniform distribution is on average kept over $\Omega\left(\frac{\log n}{(\log\log n)^2}\right)$ phases of the emulation of `GlobalAlg`.

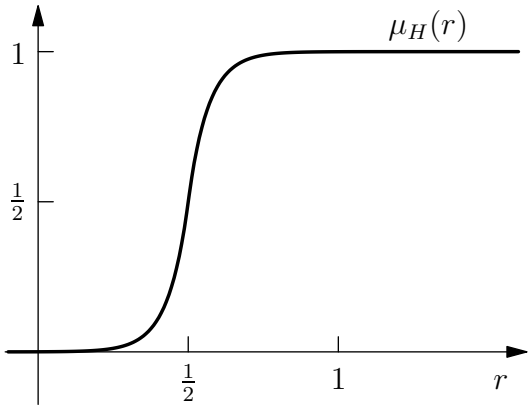

Figure 2.1 – An idealized version of $\mu_H : \mathbb{R} \to [0,1]$, in which n was fixed to a small constant and the multiplicative constant inside the exponentiation operator was lowered.

In order to achieve the near uniformity, we modify the procedure for selecting H, the set of high-degree vertices. Instead of a hard threshold on the degrees of vertices that are included in H as in the sequential algorithm, we randomize the selection by using a carefully crafted threshold function μ_H. This function specifies the probability with which a vertex is included in H. It takes as input the ratio of the vertex's degree to the current maximum degree (or, more precisely, the current upper bound on the maximum degree) and it smoothly transitions from

0 to 1 in the neighborhood of the original hard threshold (see Figure 2.1). The main intuition behind the introduction of this function is that we want to ensure that a vertex is *not* selected for H with almost the same probability, independently of the machine on which it resides. Using a hard threshold instead of μ_H could result in the following deficiency. Consider a vertex v that has slightly too few neighbors to qualify as a heavy vertex. Still, it could happen, with a non-negligible probability, that the reference set of some machine contains so many neighbors of v that v would be considered heavy on this machine. However, if v is not included in the set of heavy vertices on that machine, it becomes clear after even a single phase that the vertex is not on the given machine, i.e. the vertex is on the given machine with probability zero. At this point the distribution is clearly no longer uniform.

Function μ_H has further useful properties that we extensively exploit in our analysis. We just note that in order to ensure near uniformity with high probability, we also have to ensure that each vertex is selected for F, the set of friends, with roughly the same probability on each machine.

Organization

We start by analyzing `GlobalAlg` in Section 2.4. Then, Section 2.5 describes how to emulate of a single phase of `GlobalAlg` in the MPC model. Section 2.6 gives and analyzes our parallel algorithm by putting together components developed in the previous sections. The resulting parallel algorithm can be implemented in the MPC model in a fairly straightforward way by using the result of [GSZ11]. The details of the implementation are given in Section 2.7.

2.4 Global Algorithm

2.4.1 Overview

The starting point of our result is a peeling algorithm `GlobalAlg` that takes as input a graph G, and removes from it vertices of lower and lower degree until no edge is left. See page 16 for its pseudocode. We use the term *phase* to refer to an iteration of the main loop in Lines 2–6.

Each phase is associated with a threshold Δ. Initially, Δ equals $\widetilde{\Delta}$, the upper bound on the maximum vertex degree. In every phase, Δ is divided by two until it becomes less than one and the algorithm stops. Since during the execution of the algorithm we maintain the invariant that the maximum degree in the graph is at most Δ, the graph has no edge left when the algorithm terminates.

In each phase the algorithm matches, in expectation, a constant fraction of the vertices it removes. We use this fact to prove that, across all the phases, the algorithm computes a constant-factor approximate matching.

We now describe in more detail the execution of each phase. First, the algorithm creates H, the set of vertices that have degree at least $\Delta/2$ (Line 3). We call these vertices *heavy*. Then, the algorithm uses randomness to create F, a set of *friends* (Line 4). Each vertex v is independently included in F with probability equal to the number of its heavy neighbors divided by 4Δ. We show that $\mathbb{E}[|F|] = O(|H|)$ and $G[H \cup F]$ contains a matching of expected size $\Omega(|H|)$. This kind of matching is likely found by `MatchHeavy` in Line 5.

Note that `GlobalAlg` could as well compute a maximal matching in $G[H \cup F]$ instead of

calling `MatchHeavy`. However, for the purpose of the analysis, using `MatchHeavy` is simpler, as we can directly relate the size of the obtained matching to the size of H. In addition, we later give a parallel version of `GlobalAlg`, and `MatchHeavy` is easy to parallelize.

At the end of the phase, vertices in both H and F are removed from the graph, while the matching found in $G[H \cup F]$ is added to the global matching being constructed. It is easy to see, that by removing H, the algorithm ensures that no vertex of degree larger than $\Delta/2$ remains in the graph, and therefore the bound on the maximum degree decreases by a factor of two.

2.4.2 Analysis

We start our analysis of the algorithm by showing that the execution of `MatchHeavy` in each phase of `GlobalAlg` finds a relatively large matching in expectation.

Lemma 2.3. *Consider one phase of* `GlobalAlg`*. Let H be the set of heavy vertices.* `MatchHeavy` *finds a matching $\widetilde{M}$ such that $\mathbb{E}\left[\left|\widetilde{M}\right|\right] \geq \frac{1}{40}|H|$.*

Proof. Observe that the set $E_\star$ is a collection of vertex-disjoint stars: each edge connects a red vertex with a blue vertex and the red vertices have degree 1. Thus, a subset of $E_\star$ forms a valid matching as long as no blue vertex is incident to two matched edges. Note that this is guaranteed by how edges are added to $\widetilde{M}$ in Line 4.

The size of the computed matching is the number of blue vertices in H that have at least one incident edge in $E_\star$. Let us now lower bound the number of such vertices. Consider an arbitrary $u \in H$. It has the desired properties exactly when the following three independent events happen: some v is selected in F and v selects u in Line 1; u is colored blue; and v is colored red. The joint probability of the two latter events is exactly $\frac{1}{4}$. The probability that u is *not* selected by some its neighbor v (either because v is not selected in F, or v is selected in F but v does not select u in Line 1) is

$$\left(1 - \frac{1}{4\Delta}\right)^{|N(u)\cap V'|} \leq \left(1 - \frac{1}{4\Delta}\right)^{\Delta/2} \leq \exp\left(-\frac{1}{4\Delta} \cdot \frac{\Delta}{2}\right) \leq \exp\left(-\frac{1}{8}\right) \leq \frac{9}{10}.$$

This implies that u is selected by a neighbor $v \in F$ with probability at least $\frac{1}{10}$. Therefore, with probability at least $\frac{1}{10} \cdot \frac{1}{4} = \frac{1}{40}$, u is blue and incident to an edge in $E_\star$. Hence, $\mathbb{E}\left[\left|\widetilde{M}\right|\right] \geq \frac{1}{40}|H|$. □

Next we show an upper bound on the expected size of F, the set of friends.

Lemma 2.4. *Let H be the set of heavy vertices selected in a phase of* `GlobalAlg`*. The following bound holds on the expected size of F, the set of friends, created in the same phase: $\mathbb{E}[|F|] \leq \frac{1}{4}|H|$.*

Proof. At the beginning of a phase, every vertex $u \in V'$—including those in H—has its degree, $|N(u) \cap V'|$, bounded by Δ. Reversing the order of the summation and applying this fact, we get:

$$\mathbb{E}[|F|] = \sum_{v \in V'} \frac{|N(v) \cap H|}{4\Delta} = \sum_{u \in H} \frac{\left|N(u) \cap V'\right|}{4\Delta} \leq \frac{|H| \cdot \Delta}{4\Delta} = \frac{|H|}{4}.$$

□

We combine the last two bounds to lower bound the expected size of the matching computed by `GlobalAlg`.

Lemma 2.5. *Consider an input graph G with an upper bound $\widetilde{\Delta}$ on the maximum vertex degree.* `GlobalAlg`$(G,\widetilde{\Delta})$ *executes* $T \stackrel{\text{def}}{=} \lfloor \log \widetilde{\Delta} \rfloor + 1$ *phases. Let H_i, F_i, and $\widetilde{M}_i$ be the sets H, F, and $\widetilde{M}$ constructed in phase i for $i \in [T]$. The following relationship holds on the expected sizes of these sets:*

$$\sum_{i=1}^{T} \mathbb{E}\left[\left|\widetilde{M}_i\right|\right] \geq \frac{1}{50} \sum_{i=1}^{T} \mathbb{E}\left[|H_i| + |F_i|\right]$$

Proof. For each phase $i \in [T]$, by applying the expectation over all possible settings of the set H_i, we learn from Lemmas 2.3 and 2.4 that

$$\mathbb{E}\left[\left|\widetilde{M}_i\right|\right] \geq \frac{1}{40}\mathbb{E}\left[|H_i|\right] \qquad \text{and} \qquad \mathbb{E}\left[|F_i|\right] \leq \frac{1}{4}\mathbb{E}\left[|H_i|\right].$$

It follows that

$$\frac{1}{50}\mathbb{E}\left[|H_i| + |F_i|\right] \leq \frac{1}{50}\mathbb{E}\left[|H_i|\right] + \frac{1}{200}\mathbb{E}\left[|H_i|\right] = \frac{1}{40}\mathbb{E}\left[|H_i|\right] \leq \mathbb{E}\left[\left|\widetilde{M}_i\right|\right],$$

and the statement of the lemma follows by summing over all phases. □

We do not use this fact directly in our work, but note that the last lemma can be used to show that `GlobalAlg` can be used to find a large matching.

Corollary 2.6. `GlobalAlg` *computes a constant factor approximation to the maximum matching with $\Omega(1)$ probability.*

Proof. First, note that `GlobalAlg` finds a correct matching, i.e., no two different edges in M share an endpoint. This is implied by the fact that M is extended in every phase by a matching on a disjoint set of vertices.

Let T and sets H_i, F_i, and $\widetilde{M}_i$ for $i \in [T]$ be defined as in the statement of Lemma 2.5. Let M_{OPT} be a maximum matching in the graph. Observe that at the end of the algorithm execution, the remaining graph is empty. This implies that the size of the maximum matching can be bounded by the total number of removed vertices, because each removed vertex decreases the maximum matching size by at most one:

$$\sum_{i=1}^{T} |H_i| + |F_i| \geq |M_{\text{OPT}}|.$$

Hence, using Lemma 2.5,

$$\mathbb{E}\left[|M|\right] = \sum_{i=1}^{T} \mathbb{E}\left[\left|\widetilde{M}_i\right|\right] \geq \frac{1}{50}\sum_{i=1}^{T} \mathbb{E}\left[|H_i| + |F_i|\right] \geq \frac{1}{50}|M_{\text{OPT}}|.$$

Since $|M| \leq |M_{\text{OPT}}|$, $|M| \geq \frac{1}{100}|M_{\text{OPT}}|$ with probability at least $\frac{1}{100}$. Otherwise, $\mathbb{E}\left[|M|\right]$ would be strictly less than $\frac{1}{100} \cdot |M_{\text{OPT}}| + 1 \cdot \frac{1}{100}|M_{\text{OPT}}| = \frac{1}{50}|M_{\text{OPT}}|$, which is not possible. □

2.5 Emulation of a Phase in a Randomly Partitioned Graph

In this section, we introduce a modified version of a single phase (one iteration of the main loop) of `GlobalAlg`. Our modifications later allow for implementing the algorithm in the MPC model. The pseudocode of the new procedure, `EmulatePhase`, is presented as Algorithm 3. We partition the vertices of the current graph into m sets V_i, $1 \le i \le m$. Each vertex is assigned independently and *almost* uniformly at random to one of the sets. For each set V_i, we run a subroutine `LocalPhase` (presented as Algorithm 4). This subroutine runs a carefully crafted approximate version of one phase of `GlobalAlg` with an appropriately rescaled threshold Δ. More precisely, the threshold passed to the subroutine is scaled down by a factor of m, which corresponds to how approximately vertex degrees decrease in subgraphs induced by each of the sets. The main intuition behind this modification is that we hope to break the problem up into smaller subproblems on disjoint induced subgraph, and obtain similar *global* properties by solving the problem approximately on each smaller part. Later, in Section 2.6, we design an algorithm that assigns the subproblems to different machines and solves them in parallel.

Algorithm 3: `EmulatePhase`$(\Delta, G_\star, m, \mathscr{D})$
Emulation of a single phase in a randomly partitioned graph

Input:
- threshold Δ
- induced subgraph $G_\star = (V_\star, E_\star)$ of maximum degree $\frac{3}{2}\Delta$
- number m of subgraphs
- ϵ-near uniform and independent distribution $\mathscr{D}$ on assignments of $V_\star$ to $[m]$

Output: Remaining vertices and a matching

1 Pick a random assignment $\Phi : V_\star \to [m]$ from $\mathscr{D}$
2 **for** $i \in [m]$ **do**
3 $\quad V_i \leftarrow \{v \in V_\star : \Phi(v) = i\}$
4 $\quad (V_i', M_i) \leftarrow$ `LocalPhase`$(i, G_\star[V_i], \Delta/m)$ `/* LocalPhase = Algorithm 4 */`
5 **return** $\left(\bigcup_{i=1}^m V_i', \bigcup_{i=1}^m M_i\right)$

We now discuss `LocalPhase` (i.e., Algorithm 4) in more detail. Table 2.2 introduces two parameters, α and μ_R, and two functions, μ_H and μ_F, that are used in `LocalPhase`. Note first that α is a parameter used in the definition of μ_H but it is not used in the pseudocode of `LocalPhase` (or `EmulatePhase`) for anything else. It is, however, a convenient abbreviation in the analysis and the later parallel algorithm. The other three mathematical objects specify probabilities with which vertices are included in sets that are created in an execution of `LocalPhase`.

Apart from creating its own versions of H, the set of heavy vertices, and F, the set of friends, `LocalPhase` constructs also a set R_i, which we refer to as a *reference set*. In Line 1, the algorithm puts each vertex in R_i independently and with the same probability μ_R. The reference set is used to estimate the degrees of other vertices in the same induced subgraph in Line 2. For each vertex v_i, its estimate $\widehat{d}_v$ is defined as the number of v's neighbors in R_i multiplied by μ_R^{-1} to compensate for sampling. Next, in Line 3, the algorithm uses the estimates to create H_i, the set of heavy vertices. Recall that `GlobalAlg` uses a sharp threshold for selecting heavy vertices: all vertices of degree at least $\Delta/2$ are placed in H_i. `LocalPhase` works differently. It divides the degree estimate by the current threshold $\Delta_\star$ and uses function

A multiplicative constant used in the exponent of μ_H:

$$\alpha \stackrel{\text{def}}{=} 96 \ln n.$$

The probability of the selection for a reference set:

$$\mu_R \stackrel{\text{def}}{=} \left(10^6 \cdot \log n\right)^{-1}.$$

The probability of the selection for a heavy set (used with r equal to the ratio of the estimated degree to the current threshold):

$$\mu_H(r) \stackrel{\text{def}}{=} \begin{cases} \frac{1}{2}\exp\left(\frac{\alpha}{2}(r-1/2)\right) & \text{if } r \le 1/2, \\ 1-\frac{1}{2}\exp\left(-\frac{\alpha}{2}(r-1/2)\right) & \text{if } r > 1/2. \end{cases}$$

The probability of the selection for the set of friends (used with r equal to the ratio of the number of heavy neighbors to the current threshold):

$$\mu_F(r) \stackrel{\text{def}}{=} \begin{cases} \max\{r/4, 0\} & \text{if } r \le 4, \\ 1 & \text{if } r > 4. \end{cases}$$

Table 2.2 – Global parameters $\alpha \in (1,\infty)$ and $\mu_R \in (0,1)$ and functions $\mu_H : \mathbb{R} \to [0,1]$ and $\mu_F : \mathbb{R} \to [0,1]$ used in the parallel algorithm. α, μ_R, and μ_H depend on n, the total number of vertices in the graph.

μ_H to decide with what probability the corresponding vertex is included in H_i. A sketch of the function can be seen in Figure 2.1. The function transitions from almost 0 to almost 1 in the neighborhood of $\frac{1}{2}$ at a limited pace. As a result vertices of degrees smaller than, say, $\frac{1}{4}\Delta$ are very unlikely to be included in H_i and vertices of degree greater than $\frac{3}{4}\Delta$ are very likely to be included in H_i. `GlobalAlg` can be seen as an algorithm that instead of μ_H, uses a step function that equals 0 for arguments less than $\frac{1}{2}$ and abruptly jumps to 1 for larger arguments. Observe that without μ_H, the vertices whose degrees barely qualify them as heavy could behave very differently depending on which set they were assigned to. We use μ_H to guarantee a smooth behavior in such cases. That is one of the key ingredients that we need for making sure that a set of vertices that remains on one machine after a phase has almost the same statistical properties as a set of vertices obtained by new random partitioning.

Finally, in Line 4, `LocalPhase` creates a set of friends. This step is almost identical to what happens in the global algorithm. The only difference is that this time we have no upper bound on the number of heavy neighbors of a vertex. As a result that number divided by $4\Delta_\star$ can be greater than 1, in which case we have to replace it with 1 in order to obtain a proper probability. This is taken care of by function μ_F. Once H_i and F_i have been created, the algorithm finds a maximal matching M_i in the subgraph induced by the union of these two sets. The algorithm discards from the further consideration not only H_i and F_i, but also R_i. This eliminates dependencies in the possible distribution of assignments of vertices that have not been removed yet if we condition this distribution on the configuration of sets that have been removed. Intuitively, the probability of a vertex's inclusion in any of these sets depends only on R_i and H_i but not on any other vertices. Hence, once we fix the sets of removed vertices,

Algorithm 4: `LocalPhase`$(i, G_i, \Delta_\star)$
Emulation of a single phase on an induced subgraph

Input:
- induced subgraph number i (useful only for the analysis)
- induced subgraph $G_i = (V_i, E_i)$
- threshold $\Delta_\star \in \mathbb{R}_+$

Output: Remaining vertices and a matching on V_i

1 Create a *reference set* R_i by independently selecting each vertex in V_i with probability μ_R.
2 For each $v \in V_i$, $\widehat{d}_v \leftarrow |N(v) \cap R_i|/\mu_R$.
3 Create a set H_i of *heavy vertices* by independently selecting each $v \in V_i$ with probability $\mu_H\left(\widehat{d}_v/\Delta_\star\right)$.
4 Create a set F_i of *friends* by independently selecting each vertex in $v \in V_i$ with probability $\mu_F\left(|N(v) \cap H_i|/\Delta_\star\right)$.
5 Compute a maximal matching M_i in $G[H_i \cup F_i]$.
6 **return** $(V_i \setminus (R_i \cup H_i \cup F_i), M_i)$

the assignment of the remaining vertices to subgraphs is fully independent.[2] The output of `LocalPhase` is a subset of V_i to be considered in later phases and a matching M_i, that is used to expand the matching that we construct for the entire input graph. We now introduce additional concepts and notation. They are useful for describing and analyzing properties of the algorithm. A configuration describes sets R_i, H_i, and F_i, for $1 \le i \le m$, constructed in an execution of `EmulatePhase`. We use it for conditioning a distribution of vertex assignments as described in the previous paragraph. We also formally define two important properties of distributions of vertex assignments: independence and near uniformity.

Configurations. Let m and $V_\star$ be the parameters to `EmulatePhase`: the number of subgraphs and the set of vertices in the graph to be partitioned, respectively. We say that

$$\mathscr{C} = \left(\{R_i\}_{i\in[m]}, \{H_i\}_{i\in[m]}, \{F_i\}_{i\in[m]}\right)$$

is an *m-configuration* if it represents a configuration of sets R_i, H_i, and F_i created by `EmulatePhase` in the simulation of a phase. Recall that for any $i \in [m]$, R_i, H_i, and F_i are the sets created (and removed) by the execution of `LocalPhase` for V_i, the i-th subset of vertices.

We say that a vertex v is *fixed* by $\mathscr{C}$ if it belongs to one of the sets in the configuration, i.e.,

$$v \in \bigcup_{i\in[m]} (R_i \cup H_i \cup F_i)\,.$$

Conditional distribution. Let $\mathscr{D}$ be a distribution on assignments $\varphi : V_\star \to [m]$. Suppose that we execute `EmulatePhase` for $\mathscr{D}$ and let $\mathscr{C}$ be a *non-zero probability* m-configuration—composed of sets R_i, H_i, and F_i for $i \in [m]$—that can be created in this setting. Let $V'_\star$ be the set of vertices in $V_\star$ that are *not* fixed by $\mathscr{C}$. We write $\mathscr{D}[\mathscr{C}]$ to denote the conditional distribution of possible assignments of vertices in $V'_\star$ to $[m]$, given that for all $i \in [m]$, R_i, H_i, and F_i in $\mathscr{C}$ were the sets constructed by `LocalPhase` for the i-th induced subgraph.

[2] By way of comparison, consider observing an experiment in which we toss the same coin twice. The bias of the coin is not fixed but comes from a random distribution. If we do not know the bias, the outcomes of the coin tosses are not independent. However, if we do know the bias, the outcomes are independent, even though they have the same bias.

Near uniformity and independence. Let $\mathscr{D}$ be a distribution on assignments $\varphi : \widetilde{V} \to [m]$ for some set $\widetilde{V}$ and m. For each vertex $v \in \widetilde{V}$, let $p_v : [m] \to [0,1]$ be the probability mass function of the marginal distribution of v's assignment. For any $\epsilon \geq 0$, we say that $\mathscr{D}$ is *ϵ-near uniform* if for every vertex v and every $i \in [m]$, $p_v(i) \in [\![(1 \pm \epsilon)/m]\!]$. We say that $\mathscr{D}$ is an *independent* distribution if the probability of every assignment φ in $\mathscr{D}$ equals exactly $\prod_{v \in V'} p_v(\varphi(v))$.

Concentration inequality. We use the following version of the Chernoff bound that depends on an upper bound on the expectation of the underlying independent random variables. It can be shown by combining two applications of the more standard version.

Lemma 2.7 (Chernoff bound)**.** *Let $X_1, \ldots, X_k$ be independently distributed random variables taking values in $[0,1]$. Let $X \stackrel{\text{def}}{=} X_1 + \cdots + X_k$ and let $U \geq 0$ be an upper bound on the expectation of X, i.e., $\mathbb{E}[X] \leq U$. For any $\delta \in [0,1]$, $\Pr(|X - \mathbb{E}[X]| > \delta U) \leq 2\exp(-\delta^2 U/3)$.*

Concise range notation. Multiple times throughout this chapter, we want to denote a range around some value. Instead of writing, say, $[x-\delta, x+\delta]$, we introduce a more concise notation. In this specific case, we would simply write $[\![x \pm \delta]\!]$. More formally, let E be a numerical expression that apart from standard operations also contains a single application of the binary or unary operator $\pm$. We create two standard numerical expressions from E: E_- and E_+ that replace $\pm$ with $-$ and $+$, respectively. Now we define $[\![E]\!] \stackrel{\text{def}}{=} [\min\{E_-, E_+\}, \max\{E_-, E_+\}]$.

As another example, consider $E = \sqrt{101 \pm 20}$. We have $E_- = \sqrt{101 - 20} = 9$ and $E_+ = \sqrt{101 + 20} = 11$. Hence $[\![\sqrt{101 \pm 20}]\!] = [\min\{9,11\}, \max\{9,11\}] = [9,11]$.

We now show the properties of `EmulatePhase` that we use to obtain our final parallel algorithm.

2.5.1 Outline of the Section

We start by showing that `EmulatePhase` computes a large matching as follows. Each vertex belonging to H_i or F_i that `EmulatePhase` removes in the calls to `LocalPhase` can decrease the maximum matching size in the graph induced by the remaining vertices by one. We show that the matching that `EmulatePhase` constructs in the process captures on average at least a constant fraction of that loss. We also show that the effect of removing R_i is negligible. More precisely, in Section 2.5.2 we prove the following lemma.

Lemma 2.8. *Let Δ, $G_\star = (V_\star, E_\star)$, m, and $\mathscr{D}$ be parameters for* `EmulatePhase` *such that*

- *$\mathscr{D}$ is an independent and ϵ-near uniform distribution on assignments of vertices $V_\star$ to $[m]$ for $\epsilon \in [0, 1/200]$,*
- *$\frac{\Delta}{m} \geq 4000\mu_R^{-2} \ln^2 n$,*
- *the maximum degree of a vertex in $G_\star$ is at most $\frac{3}{2}\Delta$.*

For each $i \in [m]$, let H_i, F_i, and M_i be the sets constructed by `LocalPhase` *for the i-th induced subgraph. Then, the following relationship holds for their expected sizes:*

$$\sum_{i \in [m]} \mathbb{E}\left[|H_i \cup F_i|\right] \leq n^{-9} + 1200 \sum_{i \in [m]} \mathbb{E}\left[|M_i|\right].$$

Note that Lemma 2.8 requires that the vertices are distributed independently and near uniformly in the m sets. This is trivially the case right after the vertices are partitioned independently at random. However, in the final algorithm, after we partition the vertices, we run *multiple* phases on each machine. In the rest of this section we show that running a single phase *preserves* independence of vertex distribution and only slightly disturbs the uniformity (Lemma 2.9 and Lemma 2.10). As we have mentioned before, independence stems from the fact that we use reference sets to estimate vertex degrees. We discard them at the end and condition on them, which leads to the independence of the distribution of vertices that are not removed.

Lemma 2.9. *Let $\mathscr{D}$ be an independent distribution of assignments of vertices in $V_\star$ to $[m]$. Let $\mathscr{C}$ be a non-zero probability m-configuration that can be constructed by* `EmulatePhase` *for $\mathscr{D}$. Let $V'_\star$ be the set of vertices of $V_\star$ that are not fixed by $\mathscr{C}$. Then $\mathscr{D}[\mathscr{C}]$ is an independent distribution of vertices in $V'_\star$ on $[m]$.*

Independence of the vertex assignment is a very handy feature that allows us to use Chernoff-like concentration inequalities in the analysis of multiple phase emulation. However, although the vertex assignment of non-removed vertices remains independent across machines from phase to phase, as stated by Lemma 2.9, their distribution is not necessarily uniform. Fortunately, we can show it is near uniform.
The proof of near uniformity is the most involved proof in this chapter. In a nutshell, the proof is structured as follows. We pick an arbitrary vertex v that has not been removed and show that with high probability it has the same number of neighbors in all sets R_i. The same property holds for v's neighbors in all sets H_i. We use this to show that the probability of a fixed configuration of sets removed in a single phase is roughly the same for all assignments of v to subgraphs. In other words, if v was distributed nearly uniformly before the execution of `EmulatePhase`, it is distributed only slightly less uniformly after the execution.

Lemma 2.10. *Let Δ, $G_\star = (V_\star, E_\star)$, m, and $\mathscr{D}$ be parameters for* `EmulatePhase` *such that*

- *$\mathscr{D}$ is an independent and ϵ-near uniform distribution on assignments of vertices $V_\star$ to $[m]$ for $\epsilon \in [0, (200 \ln n)^{-1}]$,*
- *$\frac{\Delta}{m} \geq 4000 \mu_R^{-2} \ln^2 n$.*

Let $\mathscr{C}$ be an m-configuration constructed by `EmulatePhase`*. With probability at least $1 - n^{-4}$ both the following properties hold:*

- *The maximum degree in the graph induced by the vertices not fixed in $\mathscr{C}$ is bounded by $\frac{3}{4}\Delta$.*
- *$\mathscr{D}[\mathscr{C}]$ is $60\alpha\left(\left(\frac{\Delta}{m}\right)^{-1/4} + \epsilon\right)$-near uniform.*

2.5.2 Expected Matching Size

Now we prove Lemma 2.8, i.e. we show that `EmulatePhase` computes a large matching. In the proof we argue that the expected total size of sets H_i and F_i is not significantly impacted by relatively low-degree vertices classified as heavy or by an unlucky assignment of vertices to

subgraphs resulting in local vertex degrees not corresponding to global degrees. Namely, we show that the expected number of friends a heavy vertex adds is $O(1)$ and at the same time the probability that the vertex gets matched is $\Omega(1)$.

Lemma 2.8. *Let Δ, $G_\star = (V_\star, E_\star)$, m, and $\mathscr{D}$ be parameters for* `EmulatePhase` *such that*

- *$\mathscr{D}$ is an independent and ϵ-near uniform distribution on assignments of vertices $V_\star$ to $[m]$ for $\epsilon \in [0, 1/200]$,*
- *$\frac{\Delta}{m} \geq 4000\mu_R^{-2}\ln^2 n$,*
- *the maximum degree of a vertex in $G_\star$ is at most $\frac{3}{2}\Delta$.*

For each $i \in [m]$, let H_i, F_i, and M_i be the sets constructed by `LocalPhase` *for the i-th induced subgraph. Then, the following relationship holds for their expected sizes:*

$$\sum_{i\in[m]} \mathbb{E}\left[|H_i \cup F_i|\right] \leq n^{-9} + 1200 \sum_{i\in[m]} \mathbb{E}\left[|M_i|\right].$$

Proof. We borrow more notation from `EmulatePhase` and the m executions of `LocalPhase` initiated by it. For $i \in [m]$, V_i is the set inducing the i-th subgraph. Value $\Delta_\star = \frac{\Delta}{m}$ is the rescaled threshold passed to the executions of `LocalPhase`. R_i is the reference set created by `LocalPhase` for the i-th induced subgraph.

For each induced subgraph, `LocalPhase` computes a maximal matching M_i in Line 5. While such a matching is always large—its size is at least half the maximum matching size—it is hard to relate its size directly to the sizes of H_i and F_i. Therefore, we first analyze the size of a matching that would be created by `MatchHeavy`$(G_\star[H_i \cup F_i], H_i, F_i)$. We refer to this matching as $\widetilde{M}_i$ and we later use the inequality $\left|\widetilde{M}_i\right| \leq 2|M_i|$.

We partition each H_i, $i \in [m]$, into two sets: H_i' and H_i''. H_i' is the subset of vertices in H_i of degree less than $\frac{1}{8}\Delta$ in $G_\star$. $H_{i,t+1}''$ is its complement, i.e., $H_i'' \stackrel{\text{def}}{=} H_i \setminus H_i'$. We start by bounding the expected total size of sets H_i'. What is the probability that a given vertex v of degree less than $\frac{1}{8}\Delta$ is included in $\bigcup_{i\in[m]} H_i$? Suppose that $v \in V_k$, where $k \in [m]$. The expected number of v's neighbors in R_k is at most $(1+\epsilon)\cdot\mu_R\cdot\frac{1}{8}\Delta/m \leq \frac{3}{16}\mu_R\Delta_\star$ due to the independence and ϵ-near uniformity of $\mathscr{D}[\mathscr{C}]$. Using the independence, Lemma 2.7, and the lower bound on $\Delta_\star$, we obtain the following bound:

$$\Pr\left[\mu_R\widehat{d}_v > \frac{1}{4}\mu_R\Delta_\star\right] \leq 2\exp\left(-\frac{1}{3}\cdot\left(\frac{1}{3}\right)^2\cdot\frac{3}{16}\mu_R\Delta_\star\right) \leq 2\exp\left(-27\ln n\right) = 2n^{-27}.$$

If $\widehat{d}_v \leq \frac{1}{4}\Delta_\star$, the probability that v is selected to H_k is at most $\mu_H(\widehat{d}_v/\Delta_\star) \leq \mu_H(1/4) \leq \frac{1}{2}n^{-12}$. Hence v is selected to H_k—and therefore to H_k'—with probability at most $2n^{-27} + \frac{1}{2}n^{-12} \leq n^{-12}$. This implies that $\sum_{i\in[m]}\mathbb{E}\left[\left|H_i'\right|\right] \leq n\cdot n^{-12} = n^{-11}$.

We also partition the sets of friends, F_i for $i \in [m]$, into two sets each: F_i' and F_i''. This partition is based on the execution of `MatchHeavy` for the i-th subgraph. In Line 1, this algorithm selects for every vertex $v \in F_i$ a random heavy neighbor $v_\star \in H_i$. If $v_\star \in H_i'$, we assign v to F_i'. Analogously, if $v_\star \in H_i''$, we assign v to F_i''. Obviously, a heavy vertex in H_i' can be selected only if H_i' is non-empty. By Markov's inequality and the upper bound on $\sum_{i\in[m]}\mathbb{E}\left[\left|H_i'\right|\right]$, the probability that at least one set H_i' is non-empty is at most n^{-11}. Even if

for all $i \in [m]$, all vertices in F_i select a heavy neighbor in H'_i whenever it is available, the total expected number of vertices in sets F'_i is at most $\sum_{i\in[m]} \mathbb{E}\left[\left|F'_{i,t+1}\right|\right] \leq n \cdot n^{-11} = n^{-10}$.

Before we proceed to bounding sizes of the remaining sets, we prove that with high probability, all vertices have a number of neighbors close to the expectation. Let $\varphi : V_\star \to [m]$ be the assignment of vertices to subgraphs. We define $\mathscr{E}$ as the event that for all $v \in V_\star$,

$$\left|\frac{1}{m}|N(v)\cap V_\star| - \left|N(v)\cap V_{\varphi(v)}\right|\right| \leq \frac{1}{16}\Delta_\star.$$

Consider first one fixed $v \in V_\star$. The degree of v in $G_\star$ is $|N(v)\cap V_\star| \leq \frac{3}{2}\Delta$. Due to the near-uniformity and independence,

$$\left|\frac{1}{m}|N(v)\cap V_\star| - \mathbb{E}\left[\left|N(v)\cap V_{\varphi(v)}\right|\right]\right| \leq \epsilon\cdot\frac{3}{2}\frac{\Delta}{m} \leq \frac{3}{400}\Delta_\star.$$

This in particular implies that $\mathbb{E}\left[\left|N(v)\cap V_{\varphi(v)}\right|\right] \leq \left(\frac{3}{2}+\frac{3}{400}\right)\Delta_\star \leq 2\Delta_\star$. Using the independence of $\mathscr{D}$, Lemma 2.7, and the lower bound on $\Delta_\star$ (i.e., $\Delta_\star = \frac{\Delta}{m} \geq 4000\mu_R^{-2}\ln^2 n = 4\cdot 10^{15}\cdot\ln^4 n$),

$$\begin{aligned}\Pr\left[\left|\mathbb{E}\left[\left|N(v)\cap V_{\varphi(v)}\right|\right] - \left|N(v)\cap V_{\varphi(v)}\right|\right| > \frac{1}{20}\Delta_\star\right] &\leq 2\exp\left(-\frac{1}{3}\cdot\left(\frac{1}{20}\cdot\frac{1}{2}\right)^2\cdot 2\Delta_\star\right)\\ &\leq 2\exp\left(-(10^{12}+3)\ln n\right)\\ &\leq n^{-(10^{12}+2)} \leq n^{-12}.\end{aligned}$$

As a result, with this probability, we have

$$\left|\frac{1}{m}|N(v)\cap V_\star| - \left|N(v)\cap V_{\varphi(v)}\right|\right| \leq \frac{1}{20}\Delta_\star + \frac{3}{400}\Delta_\star \leq \frac{1}{16}\Delta_\star.$$

By the union bound, this bound holds for all vertices in $V_\star$ simultaneously—and hence $\mathscr{E}$ occurs—with probability at least $1 - n\cdot n^{-12} = 1 - n^{-11}$.

If $\mathscr{E}$ does not occur, we can bound both $\sum_{i\in[m]}\left|H''_i\right|$ and $\sum_{i\in[m]}\left|F''_i\right|$ by n. This contributes at most $n^{-11}\cdot n = n^{-10}$ to the expected size of each of these quantities. Suppose now that $\mathscr{E}$ occurs. Consider an arbitrary $v \in H''_i$ for some i. The number of neighbors of v in V_i lies in the range $\left[\frac{1}{8}\Delta_\star - \frac{1}{16}\Delta_\star, \frac{3}{2}\Delta_\star + \frac{1}{16}\Delta_\star\right] \subseteq \left[\frac{1}{16}\Delta_\star, 2\Delta_\star\right]$. Moreover, the expected number of vertices $w \in F''_i$ that select v in $w_\star$ in Line 1 of `MatchHeavy` is bounded by $2\Delta_\star\cdot\frac{1}{4\Delta_\star} = \frac{1}{2}$. It follows that $\mathbb{E}\left[\left|F''_i\right|\right] \leq \frac{1}{2}\mathbb{E}\left[|H''_i|\right]$, given $\mathscr{E}$. We now lower bound the expected size of $\widetilde{M}_i$ given $\mathscr{E}$. What is the probability that some vertex $w \in F_i$ selects v as $w_\star$ in `MatchHeavy` and (v,w) is added to $\widetilde{M}_i$?

This occurs if one of v's neighbors w is added to F_i and selects v as $w_\star$, and additionally, v and w are colored blue and red, respectively. The number of v's neighbors is at least $\frac{1}{16}\Delta_\star$. Since each vertex w in V_i has at most $2\Delta_\star$ neighbors, the number of heavy neighbors of w is bounded by the same number. This implies that in the process of selecting F_i, only the first branch in the definition of μ_F is used and each vertex w is included with probability exactly equal to the number of its neighbors in H_i divided by $4\Delta_{t+1}$. Then each heavy neighbor of w is selected as $w_\star$ with probability one over the number of heavy neighbors of w. What this implies is that each neighbor w of v is selected for F_i *and* selects v as $w_\star$ with probability

exactly $(4\Delta_\star)^{-1}$. Hence the probability that v is *not* selected as $w_\star$ by any of its at least $\frac{1}{16}\Delta_\star$ neighbors w can be bounded by

$$\left(1-\frac{1}{4\Delta_\star}\right)^{\frac{1}{16}\Delta_\star} \leq \exp\left(-\frac{1}{4\Delta_\star}\cdot\frac{1}{16}\Delta_\star\right) = e^{-1/64}.$$

Therefore the probability that v is selected by some vertex $w \in F_i$ as $w_\star$ is at least $1-e^{-1/64} \geq 1/100$. Then with probability $1/4$, these two vertices have appropriate colors and this or another edge incident to v with the same properties is added to $\widetilde{M}_i$. In summary, the probability that an edge (v,w) for some w as described is added to $\widetilde{M}_i$ is at least $1/400$. Since we do not count any edge in the matching twice for two heavy vertices, by the linearity of expectation $\mathbb{E}\left[\left|\widetilde{M}_i\right|\right] \geq \frac{1}{400}\mathbb{E}\left[|H_i''|\right]$ given $\mathscr{E}$. Overall, given $\mathscr{E}$, we have

$$\sum_{i\in[m]} \mathbb{E}\left[\left|H_i''\right| + \left|F_i''\right|\right] \leq \frac{3}{2}\sum_{i\in[m]} \mathbb{E}\left[\left|H_i''\right|\right] \leq 600 \sum_{i\in[m]} \mathbb{E}\left[\left|\widetilde{M}_i\right|\right].$$

In general, without conditioning on $\mathscr{E}$,

$$\sum_{i\in[m]} \mathbb{E}\left[\left|H_i''\right| + \left|F_i''\right|\right] \leq 2\cdot n^{-10} + 600 \sum_{i\in[m]} \mathbb{E}\left[\left|\widetilde{M}_i\right|\right].$$

We now combine bounds on all terms to finish the proof of the lemma.

$$\begin{aligned}\sum_{i\in[m]} \mathbb{E}\left[|H_i \cup F_i|\right] &\leq \sum_{i\in[m]} \mathbb{E}\left[\left|H_i'\right| + \left|F_i'\right| + \left|H_i''\right| + \left|F_i''\right|\right] \\ &\leq n^{-11} + n^{-10} + 2n^{-10} + 600 \sum_{i\in[m]} \mathbb{E}\left[\left|\widetilde{M}_i\right|\right] \\ &\leq n^{-9} + 1200 \sum_{i\in[m]} \mathbb{E}\left[|M_i|\right].\end{aligned}$$

□

2.5.3 Independence

Next we prove Lemma 2.9. We start with an auxiliary lemma that gives a simple criterion under which an independent distribution remains independent after conditioning on a random event. Consider a random vector with independently distributed coordinates. Suppose that for any value of the vector, a random event $\mathscr{E}$ occurs when all coordinates "cooperate", where each coordinate cooperates independently with probability that depends only on the value of that coordinate. We then show that the distribution of the vector's coordinates given $\mathscr{E}$ remains independent.

Lemma 2.11. *Let k be a positive integer and A an arbitrary finite set. Let $X = (X_1,\ldots,X_k)$ be a random vector in A^k with independently distributed coordinates. Let $\mathscr{E}$ be a random event of non-zero probability. If there exist functions $p_i : A \to [0,1]$, for $i \in [k]$, such that for any $x = (x_1,\ldots,x_k) \in A^k$ appearing with non-zero probability,*

$$\Pr[\mathscr{E}|X = x] = \prod_{i=1}^{k} p_i(x_i),$$

then the conditional distribution of coordinates in X given $\mathscr{E}$ is independent as well.

Proof. Since the distribution of coordinates in X is independent, there are k probability mass functions $p'_i : A \to [0,1]$, $i \in [k]$, such that for every $x = (x_1, \ldots, x_k) \in A^k$, $\Pr[X = x] = \prod_{i=1}^{k} p'_i(x_i)$. The probability of $\mathscr{E}$ can be expressed as

$$\Pr[\mathscr{E}] = \sum_{x=(x_1,\ldots,x_k)\in A^k} \Pr[\mathscr{E} \wedge X = x] = \sum_{\substack{x=(x_1,\ldots,x_k)\in A^k \\ \Pr[X=x]>0}} \Pr[\mathscr{E}|X = x] \cdot \Pr[X = x]$$

$$= \sum_{x=(x_1,\ldots,x_k)\in A^k} \prod_{i=1}^{k} p_i(x_i)p'_i(x_i) = \prod_{i=1}^{k} \sum_{y\in A} p_i(y)p'_i(y).$$

Note that since the probability of $\mathscr{E}$ is positive, each multiplicative term $\sum_{y\in A} p_i(y)p'_i(y)$, $i \in [k]$, in the above expression is positive. We can express the probability of any vector $x = (x_1, \ldots, x_k) \in A^k$ given $\mathscr{E}$ as follows:

$$\Pr[X = x|\mathscr{E}] = \frac{\Pr[\mathscr{E} \wedge X = x]}{\Pr[\mathscr{E}]} = \frac{\Pr[\mathscr{E}|X = x] \cdot \Pr[X = x]}{\Pr[\mathscr{E}]}$$

$$= \frac{\prod_{i=1}^{k} p_i(x_i)p'_i(x_i)}{\prod_{i=1}^{k} \sum_{y\in A} p_i(y)p'_i(y)} = \prod_{i=1}^{k} \frac{p_i(x_i)p'_i(x_i)}{\sum_{y\in A} p_i(y)p'_i(y)}.$$

We define $p''_i : A \to [0,1]$ as $p''_i(x) \stackrel{\text{def}}{=} p_i(x_i)p'_i(x_i)/\sum_{y\in A} p_i(y)p'_i(y)$ for each $i \in [k]$. Each p''_i is a valid probability mass function on A. As a result we have $\Pr[X = x|\mathscr{E}] = \prod_{i=1}^{k} p''_i(x_i)$, which proves that the distribution of coordinates in X given $\mathscr{E}$ is still independent with each coordinate distributed according to its probability mass function p''_i. □

We now prove Lemma 2.9 by applying Lemma 2.11 thrice. We refer to functions p_i, that describe the probability of each coordinate cooperating, as *cooperation probability functions*.

Lemma 2.9. *Let $\mathscr{D}$ be an independent distribution of assignments of vertices in $V_\star$ to $[m]$. Let $\mathscr{C}$ be a non-zero probability m-configuration that can be constructed by* `EmulatePhase` *for $\mathscr{D}$. Let $V'_\star$ be the set of vertices of $V_\star$ that are not fixed by $\mathscr{C}$. Then $\mathscr{D}[\mathscr{C}]$ is an independent distribution of vertices in $V'_\star$ on $[m]$.*

Proof. $\mathscr{C}$ can be expressed as

$$\mathscr{C} = \left(\{R^\star_i\}_{i\in[m]}, \{H^\star_i\}_{i\in[m]}, \{F^\star_i\}_{i\in[m]}\right)$$

for some subsets $R^\star_i$, $H^\star_i$, and $F^\star_i$ of $V_\star$, where $i \in [m]$. We write Φ to denote the random assignment of vertices to sets selected in Line 1 of `EmulatePhase`. Φ is a random variable distributed according to $\mathscr{D}$.

Let $\mathscr{E}_R$ be the event that for all $i \in [m]$, the reference set R_i generated for the i-th induced subgraph by `LocalPhase` equals exactly $R^\star_i$. A vertex v that is assigned to a set V_i is included in R_i with probability exactly μ_R, independently of other vertices. Hence once we fix an assignment $\varphi : V_\star \to [m]$ of vertices to sets V_i, we can express the probability of $\mathscr{E}_R$ as a product of probabilities that each vertex cooperates. More formally, $\Pr[\mathscr{E}_R|\Phi = \varphi] = \prod_{v\in V_\star} q_v(\varphi(v))$ for cooperation probability functions $q_v : [m] \to [0,1]$ defined as follows.

- If $v \in \bigcup_{i\in[m]} R^\star_i$, there is exactly one $i \in [m]$ such that $v \in R^\star_i$. If v is not assigned to V_i, $\mathscr{E}_R$ cannot occur. If it is, v cooperates with $\mathscr{E}_R$ with probability exactly μ_R, i.e., the

probability of the selection for R_i. For this kind of v, the cooperation probability function is

$$q_v(i) \stackrel{\text{def}}{=} \begin{cases} \mu_R & \text{if } v \in R_i^\star, \\ 0 & \text{if } v \notin R_i^\star. \end{cases}$$

- If $v \notin \bigcup_{i\in[m]} R_i^\star$, v cooperates with $\mathscr{E}_R$ if it is not selected for $R_{\varphi(v)}$, independently of its assignment $\varphi(v)$, which happens with probability exactly $1-\mu_R$. Therefore, the cooperation probability can be defined as $q_v(i) \stackrel{\text{def}}{=} 1-\mu_R$ for all $i \in [m]$.

We invoke Lemma 2.11 to conclude that the conditional distribution of values of Φ given $\mathscr{E}_R$ is independent as well.

We now define an event $\mathscr{E}_H$ that both $\mathscr{E}_R$ occurs and for all $i \in [m]$, H_i, the set of heavy vertices constructed for the i-th subgraph equals exactly $H_i^\star$. We want to show that the conditional distribution of values of Φ given $\mathscr{E}_H$ is independent. Note that if Φ is selected from the conditional distribution given $\mathscr{E}_R$ (i.e., all sets R_i are as expected) and we fix the assignment $\phi : V_\star \to [m]$ of vertices to sets V_i, then each vertex $v \in V_\star$ is assigned to $H_{\phi(v)}$—this the only set H_i to which it can be assigned—independently of other vertices. As a result, we can express the probability of $\mathscr{E}_H$ given $\mathscr{E}_R$ and φ being the assignment as a product of cooperation probabilities for each vertex. More precisely, $\Pr[\mathscr{E}_H|\Phi=\varphi,\mathscr{E}_R] = \prod_{v\in V_\star} q'_v(\varphi(v))$ for cooperation probability functions $q'_v : [m] \to [0,1]$ defined as follows, where $\Delta_\star$ is the threshold used in the m executions of `LocalPhase`.

- If $v \in \bigcup_{i\in[m]} H_i^\star$, then there is exactly one i such that $v \in H_i^\star$. $\mathscr{E}_H$ can only occur if v is included in the corresponding H_i. This cannot happen if v is not assigned to the corresponding V_i by φ. If v is assigned to this V_i, it has to be selected for H_i, which happens with probability $\mu_H\left(|N(v)\cap R_i^\star|/(\mu_R\Delta_\star)\right)$. The cooperation probability function can be written in this case as

$$q'_v(i) \stackrel{\text{def}}{=} \begin{cases} \mu_H(|N(v)\cap R_i^\star|/(\mu_R\Delta_\star)) & \text{if } v \in H_i^\star, \\ 0 & \text{if } v \notin H_i^\star. \end{cases}$$

- If $v \notin \bigcup_{i\in[m]} H_i^\star$, v cannot be included in H_i corresponding to the set V_i to which it is assigned for $\mathscr{E}_H$ to occur. This happens with probability $1-\mu_H(|N(v)\cap R_i^\star|/(\mu_R\Delta_\star))$. Hence, we can define $q'_v(i) \stackrel{\text{def}}{=} 1-\mu_H(|N(v)\cap R_i^\star|/(\mu_R\Delta_\star))$ for all $i \in [m]$.

We can now invoke Lemma 2.11 to conclude that the distribution of values of Φ given $\mathscr{E}_H$ is independent.

Finally, we define $\mathscr{E}_F$ to be the event that both $\mathscr{E}_H$ occurs and for each $i \in [m]$, F_i, the set of friends selected for the i-th induced subgraph, equals exactly $F_i^\star$. We observe that once Φ is fixed to a specific assignment $\varphi : V_\star \to [m]$ and $\mathscr{E}_H$ occurs (i.e., all sets R_i and H_i are as in $\mathscr{C}$), then each vertex is independently included in $F_{\varphi(v)}$ with some specific probability that depends only on $H_{\varphi(v)}$, which is already fixed. In this setting, we can therefore express the probability of $\mathscr{E}_F$, which exactly specifies the composition of sets F_i, as a product of values provided by some cooperation probability functions $q''_v : [m] \to [0,1]$. More precisely, $\Pr[\mathscr{E}_F|\Phi=\varphi,\mathscr{E}_H] = \prod_{v\in V_\star} q''_v(\varphi(v))$ for q''_v that we define next.

- If $v \in \bigcup_{i\in[m]} F_i^\star$, then there is exactly one i such that $v \in F_i^\star$. $\mathscr{E}_F$ cannot occur if v is not assigned to V_i and selected for F_i. Hence, the cooperation probability function for v is

$$q_v''(i) \stackrel{\text{def}}{=} \begin{cases} \mu_F(|N(v) \cap H_i^\star|/\Delta_\star) & \text{if } v \in F_i^\star, \\ 0 & \text{if } v \notin F_i^\star. \end{cases}$$

- If $v \notin \bigcup_{i\in[m]} F_i^\star$, to whichever set V_i vertex v is assigned, it should not be included in F_i in order for $\mathscr{E}_F$ to occur. Hence, $q_v''(i) \stackrel{\text{def}}{=} 1 - \mu_F(|N(v) \cap H_{i_\star,t}^\star|/\Delta_t)$.

We invoke Lemma 2.11 to conclude that the distribution of values of Φ given $\mathscr{E}_F$ is independent as well. This is a distribution on assignments for the entire set $V_\star$. If we restrict it to assignments of $V_\star' \subseteq V_\star$, we obtain a distribution that first, is independent as well, and second, equals exactly $\mathscr{D}[\mathscr{C}]$. □

2.5.4 Near Uniformity

In this section we prove Lemma 2.10. We begin by showing a useful property of μ_H (see Table 2.2 for definition). Recall that `GlobalAlg` selects H, the set of heavy vertices, by taking all vertices of degree at least $\Delta/2$. In `LocalPhase` the degree estimate of each vertex depends on the number of neighbors in the reference set in the vertex's induced subgraph. We want the decision taken for each vertex to be approximately the same, independently of which subgraph it is assigned to. Therefore, we use μ_H—which specifies the probability of the inclusion in the set of heavy vertices—which is relatively insensitive to small argument changes. The next lemma proves that this is indeed the case. Small additive changes to the parameter x to μ_H have small multiplicative impact on both $\mu_H(x)$ and $1 - \mu_H(x)$.

Lemma 2.12 (Insensitivity of μ_H)**.** *Let $\delta \in [0, (\alpha/2)^{-1}] = [0, (48 \ln n)^{-1}]$. For any pair x and x' of real numbers such that $|x - x'| \le \delta$,*

$$\mu_H(x') \in [\![\mu_H(x)(1 \pm \alpha\delta)]\!]$$

and

$$1 - \mu_H(x') \in [\![(1 - \mu_H(x))(1 \pm \alpha\delta)]\!].$$

Proof. We define an auxiliary function $f : \mathbb{R} \to [0,1]$:

$$f(r) \stackrel{\text{def}}{=} \begin{cases} \frac{1}{2}\exp\left(\frac{\alpha}{2}r\right) & \text{if } r \le 0, \\ 1 - \frac{1}{2}\exp\left(-\frac{\alpha}{2}r\right) & \text{if } r > 0. \end{cases}$$

It is easy to verify that for all $r \in \mathbb{R}$, $\mu_H(r) = f(r - 1/2)$ and $1 - \mu_H(r) = f(-(r - 1/2))$. Therefore, in order to prove the lemma, it suffices to prove that for any r and r' such that $|r - r'| \le \delta$,

$$f(r)(1 - \alpha\delta) \le f(r') \le f(r)(1 + \alpha\delta), \tag{2.1}$$

i.e., a small additive change to the argument of f has a limited multiplicative impact on the value of f.

Note that f is differentiable in both $(-\infty,0)$ and $(0,\infty)$. Additionally, it is continuous in the entire range—the left and right branch of the function meet at 0—and both the left and right derivatives at 0 are equal. This implies that it is differentiable at 0 as well. Its derivative is

$$f'(r) = \begin{cases} \frac{\alpha}{4} \cdot \exp\left(\frac{\alpha}{2} r\right) & \text{if } r \le 0, \\ \frac{\alpha}{4} \cdot \exp\left(-\frac{\alpha}{2} r\right) & \text{if } r > 0, \end{cases}$$

which is positive for all r, and therefore, f is strictly increasing. Note that f' is increasing in $(-\infty,0]$ and decreasing in $[0,\infty)$. Hence the global maximum of f' equals $f'(0) = \alpha/4$.

In order to prove Inequality 2.1 for all r and r' such that $|r-r'| \le \delta$, we consider two cases. Suppose first that $r \ge 0$. By the upper bound on the derivative of f,

$$f(r) - \frac{\alpha}{4} \cdot |r-r'| \le f(r') \le f(r) + \frac{\alpha}{4} \cdot |r-r'|.$$

Since $r \ge 0$, $f(r) \ge 1/2$. This leads to

$$f(r) - f(r) \cdot \frac{\alpha}{2} \cdot |r-r'| \le f(r') \le f(r) + f(r) \cdot \frac{\alpha}{2} \cdot |r-r'|.$$

By the bound on $|r-r'|$,

$$f(r)(1-\alpha\delta) \le f(r') \le f(r)(1+\alpha\delta),$$

which finishes the proof in the first case.

Suppose now that $r < 0$. Since f is increasing, it suffices to bound the value of f from below at $r-\delta$ and from above and at $r+\delta$. For $r-\delta$, we obtain

$$\begin{aligned} f(r-\delta) &= \frac{1}{2} \exp\left(\frac{\alpha}{2}(r-\delta)\right) = f(r) \exp\left(-\frac{\alpha}{2}\delta\right) \\ &\ge f(r)\left(1 - \frac{\alpha}{2}\delta\right) \ge f(r)(1-\alpha\delta). \end{aligned}$$

For $r+\delta$, let us first define a function $g : \mathbb{R} \to \mathbb{R}$ as

$$g(y) \stackrel{\text{def}}{=} \frac{1}{2} \exp\left(\frac{\alpha}{2} y\right).$$

For $y \le 0$, $f(y) = g(y)$. For $y > 0$, $g'(y) \ge f'(y)$ and hence, for any $y \in \mathbb{R}$, $g(y) \ge f(y)$. As a result, we obtain

$$f(r+\delta) \le g(r+\delta) = \frac{1}{2} \exp\left(\frac{\alpha}{2}(r+\delta)\right) = f(r) \cdot \exp\left(\frac{\alpha}{2}\delta\right).$$

By the bound on δ in the lemma statement, $\frac{\alpha}{2}\delta \le 1$. It follows from the convexity of the exponential function that for any $y \in [0,1]$, $\exp(y) \le y \cdot \exp(1) + (1-y) \cdot \exp(0) \le 3y + (1-y) = 1+2y$. Continuing the reasoning,

$$f(r+\delta) \le f(r) \cdot \left(1 + 2 \cdot \frac{\alpha}{2}\delta\right) = f(r)(1+\alpha\delta),$$

which finishes the proof of Inequality (2.1). □

The main result of this section is Lemma 2.10 that states that if a distribution $\mathscr{D}$ of vertex assignments is near uniform, then `EmulatePhase` constructs a configuration $\mathscr{C}$ such that $\mathscr{D}[\mathscr{C}]$ is near uniform as well, and also, the maximum degree in the graph induced by the vertices not removed by `EmulatePhase` is bounded.

Lemma 2.10. *Let Δ, $G_\star = (V_\star, E_\star)$, m, and $\mathscr{D}$ be parameters for* `EmulatePhase` *such that*

- *$\mathscr{D}$ is an independent and ϵ-near uniform distribution on assignments of vertices $V_\star$ to $[m]$ for $\epsilon \in [0, (200 \ln n)^{-1}]$,*
- *$\frac{\Delta}{m} \geq 4000 \mu_R^{-2} \ln^2 n$.*

Let $\mathscr{C}$ be an m-configuration constructed by `EmulatePhase`*. With probability at least $1 - n^{-4}$ both the following properties hold:*

- *The maximum degree in the graph induced by the vertices not fixed in $\mathscr{C}$ is bounded by $\frac{3}{4}\Delta$.*
- *$\mathscr{D}[\mathscr{C}]$ is $60\alpha\left(\left(\frac{\Delta}{m}\right)^{-1/4} + \epsilon\right)$-near uniform.*

Proof overview (of Lemma 2.10). This is the most intricate proof of the entire chapter. We therefore provide a short overview. First, we list again the variables in `EmulatePhase` and `LocalPhase` to which we refer in the proof and define additional convenient symbols. Then we introduce five simple random events (Events 1–5) that capture properties needed to prove Lemma 2.10. In Claim 2.13, we show that the probability of all these events occurring simultaneously is high. The proof of the claim follows mostly from a repetitive application of the Chernoff bound. In the next claim, Claim 2.14, we show that the occurrence of all the events has a few helpful consequences. First, high degree vertices get removed in the execution of `EmulatePhase` (which is one of our final desired properties). Second, each vertex v that is not fixed in $\mathscr{C}$ has a very similar number of neighbors in all sets R_i and it has a very similar number of neighbors in all sets H_i. In the final proof of Lemma 2.10, we use the fact that this implies that to whichever set V_i vertex v was assigned in `EmulatePhase`, the probability of its removal in `EmulatePhase` was more or less the same. This leads to the conclusion that if v was distributed nearly uniformly in $\mathscr{D}$, it is distributed only slightly less uniformly in $\mathscr{D}[\mathscr{C}]$.

Notation. To simplify the presentation, for the rest of Section 2.5.4, we assume that Δ, $G_\star = (V_\star, E_\star)$, m, and $\mathscr{D}$ are the parameters to `EmulatePhase` as in the statement of Lemma 2.10. Additionally, for each $i \in [m]$, R_i, H_i, and F_i are the sets constructed by `LocalPhase` for the i-th subgraph in the execution of `EmulatePhase`. We also write $\mathscr{C}$ to denote the corresponding m-configuration, i.e., $\mathscr{C} = \left(\{R_i\}_{i\in[m]}, \{H_i\}_{i\in[m]}, \{F_i\}_{i\in[m]}\right)$. Furthermore, for each $v \in V_\star$, $\widehat{d}_v$ is the estimate of v's degree in the subgraph to which it was assigned. This estimate is computed in Line 2 of `LocalPhase`. We also use $\Delta_\star$ to denote the rescaled threshold passed in all calls to `LocalPhase`, i.e., $\Delta_\star = \frac{\Delta}{m}$.

We also introduce additional notation, not present in `EmulatePhase` or `LocalPhase`. For each $v \in V_\star$, $d_v \stackrel{\text{def}}{=} |N(v) \cap V_\star|$, i.e., d_v is the degree of v in $G_\star$. For each vertex $v \in V_\star$, we also introduce a notion of its *weight*: $w_v \stackrel{\text{def}}{=} \mu_H(d_v/\Delta)$, which can be seen as a very rough approximation of v's probability of being selected for the set of heavy vertices. For any $v \in V_\star$ and $U \subseteq V_\star$, we also introduce notation for the total weight of v's neighbors in U:

$$W_v(U) \stackrel{\text{def}}{=} \sum_{u \in N(v) \cap U} w_u.$$

Finally, for all $i \in [m]$ and $v \in V_\star$, we also introduce a slightly less intuitive notion of the expected number of heavy neighbors of v in the i-th subgraph after the degree estimates are fixed in Line 2 of `LocalPhase` and before vertices are assigned to the heavy set in Line 3:

$$h_{v,i} \stackrel{\text{def}}{=} \sum_{u \in N(v) \cap V_i} \mu_H\left(\widehat{d}_u / \Delta_\star\right).$$

Obviously, each $h_{v,i}$ is a random variable.

Convenient random events. We now list five random events that we hope all to occur simultaneously with high probability. The first event intuitively is the event that high-degree vertices are likely to be included in the set of heavy vertices in Line 3 of `LocalPhase`.

Event 1

For each vertex $v \in V_\star$ such that $d_v \geq \frac{3}{4}\Delta$,

$$\mu_H\left(\widehat{d}_v / \Delta_\star\right) \geq 1 - \frac{1}{2} n^{-6}.$$

Another way to define this event would be to state that $\widehat{d}_v$ for such vertices v is high, but this form is more suitable for our applications later. The next event is the event that all such vertices are in fact classified as heavy.

Event 2

Each vertex $v \in V_\star$ such that $d_v \geq \frac{3}{4}\Delta$ belongs to $\bigcup_{i \in [m]} H_i$.

The next event is the event that low-degree vertices have a number of neighbors in each set R_i close to the mean. This implies that if we were able to move a low-degree vertex v to V_i, for any $i \in [m]$, its estimated degree $\widehat{d}_v$ would not change significantly.

Event 3

For each vertex $v \in V_\star$ such that $d_v < \frac{3}{4}\Delta$ and each $i \in [m]$,

$$\left| \frac{1}{\mu_R} |N(v) \cap R_i| - \frac{d_v}{m} \right| \leq \Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star.$$

As a reminder, we use $W_v(U)$ to denote the expected number of vertices in $N(v) \cap U$ that are selected as heavy, where every vertex u is selected with respect to its global degree d_u. The next event shows that $W_v(V_i)$ does not deviate much from its mean.

Event 4

For each vertex $v \in V_\star$ such that $d_v < \frac{3}{4}\Delta$ and each $i \in [m]$,

$$|W_v(V_i) - W_v(V_\star)/m| \leq \Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star.$$

Recall that $h_{v,i}$ intuitively expresses the expected number of v's neighbors in the i-th induced subgraph at some specific stage in the execution of `LocalPhase` for the i-th induced subgraph. The final event is the event that for all bounded $h_{v,i}$, the actual number of v's neighbors in H_i does not deviate significantly from $h_{v,i}$.

Event 5

For each vertex $v \in V_\star$ and each $i \in [m]$, if $h_{v,i} \le 2\Delta_\star$, then
$$\left||N(v) \cap H_i| - h_{v,i}\right| \le \Delta_\star^{3/4}.$$

High probability of the random events. We now show that the probability of all the events occurring is high. The proof follows mostly via elementary applications of the Chernoff bound.

Claim 2.13. *If $\epsilon \in [0, 1/100]$ and $\frac{\Delta}{m} \ge 4000\mu_R^{-2}\ln^2 n$, then Events 1–5 occur simultaneously with probability at least $1 - n^{-4}$.*

Proof. We consider all events in order and later show by the union bound that all of them hold simultaneously with high probability. In the proof of the lemma, we extensively use the fact that $\Delta_\star = \frac{\Delta}{m} \ge 4000\mu_R^{-2}\ln^2 n = 4 \cdot 10^{15} \cdot \ln^4 n$.

First, we consider Event 1 and Event 2, which we handle together. Consider a vertex v such that $d_v \ge \frac{3}{4}\Delta$. Let $i_\star$ be the index of the set to which it is assigned. Since $\mathscr{D}$ is ϵ-near uniform, the expectation of $|N(v) \cap R_{i_\star}|$, the number of v's neighbors in $R_{i_\star}$, is at least $(1-\epsilon)\frac{3}{4}\mu_R\frac{\Delta}{m} \ge \frac{297}{400}\mu_R\Delta_\star$. Since vertices are both assigned to machines independently and included in the reference set independently as well, we can apply Lemma 2.7 to bound the deviation with high probability. The probability that the number of neighbors is smaller than $\frac{9}{10} \cdot \frac{297}{400}\mu_R\Delta_\star \ge \frac{5}{8}\mu_R\Delta_\star$ is at most
$$2\exp\left(-\frac{1}{3}\cdot\left(\frac{1}{10}\right)^2 \cdot \frac{297}{400}\mu_R\Delta_\star\right) \le 2\exp\left(-\frac{1}{405}\mu_R\Delta_\star\right) \le 2n^{-9} \le \frac{1}{2}n^{-6}.$$
Hence with probability at least $1 - \frac{1}{2}n^{-6}$, $\widehat{d}_v \ge \frac{5}{8}\Delta_\star$ and $\mu_H\left(\widehat{d}_v/\Delta_\star\right) \ge 1 - \frac{1}{2}n^{-6}$. If this is the case, v is not included in the set of heavy vertices in Line 3 of `LocalPhase` with probability at most $\frac{1}{2}n^{-6}$. Therefore, v has the desired value of $\mu_H\left(\widehat{d}_v/\Delta_\star\right)$ and belongs to $H_{i_\star}$ with probability at least $1 - n^{-6}$. By the union bound, this occurs for all high degree vertices with probability at least $1 - n^{-5}$, in which case both Event 1 and Event 2 occur.

We now show that Event 3 occurs with high probability. Let v be an arbitrary vertex such that $d_v < \frac{3}{4}\Delta$ and let $i \in [m]$. Let $X_{v,i} \stackrel{\text{def}}{=} |N(v) \cap R_i|$. $X_{v,i}$ is a random variable. Since $\mathscr{D}$ is ϵ-near uniform, $\mathbb{E}\left[X_{v,i}\right] \in [\![(1\pm\epsilon)\mu_R d_v/m]\!]$. In particular, due to the bounds on d_v and ϵ, $E[X_{v,i}] \le \mu_R\Delta_\star$. Due to the independence, we can use Lemma 2.7 to bound the deviation of $X_{v,i}$ from its expectation. We have
$$\begin{aligned}\Pr\left(|X_{v,i} - \mathbb{E}[X_{v,i}]| > \mu_R\Delta_\star^{3/4}\right) &\le 2\exp\left(-\frac{1}{3}\cdot\left(\frac{1}{\Delta_\star^{1/4}}\right)^2 \cdot \mu_R\Delta_\star\right)\\ &= 2\exp\left(-\frac{1}{3}\mu_R\Delta_\star^{1/2}\right) \le 2n^{-21}.\end{aligned}$$
Hence with probability $1 - 2n^{-21}$, we have
$$\begin{aligned}\left|X_{v,i} - \mu_R\frac{d_v}{m}\right| &\le \left|X_{v,i} - \mathbb{E}[X_{v,i}]\right| + \left|\mathbb{E}[X_{v,i}] - \mu_R\frac{d_v}{m}\right| \le \mu_R\Delta_\star^{3/4} + \epsilon\mu_R\frac{d_v}{m}\\ &\le \mu_R\Delta_\star^{3/4} + \frac{3}{4}\epsilon\mu_R\Delta_\star.\end{aligned}$$

By dividing both sides by μ_R, we obtain the desired bound

$$\left|\frac{X_{v,i}}{\mu_R} - \frac{d_v}{m}\right| = \left|\frac{1}{\mu_R}|N(v)\cap R_i| - \frac{d_v}{m}\right| \le \Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star.$$

By the union bound, this holds for all v and i of interest—and therefore, Event 3 occurs—with probability at least $1-|V_\star|\cdot m\cdot 2n^{-21} \ge 1-n^{-5}$.

We now move on to Event 4. Consider a vertex v such that $d_v < \frac{3}{4}\Delta$ and $i\in[m]$. Note that since the weight of every vertex is at most 1, $W_v(V_\star)/m \le d_v/m < \frac{3}{4}\Delta_\star$. Since $\mathscr{D}[\mathscr{C}]$ is ϵ-near uniform, $\mathbb{E}[W_v(V_i)] \in [\![(1\pm\epsilon)W_v(V_\star)/m]\!]$. In particular, $\mathbb{E}[W_v(V_i)] \le \frac{101}{100}W_v(V_\star)/m \le \frac{101}{100}\cdot\frac{3}{4}\Delta_\star \le \Delta_\star$. Since vertices are assigned to machines independently, we can apply Lemma 2.7 to bound the deviation of $W_v(V_i)$ from the expectation:

$$\begin{aligned}\Pr\left(|W_v(V_i) - \mathbb{E}[W_v(V_i)]| > \Delta_\star^{3/4}\right) &\le 2\exp\left(-\frac{1}{3}\cdot\left(\frac{1}{\Delta_\star^{1/4}}\right)^2\cdot\Delta_\star\right)\\ &= 2\exp\left(-\frac{1}{3}\Delta_\star^{1/2}\right) \le 2n^{-21}.\end{aligned}$$

As a result, with probability at least $1-2n^{-21}$,

$$\begin{aligned}|W_v(V_i) - W_v(V_\star)/m| &\le |W_v(V_i) - \mathbb{E}[W_v(V_i)]| + |\mathbb{E}[W_v(V_i)] - W_v(V_\star)/m|\\ &\le \Delta_\star^{3/4} + \epsilon W_v(V_\star)/m \le \Delta_\star^{3/4} + \epsilon d_v/m \le \Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star.\end{aligned}$$

By the union bound, this holds for all v and i of interest—and therefore, Event 4 occurs—with probability at least $1-|V_\star|\cdot m\cdot 2n^{-21} \ge 1-n^{-5}$.

To show that Event 5 occurs with high probability, recall first that $h_{v,i}$ is the expected number of v's neighbors to be added in Line 3 to H_i in the execution of `LocalPhase` for the i-th subgraph. Note that the decision of adding a vertex to H_i is made independently for each neighbor of v. Fix a $v\in V_\star$ and $i\in[m]$ such that $h_{v,i}\le 2\Delta_\star$. We apply Lemma 2.7 to bound the probability of a large deviation from the expectation:

$$\begin{aligned}\Pr\left(\left||N(v)\cap H_i| - h_{v,i}\right| > \Delta_\star^{3/4}\right) &\le 2\exp\left(-\frac{1}{3}\cdot\left(\frac{1}{2\Delta_\star^{1/4}}\right)^2\cdot 2\Delta_\star\right)\\ &= 2\exp\left(-\frac{1}{6}\Delta_\star^{1/2}\right) \le 2n^{-10}.\end{aligned}$$

By the union bound the probability that this bound does not hold for some v and i such that $h_{v,i}\le 2\Delta_\star$ is by the union bound at most $|V_\star|\cdot m\cdot 2n^{-10} \le n^{-5}$. Hence, Event 5 occurs with probability at least $1-n^{-5}$.

In summary, Events 1–5 occur simultaneously with probability at least $1-4\cdot n^{-5} \ge 1-n^{-4}$ by another application of the union bound. □

Consequences of the random events. We now show that if all the random events occur, then a few helpful properties hold for every vertex v that is not fixed by the constructed configuration $\mathscr{C}$. Namely, v's degree is at most $\frac{3}{4}\Delta$, the number of v's neighbors is similar in all sets R_i is approximately the same, and the number of v's neighbors is similar in all sets H_i.

Claim 2.14. *If Events 1–5 occur for* $\epsilon \in [0, (200 \ln n)^{-1}]$ *and* $\frac{\Delta}{m} \geq 4000\mu_R^{-2} \ln^2 n$*, then the following properties hold for every vertex* $v \in V_\star$ *that is not fixed by* $\mathscr{C}$*:*

1. $d_v < \frac{3}{4}\Delta$.
2. *There exists* χ_v *such that for all* $i \in [m]$,
$$|N(v) \cap R_i| / \mu_R \in \left[\!\!\left[\chi_v \pm \left(\Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star \right) \right]\!\!\right].$$
3. *There exists* $\psi_v \in \left[0, \frac{3}{4}\Delta_\star\right]$ *such that for all* $i \in [m]$,
$$|N(v) \cap H_i| \in \left[\!\!\left[\psi_v \pm \alpha \left(\Delta_\star^{3/4} + \epsilon\Delta_\star \right) \right]\!\!\right].$$

Proof. We use in the proof of the claim the fact that $\Delta_\star = \frac{\Delta}{m} \geq 4000\mu_R^{-2} \ln^2 n = 4 \cdot 10^{15} \cdot \ln^4 n$. To prove the lemma, we fix a vertex v that is not fixed by $\mathscr{C}$. The first property is directly implied by Event 2. Suppose that $d_v \geq \frac{3}{4}\Delta$. Then v is included in the H_i corresponding to the subgraph to which it has been assigned and v is fixed by $\mathscr{C}$. We obtain a contradiction that implies that $d_v < \frac{3}{4}\Delta$.

For the second property, we now know that $d_v < \frac{3}{4}\Delta$. The property follows then directly from Event 3 with $\chi_v \stackrel{\text{def}}{=} d_v / m$.

The last property requires a more complicated reasoning. We set $\psi_v \stackrel{\text{def}}{=} W_v(V_\star)/m < \frac{3}{4}\Delta_\star$. Consider any $i \in [m]$. By Event 4,

$$W_v(V_i) \in \left[\!\!\left[\psi_v \pm \left(\Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star \right) \right]\!\!\right]. \tag{2.2}$$

Consider now an arbitrary $u \in V_\star$. We bound the difference between $w_u = \mu_H(d_u/\Delta)$, which can be seen as the ideal probability of the inclusion in the set of heavy vertices, and $\mu_H\left(\widehat{d}_u/\Delta_\star\right)$, the actual probability of this event in Line 3 of the appropriate execution of `LocalPhase`. Let $\delta_\star \stackrel{\text{def}}{=} \alpha\left(\Delta_\star^{-1/4} + \frac{3}{4}\epsilon\right)$. We consider two cases.

- If $d_u < \frac{3}{4}\Delta$, by Event 3, the monotonicity of μ_H, and Lemma 2.12,
$$\begin{aligned}\mu_H\left(\widehat{d}_u/\Delta_\star\right) &\in \left[\!\!\left[\mu_H\left(\frac{d_u}{\Delta} \pm \left(\Delta_\star^{-1/4} + \frac{3}{4}\epsilon \right) \right) \right]\!\!\right] \\ &\subseteq \left[\!\!\left[w_u \cdot (1 \pm \delta_\star) \right]\!\!\right].\end{aligned}$$
Note that Lemma 2.12 is applied properly because $\Delta_\star^{-1/4} + \frac{3}{4}\epsilon \leq (200 \ln n)^{-1} + (200 \ln n)^{-1} \leq (48 \ln n)^{-1}$.
- If $d_u \geq \frac{3}{4}\Delta$, by Event 1, $\mu_H\left(\widehat{d}_u/\Delta_\star\right) \in \left[1 - \frac{1}{2}n^{-6}, 1\right]$. Concurrently, $w_u \in \left[\mu_H(3/4), 1\right] = \left[1 - \frac{1}{2}n^{-12}, 1\right]$. Because $\Delta_\star$ is relatively small, i.e., $\Delta_\star \leq n$,
$$\mu_H\left(\widehat{d}_u/\Delta_\star\right) \in \left[\!\!\left[w_u \left(1 \pm \Delta_\star^{-1/4}\right) \right]\!\!\right] \subseteq \left[\!\!\left[w_u \cdot (1 \pm \delta_\star) \right]\!\!\right],$$
which is the same bound as in the previous case.

It follows from the bound that we just obtained and the definitions of W_v and $h_{v,i}$ that

$$\begin{aligned} h_{v,i} &= \sum_{u \in N(v) \cap V_i} \mu_H\left(\widehat{d}_u / \Delta_\star\right) \in \left[\!\!\left[(1 \pm \delta_\star) \cdot \sum_{u \in N(v) \cap V_i} w_u \right]\!\!\right] \\ &= \left[\!\left[W_v\left(V_i\right) \cdot (1 \pm \delta_\star) \right]\!\right]. \end{aligned} \tag{2.3}$$

We now combine bounds (2.2) and (2.3):

$$\begin{aligned} h_{v,i} &\in \left[\psi_v (1-\delta_\star) - \left(\Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star\right)(1+\delta_\star), \psi_v (1+\delta_\star) + \left(\Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star\right)(1+\delta_\star) \right] \\ &\subseteq \left[\!\!\left[\psi_v \pm \left(\psi_v \delta_\star + \left(\Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star\right)(1+\delta_\star) \right) \right]\!\!\right]. \end{aligned}$$

Due to the lower bound on $\Delta_\star$, we obtain $\delta_\star \le \alpha\left((200\ln n)^{-1} + (200\ln n)^{-1}\right) \le 1$. This enables us to simplify and further transform the bound on $h_{v,i}$:

$$\begin{aligned} h_{v,i} &\in \left[\!\!\left[\psi_v \pm \left(\psi_v \delta_\star + 2\left(\Delta_\star^{3/4} + \frac{3}{4}\epsilon\Delta_\star\right) \right) \right]\!\!\right] \\ &\subseteq \left[\!\!\left[\psi_v \pm \left(\frac{3}{4}\alpha\Delta_\star^{3/4} + \frac{9}{16}\alpha\epsilon\Delta_\star + 2\Delta_\star^{3/4} + \frac{3}{2}\epsilon\Delta_\star \right) \right]\!\!\right] \\ &\subseteq \left[\!\!\left[\psi_v \pm \alpha\left(\frac{4}{5}\Delta_\star^{3/4} + \epsilon\Delta_\star \right) \right]\!\!\right]. \end{aligned}$$

By applying the bound on $\Delta_\star$ again, we obtain a bound on the magnitude of the second term in the above bound:

$$\alpha\left(\frac{4}{5}\Delta_\star^{3/4} + \epsilon\Delta_\star\right) = \alpha\left(\frac{4}{5}\Delta_\star^{-1/4} + \epsilon\right)\Delta_\star \le 96\ln n\left(\frac{1}{200\ln n} + \frac{1}{200\ln n}\right)\Delta_\star \le \Delta_\star.$$

This implies that $h_{v,i} \le \psi_v + \Delta_\star \le 2\Delta_\star$. The condition in Event 5 holds, and therefore, $\left||N(v) \cap H_i| - h_{v,i}\right| \le \Delta_\star^{3/4}$. We combine this with the bound on $h_{v,i}$ to obtain

$$|N(v) \cap H_i| \in \left[\!\!\left[\psi_v \pm \left(\alpha\frac{4}{5}\Delta_\star^{3/4} + \alpha\epsilon\Delta_\star + \Delta_\star^{3/4} \right) \right]\!\!\right] \subseteq \left[\!\left[\psi_v \pm \alpha\left(\Delta_\star^{3/4} + \epsilon\Delta_\star\right) \right]\!\right].$$

□

Wrapping up the proof of near uniformity. We now finally prove Lemma 2.10. Recall that it states that an ϵ-near uniform $\mathscr{D}$ is very likely to result in a near uniform $\mathscr{D}[\mathscr{C}]$ with a slightly worse parameter and that all vertices not fixed by $\mathscr{C}$ have bounded degree. The proof combines the last two claims: Claim 2.13 and Claim 2.14. We learn that $\mathscr{C}$, the m-configuration constructed in the process is very likely to have the properties listed in Claim 2.14. One of those properties is exactly the property that all vertices not fixed by $\mathscr{C}$ have bounded degree. Hence we have to prove only the near uniformity property. We accomplish this by observing that the probability of $\mathscr{C}$ equal to a specific m-configuration $\mathscr{C}_\star$ with good properties—those in Claim 2.14—does not depend significantly on to which induced subgraph a given vertex v not fixed in $\mathscr{C}_\star$ is assigned. This can be used to show that the conditional distribution of v given that $\mathscr{C} = \mathscr{C}_\star$ is near uniform as desired.

Proof of Lemma 2.10. By combining Claim 2.13 and Claim 2.14, we learn that with probability at least $1-n^{-4}$, all properties listed in the statement of Claim 2.14 hold for $\mathscr{C}$, the configuration constructed by `EmulatePhase`. Since one of the properties is exactly the same as in the statement of Lemma 2.10, it suffices to prove the other one: that $\mathscr{D}[\mathscr{C}]$ is $60\alpha\left(\Delta_\star^{-1/4}+\epsilon\right)$-near uniform for $\mathscr{C}$ with this set of properties.

Fix $\widetilde{\mathscr{C}} = \left(\{\widetilde{R}_i\}_{i\in[m]}, \{\widetilde{H}_i\}_{i\in[m]}, \{\widetilde{F}_i\}_{i\in[m]}\right)$ to be an m-configuration that has non-zero probability when `EmulatePhase` is run for $\mathscr{D}$ and has the properties specified by Claim 2.14. Consider an arbitrary vertex $v \in V_\star$. In order to prove the near uniformity of $\mathscr{D}\left[\widetilde{\mathscr{C}}\right]$, we show that v is assigned by it almost uniformly to $[m]$. Let $\mathscr{E}$ be the event that `EmulatePhase` constructs $\widetilde{\mathscr{C}}$, i.e., $\mathscr{C} = \widetilde{\mathscr{C}}$. For each $i \in [m]$, let $\mathscr{E}_{\to i}$ be the event that v is assigned to the i-th induced subgraph. Let $p : [m] \to [0,1]$ be the probability mass function describing the probability of the assignment of v to each of the m subgraphs in $\mathscr{D}$. Obviously, $p(i) = \Pr[\mathscr{E}_{\to i}]$ for all $i \in [m]$. Due to the ϵ-near uniformity of $\mathscr{D}$, $p(i) = \left[\!\left[\frac{1}{m}(1\pm\epsilon)\right]\!\right]$.

For each $i \in [m]$, let $q_i \stackrel{\text{def}}{=} \Pr[\mathscr{E}|\mathscr{E}_{\to i}]$. In order to express all q_i's in a suitable form, we conduct a thought experiment. Suppose v were not present in the graph, but the distribution of all the other vertices in the modified $\mathscr{D}$ remained the same. Let $q_\star$ be the probability of $\mathscr{E}$, i.e., $\mathscr{C} = \widetilde{\mathscr{C}}$, in this modified scenario. How does the probability of $\mathscr{E}$ change if we add v back and condition on its assignment to a machine i? Note first that conditioning on $\mathscr{E}_{\to i}$ does not impact the distribution of the other vertices, because vertices are assigned to machines independently in $\mathscr{D}$. In order for $\mathscr{E}$ still to occur in this scenario, v cannot be assigned to any of $\widetilde{R}_i$, $\widetilde{H}_i$, or $\widetilde{F}_i$, for which it is considered. Additionally, as long as this the case, v does not impact the behavior of other vertices, which only depends on the content of these sets and independent randomized decisions to include vertices. As a result we can express q_i as a product of $q_\star$ and three probabilities: of v not being included in sets $\widetilde{R}_i$, $\widetilde{H}_i$, or $\widetilde{F}_i$.

$$q_i = q_\star \cdot \left(1-\mu_R\right) \cdot \left(1-\mu_H\left(\frac{\left|N(v)\cap\widetilde{R}_i\right|/\mu_R}{\Delta_\star}\right)\right) \cdot \left(1-\mu_F\left(\frac{\left|N(v)\cap\widetilde{H}_i\right|}{\Delta_\star}\right)\right). \tag{2.4}$$

Using the properties listed in Claim 2.14, we have

$$\left|N(v)\cap\widetilde{R}_i\right|/\mu_R \in \left[\!\left[\chi_v \pm \left(\Delta_\star^{3/4}+\frac{3}{4}\epsilon\Delta_\star\right)\right]\!\right],$$

and

$$\left|N(v)\cap\widetilde{H}_i\right| \in \left[\!\left[\psi_v \pm \alpha\left(\Delta_\star^{3/4}+\epsilon\Delta_\star\right)\right]\!\right],$$

where χ_v and ψ_v are constants independent of machine i to which v has been assigned and $\psi \le \frac{3}{4}\Delta_\star$. In the next step, we use these bounds to derive bounds on the multiplicative terms in Equation (2.4) that may depend on i. We also repeatedly use the bounds $\Delta_\star = \frac{\Delta}{m} \ge 4000\mu_R^{-2}\ln^2 n = 4\cdot 10^{15}\cdot\ln^4 n$ and $\epsilon \le (200\ln n)^{-1}$ from the lemma statement. First, due to Lemma 2.12,

$$\begin{aligned}
1-\mu_H\left(\frac{\left|N(v)\cap\widetilde{R}_i\right|/\mu_R}{\Delta_\star}\right) &\in \left[\!\left[1-\mu_H\left(\frac{\chi_v}{\Delta_\star}\pm\left(\Delta_\star^{-1/4}+\frac{3}{4}\epsilon\right)\right)\right]\!\right] \\
&\subseteq \left[\!\left[\left(1-\mu_H\left(\frac{\chi_v}{\Delta_\star}\right)\right)\cdot\left(1\pm\alpha\left(\Delta_\star^{-1/4}+\frac{3}{4}\epsilon\right)\right)\right]\!\right].
\end{aligned}$$

(Note that the application of Lemma 2.12 was correct, because $\Delta_\star^{-1/4} + \frac{3}{4}\epsilon \le (200\ln n)^{-1} + (200\ln n)^{-1} < (96\ln n)^{-1}$.) Second,

$$1-\mu_F\left(\frac{\left|N(v)\cap\widetilde{H}_i\right|}{\Delta_\star}\right)\in\left[\!\!\left[1-\mu_F\left(\frac{\psi_v}{\Delta_\star}\pm\alpha\left(\Delta_\star^{-1/4}+\epsilon\right)\right)\right]\!\!\right].$$

Since $\psi_v/\Delta_\star \le \frac{3}{4}$ and $\alpha\left(\Delta_\star^{-1/4}+\epsilon\right) \le (96\ln n)\cdot\left((200\ln n)^{-1}+(200\ln n)^{-1}\right) < 1$, the argument to μ_F in the above bound is always less than 4, and therefore, only one branch of μ_F's definitions gets applied. Hence, we can eliminate μ_F:

$$1-\mu_F\left(\frac{\left|N(v)\cap\widetilde{H}_i\right|}{\Delta_\star}\right)\in\left[\!\!\left[1-\frac{\psi_v}{4\Delta_\star}\pm\frac{\alpha}{4}\left(\Delta_\star^{-1/4}+\epsilon\right)\right]\!\!\right].$$

Since $1-\frac{\psi_v}{4\Delta_\star}\ge\frac{3}{4}$, we can further transform the bound to

$$1-\mu_F\left(\frac{\left|N(v)\cap\widetilde{H}_i\right|}{\Delta_\star}\right)\in\left[\!\!\left[\left(1-\frac{\psi_v}{4\Delta_\star}\right)\left(1\pm\frac{\alpha}{3}\left(\Delta_\star^{-1/4}+\epsilon\right)\right)\right]\!\!\right].$$

Let $\delta_1 \stackrel{\text{def}}{=} \alpha\left(\Delta_\star^{-1/4}+\frac{3}{4}\epsilon\right)$ and $\delta_2 \stackrel{\text{def}}{=} \frac{\alpha}{3}\left(\Delta_\star^{-1/4}+\epsilon\right)$. As a result, every q_i can be expressed as $q_i = \eta_v\lambda_i\lambda_i'$, where η_v is a constant independent of i, $\lambda_i\in[\![1\pm\delta_1]\!]$, and $\lambda_i'\in[\![1\pm\delta_2]\!]$. For every i, we can also write

$$\Pr[\mathscr{E}\wedge\mathscr{E}_{\to i}] = \Pr[\mathscr{E}|\mathscr{E}_{\to i}]\cdot\Pr[\mathscr{E}_{\to i}] = \eta_v\lambda_i\lambda_i'\cdot p(i) = \frac{\eta_v}{m}\lambda_i\lambda_i'\lambda_i'',$$

where $\lambda_i''\in[\![1\pm\epsilon]\!]$. We now express the conditional probability of v being assigned to the i-th subgraph in $\mathscr{D}$ given $\mathscr{E}$:

$$\Pr[\mathscr{E}_{\to i}|\mathscr{E}] = \frac{\Pr[\mathscr{E}\wedge\mathscr{E}_{\to i}]}{\sum_{j=1}^m\Pr[\mathscr{E}\wedge\mathscr{E}_{\to j}]} = \frac{\lambda_i\lambda_i'\lambda_i''}{\sum_{j=1}^m\lambda_j\lambda_j'\lambda_j''}.$$

Note that for any i, this implies that

$$\frac{1}{m}\cdot\frac{(1-\delta_1)(1-\delta_2)(1-\epsilon)}{(1+\delta_1)(1+\delta_2)(1+\epsilon)} \le \Pr[\mathscr{E}_{\to i}|\mathscr{E}] \le \frac{1}{m}\cdot\frac{(1+\delta_1)(1+\delta_2)(1+\epsilon)}{(1-\delta_1)(1-\delta_2)(1-\epsilon)}. \tag{2.5}$$

Observe that

$$\delta_1 \le (96\ln n)\cdot\left((7000\ln n)^{-1}+(250\ln n)^{-1}\right) < 1/2,$$

and

$$\delta_2 \le \frac{1}{3}\cdot(96\ln n)\cdot\left((7000\ln n)^{-1}+(200\ln n)^{-1}\right) < 1/2.$$

Hence all of δ_1, δ_2, and ϵ are at most $1/2$. We can therefore transform (2.5) to

$$\frac{1}{m}\cdot(1-\delta_1)^2(1-\delta_2)^2(1-\epsilon)^2 \le \Pr[\mathscr{E}_{\to i}|\mathscr{E}] \le \frac{1}{m}\cdot(1+\delta_1)(1+\delta_2)(1+\epsilon)(1+2\delta_1)(1+2\delta_2)(1+2\epsilon),$$

and then

$$\frac{1}{m}\cdot(1-2\delta_1-2\delta_2-2\epsilon) \le \Pr[\mathscr{E}_{\to i}|\mathscr{E}] \le \frac{1}{m}\cdot(1+45\delta_1+45\delta_2+45\epsilon).$$

Hence

$$\Pr[\mathscr{E}_{\to i}|\mathscr{E}]\in\left[\!\!\left[\frac{1}{m}\cdot(1\pm45(\delta_1+\delta_2+\epsilon))\right]\!\!\right] \subseteq \left[\!\!\left[\frac{1}{m}\cdot\left(1\pm60\alpha\left(\Delta_\star^{-1/4}+\epsilon\right)\right)\right]\!\!\right],$$

which finishes the proof that $\mathscr{D}\left[\widetilde{\mathscr{E}}\right]$ is $60\alpha\left(\Delta_\star^{-1/4}+\epsilon\right)$-near uniform. □

2.6 Parallel Algorithm

In this section, we introduce our main parallel algorithm. It builds on the ideas introduced in `EmulatePhase`. `EmulatePhase` randomly partitions the graph into m induced subgraphs and runs on each of them `LocalPhase`, which resembles a phase of `GlobalAlg`. As we have seen, the algorithm performs well even if vertices are assigned to subgraphs not exactly uniformly so long as the assignment is fully independent. Additionally, with high probability, if we condition on the configuration of sets R_i, H_i, and F_i that were removed, the distribution of assignments of the remaining vertices is still nearly uniform and also independent.

These properties allow for the main idea behind the final parallel algorithm. We partition vertices randomly into m induced subgraphs and then run `LocalPhase` multiple times on each of them with no repartitioning in the meantime. In each iteration, for a given subgraph, we halve the local threshold $\Delta_\star$. This corresponds to multiple phases of the original global algorithm. As long as we can show that this approach leads to finding a large matching, the obvious gain is that *multiple* phases of the original algorithm translate to $O(1)$ parallel rounds. This approach enables our main result: the parallel round complexity reduction from $O(\log n)$ to $O((\log\log n)^2)$.

Algorithm 5: `ParallelAlg`(G, S)
The final parallel matching algorithm

Input:
- graph $G = (V, E)$ on n vertices
- parameter $S \in \mathbb{Z}_+$ such that $S \leq n$ and $S = n^{\Omega(1)}$ (each machine uses $O(S)$ space)

Output: matching in G

1 $\Delta \leftarrow n$, $V' \leftarrow V$, $M \leftarrow \emptyset$
2 **while** $\Delta \geq \frac{n}{S}(200 \ln n)^{32}$ **do**
  `/* High-probability invariant: maximum degree in` $G[V']$ `bounded by` $\frac{3}{2}\Delta$ `*/`
3 $m \leftarrow \left\lfloor \sqrt{\frac{n\Delta}{S}} \right\rfloor$ `/* number of machines used */`
4 $\tau \leftarrow \left\lceil \frac{1}{16} \log_{120\alpha}(\Delta/m) \right\rceil$ `/* number of phases to emulate */`
5 Partition V' into m sets $V_1, \ldots, V_m$ by assigning each vertex independently uniformly at random.
6 **foreach** $i \in [m]$ **do in parallel**
7 If the number of edges in $G[V_i]$ is greater than $8S$, $V_i \leftarrow \emptyset$.
8 **for** $j \in [\tau]$ **do** $(V_i, M_{i,j}) \leftarrow$ `LocalPhase`$\left(i, G[V_i], \Delta/\left(2^{j-1}m\right)\right)$
9 $V' \leftarrow \bigcup_{i=1}^{m} V_i$
10 $M \leftarrow M \cup \bigcup_{i=1}^{m} \bigcup_{j=1}^{\tau} M_{i,j}$
11 $\Delta \leftarrow \Delta/2^{\tau}$
12 Compute degrees of vertices V' in $G[V']$ and remove from V' vertices of degree at least 2Δ.
13 Directly simulate $M' \leftarrow$ `GlobalAlg`$(G[V'], 2\Delta)$, using $O(1)$ rounds per phase.
14 **return** $M \cup M'$

We present `ParallelAlg`, our parallel algorithm, as Algorithm 5. We write S to denote a parameter specifying the amount of space per machine. After the initialization of variables, the algorithm enters the main loop in Lines 2–11. The loop is executed as long as Δ, an approximate upper bound on the maximum degree in the remaining graph, is large enough.

The loop implements the idea of running multiple iterations of `LocalPhase` on each induced subgraph in a random partition. At the beginning of the loop, the algorithm decides on m, the number of machines, and τ, the number of phases to be emulated. Then it creates a random partition of the current set of vertices that results in m induced subgraphs. Next for each subgraph, the algorithm first runs a security check that the set of edges fits onto a single machine (see Line 7). If it does not, which is highly unlikely, the entire subgraph is removed from the graph. Otherwise, the entire subgraph is sent to a single machine that runs τ consecutive iterations of `LocalPhase`. Then the vertices not removed in the executions of `LocalPhase` are collected for further computation and new matching edges are added to the matching being constructed. During the execution of the loop, the maximum degree in the graph induced by V', the set of vertices to be considered is bounded by $\frac{3}{2}\Delta$ with high probability. Once the loop finishes, we remove from the graph vertices of degree higher than 2Δ—there should be none—and we directly simulate `GlobalAlg` on the remaining graph, using $O(1)$ rounds per phase.

2.6.1 Properties of Thresholds

Before we analyze the behavior of the algorithm, we observe that the value of $\frac{\Delta}{m}$ inside the main loop is at least polylogarithmic and that the same property holds for the rescaled threshold that is passed to `LocalPhase`.

Lemma 2.15. *Consider a single iteration of the main loop of* `ParallelAlg` *(Lines 2–11). Let* Δ *and* m *be set as in this iteration. The following two properties hold:*

- $\Delta/m \geq (200\log n)^{16}$.
- *The threshold* $\Delta/\left(2^{j-1}m\right)$ *passed to* `LocalPhase` *in Line 8 is always at least* $(\Delta/m)^{15/16} \geq 4000\mu_R^{-2}\ln^2 n$.

Proof. Let τ be also as in this iteration of the loop. The smallest threshold passed to `LocalPhase` is $\Delta/(2^{\tau-1}m)$. Let $\lambda \stackrel{\text{def}}{=} S\Delta/n$, where S is the parameter to `ParallelAlg`. Due to the condition in Line 2, $\lambda \geq (200\ln n)^{32}$. Note that $\Delta = \lambda n/S$. Hence $m \leq \sqrt{n\Delta/S} = \frac{n}{S}\sqrt{\lambda}$. This implies that $\Delta/m \geq \sqrt{\lambda} \geq (200\ln n)^{16}$, which proves the first claim. Due to the definition of τ,

$$2^{\tau-1} \leq (120\alpha)^{\tau-1} \leq (\Delta/m)^{1/16}.$$

This implies that

$$\Delta/(2^{\tau-1}m) \geq (\Delta/m)^{15/16} \geq (200\ln n)^{15} > 4\cdot 10^{15}\cdot\ln^4 n = 4000\mu_R^{-2}\ln^2 n.$$

□

We also observe that the probability of any set of vertices deleted by the security check in Line 7 of `ParallelAlg` is low as long as the maximum degree in the graph induced by the remaining vertices is bounded.

Lemma 2.16. *Consider a single iteration of the main loop of* `ParallelAlg` *and let* Δ *and* V' *be as in that iteration. If the maximum degree in* $G[V']$ *is bounded by* $\frac{3}{2}\Delta$*, then the probability of any subset of vertices deleted in Line 7 is* n^{-8}.

Proof. Let m be as in the same iteration of the main loop of `ParallelAlg`. Consider a single vertex $v \in V'$. The expected number of v's neighbors assigned to the same subgraph is at most $\frac{3}{2}\Delta/m$. Recall that due to Lemma 2.15, $\frac{\Delta}{m} \geq 200 \ln n$. Since the assignment of vertices to machines is fully independent, by Lemma 2.7 (i.e., the Chernoff bound), the probability that v has more than $2\Delta/m$ neighbors is bounded by

$$2\exp\left(-\frac{1}{3}\cdot\left(\frac{1}{3}\right)^2\cdot\frac{3}{2}\cdot\frac{\Delta}{m}\right) \leq 2\exp\left(-\frac{1}{18}\cdot 200\ln n\right) \leq n^{-10}.$$

Therefore, by the union bound, with probability $1-n^{-9}$, no vertex has more than 2Δ neighbors in the same induced subgraph. As $|V'| \leq n$, the expected number of vertices in each set V_i constructed in the iteration of the main loop is at most $n/m \geq \Delta/m \geq 200\ln n$. What is the probability that $|V_i| > 2n/m$? By the independence of vertex assignments and Lemma 2.7, the probability of such event is at most

$$2\exp\left(-\frac{1}{3}\cdot\frac{n}{m}\right) \leq 2\exp\left(-\frac{1}{3}\cdot 200\ln n\right) \leq n^{-10}.$$

Again by the union bound, the event $|V_i| \leq 2n/m$, for all i simultaneously, occurs with probability at least $1-n^{-9}$. Combining both bounds, with probability at least $1-2n^{-9} \geq 1-n^{-8}$, all induced subgraphs have at most $2n/m$ vertices and the degree of every vertex is bounded by $2\Delta/m$. Hence the number of edges in one induced subgraph is at most $\frac{1}{2}\cdot\frac{2n}{m}\cdot\frac{2\Delta}{m} = \frac{2n\Delta}{m^2}$. By the definition of m and the setting of parameters in the algorithm, $m \geq \frac{1}{2}\sqrt{\frac{n\Delta}{S}}$, where S is the parameter to `ParallelAlg`. This implies that the number of edges is at most $2n\Delta/\left(\frac{1}{2}\sqrt{\frac{n\Delta}{S}}\right)^2 = 8S$ in every induced subgraph with probability $1-n^{-8}$, in which case no set V_i is deleted in Line 7 of `ParallelAlg`. □

2.6.2 Matching Size Analysis

The parallel algorithm runs multiple iterations of `LocalPhase` on each induced subgraph, without repartitioning. A single iteration on all subgraphs corresponds to running `EmulatePhase` once. We now show that in most cases, the global algorithm simulates `EmulatePhase` on a well behaved distribution with independently assigned vertices and all vertices distributed nearly uniformly conditioned on the configurations of the previously removed sets R_i, H_i, and F_i. We also show that the maximum degree in the remaining graph is likely to decrease gracefully during the process.

Lemma 2.17. *With probability at least* $1-n^{-3}$*:*

- *all parallel iterations of* `LocalPhase` *in* `ParallelAlg` *on each induced subgraph correspond to running* `EmulatePhase` *on independent and* $(200\ln n)^{-1}$*-near uniform distributions of assignments,*
- *the maximum degree of the graph induced by the remaining vertices after the* k*-th simulation of* `EmulatePhase` *is* $\frac{3}{2}\Delta/2^k$*.*

Proof. We first consider a single iteration of the main loop in `ParallelAlg`. Let Δ, τ, and m be set as in this iteration of the loop. For $j \in [\tau]$, let $\Delta_j \stackrel{\text{def}}{=} \Delta/\left(2^{j-1}m\right)$ be the threshold passed

to `LocalPhase` for the j-th iteration of `LocalPhase` on each of the induced subgraphs. The parallel algorithm assigns vertices to subgraphs and then iteratively runs `LocalPhase` on each of them. In this analysis we ignore the exact assignment of vertices to subgraphs until they get removed as a member of one of sets R_i, H_i, or F_i. Instead we look at the conditional distribution on assignments given the configurations of sets R_i, H_i, and F_i removed in the previous iterations corresponding to `EmulatePhase`. We write $\mathscr{D}_j$, $1 \le j \le \tau$, to denote this distribution of assignments before the execution of j-th iteration of `LocalPhase` on the induced subgraphs, which corresponds to the j-th iteration of `EmulatePhase` for this iteration of the main loop of `ParallelAlg`. Additionally, we write $\mathscr{D}_{\tau+1}$ to denote the same distribution after the τ-th iteration, i.e., at the end of the execution of the parallel block in Lines 6–8 of `ParallelAlg`. Due to Lemma 2.9, the distributions of assignments are all independent. We define ϵ_j, $j \in [\tau+1]$, to be the minimum positive value such that $\mathscr{D}_j$ is ϵ_j-near uniform. Obviously, $\epsilon_1 = 0$, since the first distribution corresponds to a perfectly uniform assignment. We want to apply Lemma 2.10 inductively to bound the value of ϵ_{j+1} as a function of ϵ_j with high probability. The lemma lists two conditions: ϵ_j must be at most $(200\ln n)^{-1}$ and the threshold passed to `EmulatePhase` has to be at least $4000\mu_H^{-2}\ln^2 n$. The latter condition holds due to Lemma 2.15. Hence as long as ϵ_j is sufficiently small, Lemma 2.10 implies that with probability at least $1-n^{-4}$,

$$\epsilon_{j+1} \le 60\alpha\left(\left(\frac{\Delta}{2^{\tau-1}m}\right)^{-1/4} + \epsilon_j\right) \le 60\alpha\left(\left(\frac{\Delta}{m}\right)^{-15/64} + \epsilon_j\right),$$

and no high degree vertex survives in the remaining graph. One can easily show by induction that if this recursion is satisfied for all $1 \le j \le \tau$, then $\epsilon_j \le (120\alpha)^{j-1}\cdot\left(\frac{\Delta}{m}\right)^{-15/64}$ for all $j \in [\tau+1]$. In particular, by the definition of τ and Lemma 2.15, for any $j \in [\tau]$,

$$\epsilon_j \le (120\alpha)^{\tau-1}\cdot\left(\frac{\Delta}{m}\right)^{-15/64} \le \left(\frac{\Delta}{m}\right)^{1/16}\cdot\left(\frac{\Delta}{m}\right)^{-15/64} \le \left(\frac{\Delta}{m}\right)^{-11/64} \le (200\ln n)^{-1},$$

This implies that as long the unlikely events specified in Lemma 2.10 do not occur for any phase in any iteration of the main loop of `ParallelAlg`, we obtain the desired properties: all intermediate distributions of possible assignments are $(200\ln n)^{-1}$-near uniform and the maximum degree in the graph decreases at the expected rate. It remains to bound the probability of those unlikely events occurring for any phase. By the union bound, their total probability is at most $\log n \cdot n^{-4} \le n^{-3}$. □

We now prove that the algorithm finds a large matching with constant probability.

Theorem 2.18

Let M_{OPT} be an arbitrary maximum matching in a graph G. With $\Omega(1)$ probability, `ParallelAlg` constructs a matching of size $\Omega(|M_{\mathrm{OPT}}|)$.

Proof. By combining Lemma 2.16 and Lemma 2.17, we learn that with probability at least $1 - n\cdot n^{-8} - n^{-3} \ge 1 - 2n^{-3}$, we obtain a few useful properties. First, all relevant distributions corresponding to iterations of `EmulatePhase` are independent and $(200\ln n)^{-1}$-near uniform. Second, the maximum degree in the graph induced by vertices still under consideration is

bounded by $\frac{3}{2}\Delta$ before and after every simulated execution of `EmulatePhase`, where Δ is the corresponding. As a result, no vertex is deleted in Lines 7 or 12 due to the security checks.

We now use Lemma 2.8 to lower bound the expected size of the matching created in every `EmulatePhase` simulation. Let $\tau_\star$ be the number of phases we simulate this way. We have $\tau_\star \leq \log n$. Let $\mathbf{H}_j$, $\mathbf{F}_j$, and $\mathbf{M}_j$ be random variables equal to the total size of sets H_i, F_i, and M_i created in the j-th phase. If the corresponding distribution in the j-th phase is near uniform and the maximum is bounded, Lemma 2.8 yields

$$\mathbb{E}\left[\mathbf{H}_j + \mathbf{F}_j\right] \leq n^{-9} + 1200 \cdot \mathbb{E}\left[\mathbf{M}_j\right],$$

i.e.,

$$\mathbb{E}\left[\mathbf{M}_j\right] \geq \frac{1}{1200}\left(\mathbb{E}\left[\mathbf{H}_j + \mathbf{F}_j\right] - n^{-9}\right).$$

Overall, without the assumption that the conditions of Lemma 2.8 are always met, we obtain a lower bound

$$\sum_{j\in[\tau_\star]} \mathbb{E}\left[\mathbf{M}_j\right] \geq \sum_{j\in[\tau_\star]} \frac{1}{1200}\left(\mathbb{E}\left[\mathbf{H}_j + \mathbf{F}_j\right] - n^{-9}\right) - 2n^{-3}\cdot\frac{n}{2},$$

in which we consider the worst case scenario that we lose as much as $n/2$ edges in the size of the constructed matching when the unlikely negative events happen. `ParallelAlg` continues the construction of a matching by directly simulating the global algorithm. Let $\tau'_\star$ be the number of phases in that part of the algorithm. We define $\mathbf{H}'_j$, $\mathbf{F}'_j$, and $\mathbf{M}'_j$, for $j \in [\tau'_\star]$, to be random variables equal to the size of sets H, F, and $\widetilde{M}$ in `GlobalAlg` in the j-th phase of the simulation. By Lemma 2.5, we have

$$\sum_{j\in[\tau'_\star]} \mathbb{E}\left[\mathbf{M}'_j\right] \geq \sum_{j\in[\tau'_\star]} \frac{1}{50}\left(\mathbb{E}\left[\mathbf{H}'_j + \mathbf{F}'_j\right]\right).$$

By combining both bounds we obtain a lower bound on the size of the constructed matching. Let

$$\mathbf{M}_\star \stackrel{\text{def}}{=} \sum_{j\in[\tau_\star]} \mathbb{E}\left[\mathbf{M}_j\right] + \sum_{j\in[\tau'_\star]} \mathbb{E}\left[\mathbf{M}'_j\right]$$

be the expected matching size, and let

$$\mathbf{V}_\star \stackrel{\text{def}}{=} \sum_{j\in[\tau_\star]} \mathbb{E}\left[\mathbf{H}_j + \mathbf{F}_j\right] + \sum_{j\in[\tau'_\star]} \mathbb{E}\left[\mathbf{H}'_j + \mathbf{F}'_j\right].$$

We have

$$\mathbf{M}_\star \geq \frac{1}{1200}\mathbf{V}_\star - \frac{1}{n^2}.$$

Consider a maximum matching M_{OPT}. At the end of the algorithm, the graph is empty. The expected number of edges in M_{OPT} incident to a vertex in one of the reference sets is bounded by $|M_{\text{OPT}}|\cdot 2\mu_R \cdot \log n \leq 10^{-5}|M_{\text{OPT}}|$. The expected number of edges removed by the security checks is bounded by $\frac{n}{2}\cdot n^{-3}$. Hence the expected number of edges in M_{OPT} deleted as incident to vertices that are heavy or are friends is at least $(1-10^{-5})|M_{\text{OPT}}| - 1/(2n^2)$. Since we can assume without the loss of generality that the graph is non-empty, it is at least $\frac{1}{2}|M_{\text{OPT}}|$. Hence $\mathbf{V}_\star \geq \frac{1}{2}|M_{\text{OPT}}|$, and $\mathbf{M}_\star \geq \frac{1}{2400}|M_{\text{OPT}}| - \frac{1}{n^2}$. For sufficiently large n (say, $n \geq 50$), $\mathbf{M}_\star \geq \Omega(|M_{\text{OPT}}|)$ and by an averaging argument, `ParallelAlg` has to output an $O(1)$-multiplicative approximation to the maximum matching with $\Omega(1)$ probability. For smaller n, it is not difficult to show that at least one edge is output by the algorithm with constant probability as long as it is not empty. □

Finally, we want to argue that the above procedure can be used to compute $2+\epsilon$ approximation to maximum matching at the cost of increasing the running time by a factor of $\log(1/\epsilon)$. The idea is to; execute algorithm `ParallelAlg` to compute constant approximate matching; remove this matching from the graph; and repeat.

Corollary 2.19. *Let M_{OPT} be an arbitrary maximum matching in a graph G. For any $\epsilon > 0$, executing* `ParallelAlg` *on G and removing a constructed matching repetitively, $O(\log(1/\epsilon))$ times, finds a multiplicative $(2+\epsilon)$-approximation to maximum matching, with $\Omega(1)$ probability.*

Proof. Assume that the `ParallelAlg` succeeds with probability p and computes c-approximate matching. Observe that each successful execution of `ParallelAlg` finds a matching M_c of size at least $\frac{1}{c}|M_{\text{OPT}}|$. Removal of M_c from the graph decreases the size of optimal matching by at least $\frac{1}{c}|M_{\text{OPT}}|$ and at most by $\frac{2}{c}|M_{\text{OPT}}|$, because each edge of M_c can be incident to at most two edges of M_{OPT}. Hence, when the size of the remaining matching drops to at most $\epsilon|M_{\text{OPT}}|$, we have an $2+\epsilon$-multiplicative approximation to maximum matching constructed. The number t of successful applications of `ParallelAlg` need to satisfy.

$$\left(1-\frac{1}{c}\right)^t \leq \epsilon.$$

This gives $t = O(\log(1/\epsilon))$. In $\lceil t/p \rceil = O(\log(1/\epsilon))$ executions, we have t successes with probability at least $1/2$ by the properties of the median of the binomial distribution. □

2.7 MPC Implementation Details

In this section we present details of an MPC implementation of our algorithm. We also analyze its round and space complexity. In the description we heavily use some of the subroutines described in [GSZ11]. While the model used there is different, the properties of the distributed model used in [GSZ11] also hold in the MPC model. Thus, the results carry over to the MPC model.

The results of [GSZ11] allow us to sort a set A of $O(N)$ key-value pairs of size $O(1)$ and for every element of a sorted list, compute its index. Moreover, we can also do a parallel search: given a collection A of $O(N)$ key-value pairs and a collection of $O(N)$ queries, each containing a key of an element of A, we can annotate each query with the corresponding key-value pair from A. Note that multiple queries may ask for the same key, which is nontrivial to parallelize. If $S = n^{\Omega(1)}$, all the above operations can be implemented in $O(1)$ rounds.

The search operation allows us to broadcast information from vertices to their incident edges. Namely, we can build a collection of key-value pairs, where each key is a vertex and the value is the corresponding information. Then, each edge $\{u, v\}$ may issue two queries to obtain the information associated with u and v.

2.7.1 `GlobalAlg`

We first show how to implement `GlobalAlg`, which is called in Line 13 of `ParallelAlg`.

Lemma 2.20. *Let $S = n^{\Omega(1)}$. There exists an implementation of* `GlobalAlg` *in the MPC model, which with high probability executes $O(\ln \widetilde{\Delta})$ rounds and uses $O(S)$ space per machine.*

Proof. We first describe how to solve the following subproblem. Given a set X of marked vertices, for each vertex v compute $|N(v) \cap X|$. When all vertices are marked, this just computes the degree of every vertex.

The subproblem can be solved as follows. Create a set $A_X = \{(u,v) \mid u \in V, v \in X, \{u,v\} \in E\} \cup \{(v,-\infty),(v,\infty) \mid v \in V\}$, and sort its elements lexicographically. Denote the sorted sequence by Q_X. Then, for each element of A_X compute its index in Q_A.

Note that $|N(v) \cap X|$ is equal to the number of elements in Q_X between $(v,-\infty)$ and (v,∞). Thus, having computed the indices of these two elements, we can compute $|N(v) \cap X|$.

Let us now describe how to implement `GlobalAlg`. We can compute the degrees of all vertices, as described above. Once we know the degrees, we can trivially mark the vertices in H. The next step is to compute F and for that we need to obtain $|N(v) \cap H|$, which can be done as described above.

After that, `GlobalAlg` computes a matching in $G[H \cup F]$ by calling `MatchHeavy` (see Algorithm 2). In the first step, `MatchHeavy` assigns to every $v \in F$ a random neighbor $v_\star$ in H. This can again be easily done by using the sequence Q_H (i.e. Q_X build for $X = H$). Note that for each $v \in F$ we know the number of neighbors of v that belong to H. Thus, each vertex v can pick an integer $r_v \in [1, |N(v) \cap H|]$ uniformly at random. Then, by adding r_v and the index of $(v,-\infty)$ in Q_H, we obtain the index in Q_H, which corresponds to an edge between v and its random neighbor in H. The remaining lines of `MatchHeavy` are straightforward to implement. The vertices can trivially pick their colors. After that, the set $E_\star$ can be easily computed by transmitting data from vertices to their adjacent edges. Implementing the following steps of `MatchHeavy` is straightforward. Finally, picking the edges to be matched is analogous to the step, when for each $v \in F$ we picked a random neighbor in H.

Overall, each phase of `GlobalAlg` (that is, iteration of the main loop) is executed in $O(1)$ rounds. Thus, by Lemma 2.5, `GlobalAlg` can be simulated in $O(\ln \widetilde{\Delta})$ rounds as advertised. □

2.7.2 Vertex and Edge Partitioning

We now show how to implement Line 5 and compute the set of edges that are used in each call to `LocalPhase` in Line 8 of `ParallelAlg`. Our goal is to annotate each edge with the machine number it is supposed to go to. To that end, once the vertices pick their machine numbers, we broadcast them to their adjacent edges. Every edge that receives two equal numbers x is assigned to machine x.
In the implementation we do not check whether a machine is assigned too many edges (Line 7), but rather show in Lemma 2.16 that not too many edges are assigned with high probability.

2.7.3 `LocalPhase`

We now discuss the implementation of `LocalPhase`. Observe that `LocalPhase` is executed locally. Therefore, the for loop at Line 8 of `ParallelAlg` can also be executed locally on each machine. Thus, we only explain how to process the output of `LocalPhase`.

Instead of returning the set of vertices and matched edges at Line 6 of `LocalPhase`, each vertex that should be returned is marked as `discarded`, and each matched edge is marked as `matched`. After that, we need to discard edges, whose at least one endpoint has been discarded. This can be done by broadcasting information from vertices to adjacent edges. Note that some

of the discarded edges might be also marked as `matched`.

2.7.4 Concluding Proofs

Lines 5, 7 and 8 can be implemented as described in sections 2.7.2 and 2.7.3. Lines 9 and 10 do not need an actual implementation, as by that point all the vertices that are not marked as `discarded` constitute V', and all the edges incident to $V \setminus V'$ will be marked as `discarded`. Similarly, all the matched edges will be marked as `matched` by the implementation of `LocalPhase`. All the edges and vertices that are marked as `discarded` will be ignored in further processing. After all the rounds are over, the matching consists of the edges marked as `matched`.

Let $\Delta_\star$ be the value of Δ at Line 12, and hence the value of Δ at the end of the last while loop iteration. Let Δ' be the value of Δ just before the last iteration, i.e. $\Delta_\star = \Delta'/2^\tau$, for the corresponding τ. Now consider the last call of `LocalPhase` at Line 8. The last invocation has $\Delta'/(2^{\tau-1})$ as a parameter. On the other hand, by Claim 2.13 and Claim 2.14 we know that after the last invocation of `LocalPhase` with high probability there is no vertex that has degree greater then $\frac{3}{4}\Delta'/(2^{\tau-1}) < 2\Delta_\star$. Therefore, with high probability there is no vertex that should be removed at Line 12, and hence we do not implement that line either.

An implementation of Line 13 is described in Section 2.7.1. Finally, we can state the following result.

Lemma 2.21. *There exists an implementation of* `ParallelAlg` *in the MPC model that with high probability executes* $O\left((\log\log n)^2 + \max\left(\log \frac{n}{S}, 0\right)\right)$ *rounds.*

Proof. In the proof we analyze the case $S \le n$. Otherwise, for the case $S > n$, we think of each machine being split into $\lfloor S/n \rfloor$ "smaller" machines, each of the smaller machines having space n.

We will analyze the number of iterations of the while loop `ParallelAlg` performs. Let Δ_i and τ_i be the value of Δ and τ at the end of iteration i, respectively. Then, from Line 3 and Line 4 we have

$$\tau_i = \left\lceil \frac{1}{16}\log_{120\alpha}(\Delta_{i-1}/m) \right\rceil \ge \frac{1}{16}\log_{120\alpha}(\Delta_{i-1}/m) \ge \frac{1}{16}\log_{120\alpha}\sqrt{\frac{S\Delta_{i-1}}{n}}.$$

Define $\gamma \stackrel{\text{def}}{=} \frac{1}{32\log_2 120\alpha}$. By plugging in the above bound on τ_i, from Line 11, we derive

$$\Delta_i = \Delta_{i-1}\cdot 2^{\tau_i} \le \Delta_{i-1}\cdot 2^{-\frac{1}{16}\log_{120\alpha}\sqrt{\frac{S\Delta_{i-1}}{n}}} = \Delta_{i-1}\cdot 2^{-\frac{\log_2 \frac{S\Delta_{i-1}}{n}}{32\log_2 120\alpha}} = \Delta_{i-1}^{1-\gamma}\left(\frac{n}{S}\right)^{\gamma} \tag{2.6}$$

To obtain the number of iterations the while loop of `ParallelAlg` performs, we derive for which $i \ge 1$ the condition at Line 2 does not hold.

Unraveling Δ_{i-1} further from (2.6) gives

$$\Delta_i \le \Delta_0^{(1-\gamma)^i}\left(\frac{n}{S}\right)^{\gamma\sum_{j=0}^{i-1}(1-\gamma)^j} \le n^{(1-\gamma)^i}\left(\frac{n}{S}\right)^{\gamma\frac{1-(1-\gamma)^i}{1-(1-\gamma)}} = n^{(1-\gamma)^i}\left(\frac{n}{S}\right)^{1-(1-\gamma)^i} \tag{2.7}$$

Observe that $(c\log\log n)^{-1} \le \gamma \le (32\log\log n)^{-1} < 1/2$, for an absolute constant c and $n \ge 4$.

For $S \le n$ and as $\gamma < 1/2$ we have

$$\left(\frac{n}{S}\right)^{1-(1-\gamma)^i} \le \frac{n}{S}. \tag{2.8}$$

On the other hand, for $i_\star = \frac{\log\log n}{\gamma} \le c(\log\log n)^2$ we have

$$n^{(1-\gamma)^{i_\star}} < \log n. \tag{2.9}$$

Now putting together (2.7), (2.8), and (2.9) we conclude

$$\Delta_{i_\star} < \frac{n}{S}\ln n,$$

and hence the number of iteration the while loop of `ParallelAlg` performs is $O\big((\log\log n)^2\big)$.

Total round complexity. Every iteration of the while loop can be executed in $O(1)$ MPC rounds with probability at least $1 - 1/n^3$. Since there are $O\big((\log\log n)^2\big)$ iterations of the while loop, all the iterations of the loop can be performed in $O\big((\log\log n)^2\big)$ many rounds with probability at least $1 - 1/n^2$.

On the other hand, by Lemma 2.20 and the condition at Line 2 of `ParallelAlg`, the computation of Line 13 of `ParallelAlg` can be performed in $O\big(\log\big(\frac{n}{S}(\ln n)^{32}\big)\big)$ rounds. Putting the both bounds together we conclude that the round complexity of `ParallelAlg` is $O\big((\log\log n)^2 + \log\frac{n}{S}\big)$ for the case $S \le n$. For the case $S > n$ (recall that in this regime we assume that each machine is divided into machines of space n) the round complexity is $O\big((\log\log n)^2\big)$. □

3 Maximal Independent Sets in Parallel Computation

This chapter is based on a joint work with Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, and Ronitt Rubinfeld. It has been accepted to ACM Symposium on Principles of Distributed Computing (**PODC**) 2018 [GGK+18] under the title

Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover.

3.1 Introduction

In this chapter, we study one of the most fundamental problems in algorithmic graph theory – maximal independent set (MIS). The study of this problem in models of parallel computation dates back to PRAM algorithm. A seminal work of Luby [Lub86] gives a simple randomized algorithm for constructing MIS in $O(\log n)$ PRAM rounds. Similar results, also in the context of PRAM algorithms, were obtained in [ABI86, II86, IS86]. Since then, MIS was studied quite extensively in various models of computation. We design a simple randomized algorithm that constructs MIS in the MPC and the CONGESTED-CLIQUE model.

3.1.1 Model

We consider two closely related models: *Massively Parallel Computation* (MPC), and the CONGESTED-CLIQUE model of computing. We refer a reader to Section 2.1.1 for the definition of the MPC model.

CONGESTED-CLIQUE

The CONGESTED-CLIQUE model was introduced by Lotker, Pavlov, Patt-Shamir, and Peleg [LPPSP03] and has been studied extensively since then, see e.g., [PST11, DLP12, BHP12, Len13, DKO14, Nan14, HPS14, HP14, CHKK+15, HPP+15, BFARR15, Gha16, GP16, Kor16, HKN16, CHPS17, Gha17, JN18]. In this model, we have n players which can communicate in synchronous rounds. In each round, every player can send $O(\log n)$ bits to every other player. Besides this communication restriction, the model does not limit the players, e.g., they can use large space and arbitrary computations; though, in our algorithms, both of these will be small. Furthermore, in studying graph problems in this model, the standard setting is that we have an n-vertex graph $G = (V, E)$, and each player is associated with one vertex of this graph.

Initially, each player knows only the edges incident on its own vertex. At the end, each player should know the part of the output related to its own vertex, e.g., whether its vertex is in the computed maximal independent set or not, or whether some of its edges is in the matching or not.

We emphasize that CONGESTED-CLIQUE provides an all-to-all communication model. It is worth contrasting this with the more classical models of distributed computing. For instance, the LOCAL model, first introduced by Linial [Lin87], allows the players to communicate only along the edges of the graph problem G (with unbounded size messages).

3.1.2 Our Results

In Section 3.4 we present an algorithm for constructing MIS in the MPC and the CONGESTED-CLIQUE model.

Theorem 3.1

For any graph with maximum degree Δ, there is an algorithm that with high probability computes an MIS in $O(\log\log\Delta)$ rounds of the MPC model, with $\Theta(n)$ space per each machine. Moreover, the same algorithm can be adapted to compute an MIS in $O(\log\log\Delta)$ rounds of the CONGESTED-CLIQUE model.

3.1.3 Related Work

Maximal independent set has been central in the study of graph algorithms in both the parallel and the distributed models. The seminal work of Luby [Lub86] and Alon, Babai, and Itai [ABI86] provide $O(\log n)$-round parallel and distributed algorithms for constructing MIS. The distributed complexity in the LOCAL model was first improved by Barenboim et al.[BEPS12] and consequently by Ghaffari [Gha16], which led to the current best round complexity of $O(\log\Delta)+2^{O(\sqrt{\log\log n})}$. In the CONGESTED-CLIQUE model of distributed computing, Ghaffari [Gha17] gave another algorithm which computes an MIS in $\tilde{O}(\sqrt{\log\Delta})$ rounds. A deterministic $O(\log n\log\Delta)$-round CONGESTED-CLIQUE algorithm was given by Censor-Hillel et al. [CHPS17].

It is also worth referring to the literature on one particular MIS algorithm, known as the *randomized greedy MIS*, which is relevant to what we do for MIS. In this algorithm, we permute the vertices uniformly at random and then add them to the MIS greedily. Blelloch et al. [BFS12] showed that one can implement this algorithm in $O(\log^2 n)$ parallel/distributed rounds, and recently Fischer and Noever [FN18] improved that to a tight bound of $\Theta(\log n)$. We will show a $O(\log\log\Delta)$-round simulation of the randomized greedy MIS algorithm in the MPC and the CONGESTED-CLIQUE model.

3.2 Preliminaries

For a graph $G=(V,E)$ and a set $V'\subseteq V$, $G[V']$ denotes the subgraph of G *induced* on the set V', i.e., $G[V']=(V',E\cap(V'\times V'))$. We use $N(v)$ to refer to the neighborhood of v in G. Throughout the chapter, we use $n\stackrel{\text{def}}{=}|V|$ to denote the number of vertices in the input graph.

3.3 Overview and Organization

Our MIS algorithm is based on the randomized greedy MIS algorithm (`RandGreedyMIS`). This algorithm ranks/permutes vertices 1 to n randomly and then greedily adds vertices to the MIS, while walking through this permutation. We provide this algorithm below.

Algorithm 6: `RandGreedyMIS`(G)
Randomized greedy MIS algorithm

Input: Graph $G = (V, E)$
Output: An MIS in G

$V' \leftarrow V$
Choose a permutation $\pi : [n] \rightarrow [n]$ uniformly at random.
while $V' \neq \emptyset$ **do**
    Select the vertex v which according to π has the smallest rank in V'.
    Add v to the MIS.
    Remove v and $N(v)$ from V'.
return *the constructed MIS*

This algorithm has been studied before in the literature of parallel algorithms [FN18, BFS12]. One of the features of `RandGreedyMIS` is that, informally speaking, high-degree vertices get removed quickly and hence the maximum degree of the graph decreases at a high rate. In the rest of this chapter, we show how to efficiently implement a variant of this algorithm in only $O(\log\log\Delta)$ MPC and CONGESTED-CLIQUE rounds.

3.4 Maximal Independent Set

In this section, we simulate `RandGreedyMIS` in $O(\log\log\Delta)$ rounds of the MPC and the CONGESTED-CLIQUE model, thus proving the following result:

Theorem 3.1

For any graph with maximum degree Δ, there is an algorithm that with high probability computes an MIS in $O(\log\log\Delta)$ rounds of the MPC model, with $\Theta(n)$ space per each machine. Moreover, the same algorithm can be adapted to compute an MIS in $O(\log\log\Delta)$ rounds of the CONGESTED-CLIQUE model.

3.4.1 A Variant of `RandGreedyMIS`

We first formulate a variant of `RandGreedyMIS`, that we call `RandMPCMIS` and show how to implement in the CONGESTED-CLIQUE and the MPC model.

Algorithm 7: `RandMPCMIS`(G)
A variant of `RandGreedyMIS`

Input: Graph $G = (V, E)$
Output: An MIS in G

$V' \leftarrow V$
Choose a permutation $\pi : [n] \to [n]$ uniformly at random.
repeat
  Select the vertex v which according to π has the smallest rank in V'.
  Add v to the MIS.
  Remove v and $N(v)$ from V'.
until *the next rank is at least* $n/\log^{10} n$ *and the maximum degree in* $G[V']$ *is at most* $\log^{10} n$
Run $O(\log\log\Delta)$ rounds of the Sparsified MIS Algorithm of [Gha17] on $G[V']$. Remove from V' the constructed MIS and its neighborhood.
Deliver $G[V']$ to a single machine and find its MIS.
return *union of the constructed MIS sets*

3.4.2 Simulation of `RandMPCMIS` in $O(\log\log\Delta)$ Rounds of MPC and CONGESTED-CLIQUE

Simulation in the MPC model We now describe how to simulate `RandMPCMIS` in the MPC model with $\Theta(n)$ space per machine. In each iteration, we consider an induced subgraph of G that is guaranteed to have $O(n)$ edges and simulate the above algorithm on that graph. We show that the total number of edges drops fast enough, so that $O(\log\log\Delta)$ rounds will suffice.

More concretely, we first consider the subgraph induced by vertices with ranks 1 to n/Δ^{α}, for $\alpha = 3/4$. This subgraph has $O(n)$ edges, with high probability. So we can deliver it to one machine, and have it simulate the algorithm up to this rank. Now, this machine sends the resulting MIS to all other machines. Then, each machine removes its vertices that are in MIS or neighboring MIS. In the second phase, we take the subgraph induced by remaining vertices with ranks n/Δ^{α} to n/Δ^{α^2}. Again, we can see that this subgraph has $O(n)$ edges (a proof is given below), so we can simulate it in $O(1)$ rounds. More generally, in the i-th iteration, we will go up to rank n/Δ^{α^i}.

Once the next rank becomes $n/\log^{10} n$, which as we show happens after $O(\log\log\Delta)$ rounds, the maximum degree of the graph is some value $\Delta' \leq O\left(\log^{11} n\right)$ (see Lemma 3.2). Note that clearly also $\Delta' \leq \Delta$. At that point, we apply the MIS Algorithm of [Gha17] for sparse graphs to the remaining graph. This algorithm is applicable whenever the maximum degree is at most $2^{O\left(\sqrt{\log n}\right)}$ (see Theorem 1.1 of [Gha17]). After $O(\log\log\Delta')$ rounds, w.h.p., that algorithm finds an MIS which after removed along with its neighborhood results in the graph having $O(n)$ edges. Now we deliver the whole remaining graph to one machine where it is processed in a single MPC round.

We note that the Algorithm of [Gha17] performs only simple local decisions with low communication, and hence every iteration of the algorithm can be implemented in $O(1)$ MPC rounds, with $\Theta(n)$ space per machine, by using standard techniques.

Simulation in CONGESTED-CLIQUE We now argue that each iteration of the described MPC simulation can be implemented in $O(1)$ rounds of CONGESTED-CLIQUE. For that, in

each iteration, we make all vertices with permutation rank in the selected range send their edges to the leader vertex. Here, the leader is an arbitrarily chosen vertex, e.g., the one with the minimum identifier. As we show below, the number of these edges per iteration is $O(n)$ with high probability, and thus we can deliver all the messages to the leader in $O(1)$ rounds using Lenzen's routing method [Len13]. Then, the leader can compute the MIS among the vertices with ranks in the selected range. It then reports the result to all the vertices in a single round, by telling each vertex whether it is in the computed independent set or not. A single round of computation, in which the vertices in the independent set report to all their neighbors, is then used to remove all the vertices that have a neighbor in the independent set (or are in the set). After these steps, the algorithm proceeds to the next iteration.

Regarding the round complexity of the algorithm once the rank becomes $n/\log^{10} n$: The work [Gha17] already provides a way to solve MIS in $O(\log\log\Delta')$ CONGESTED-CLIQUE rounds for any $\Delta' = 2^{O\left(\sqrt{\log n}\right)}$. Here, Δ' is the maximum degree of the graph remained after processing the vertices up to rank $n/\log^{10} n$, and, as we show by Lemma 3.2, it holds $\Delta' \leq \operatorname{polylog} n \ll 2^{O\left(\sqrt{\log n}\right)}$. Hence, the overall round complexity is again $O(\log\log\Delta)$ rounds.

3.4.3 Analysis

Since by the i-th iteration the algorithm has processed the ranks up to n/Δ^{α^i}, the rank $n/\log^{10} n$ is processed within $O(\log\log\Delta)$ iterations. In the proof of Theorem 3.1 presented below, we prove that with high probability the number of edges sent to one machine per phase is $O(n)$. Before that, we present a lemma that will aid in bounding the degrees and the number of edges in our analysis.

Lemma 3.2. *Suppose that we have simulated the algorithm up to rank r. Let G_r be the remaining graph. Then, the maximum degree in G_r is $O(n\log n/r)$ with high probability.*

Proof. We first upper-bound the probability that G_r contains a vertex of degree at least d. Then, we conclude that the degree of every vertex in G_r is $O(n\log n/r)$ with high probability.

Consider a vertex v whose degree is still d. When the sequential algorithm considers one more vertex, which is selected by choosing a random vertex among the remaining vertices, then vertex v or one of its neighbors gets chosen with probability at least d/n. If that happens, then v is removed. The probability that this does not happen throughout ranks 1 to r is at most $(1-d/n)^r \leq \exp(-rd/n)$. Now, the probability that a vertex in G_r has degree more than $20n\log n/r$ is at most $1/n^5$, which implies that, the maximum degree of G_r is at most $20n\log n/r$ with probability at least $1-n^{-4}$. □

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

Proof of Theorem 3.1. We first argue about the MPC-round complexity of the algorithm, and then show that it requires $\Theta(n)$ space.

Round complexity Recall that the algorithm considers ranks of the form $r_i \stackrel{\text{def}}{=} n/\Delta^{\alpha^i}$, until the rank becomes $n/\log^{10} n$ or greater. When that occurs, it applies other algorithms for $O(\log\log\Delta)$ iterations, as described in Section 3.4.2. Hence, the algorithm runs for at most $i_\star + \log\log\Delta$ iterations, where $i_\star$ is the smallest integer such that rank $r_{i_\star} = n/\Delta^{\alpha^{i_\star}} \geq n/\log^{10} n$.

A simple calculation gives $i_\star \le \log_{4/3} \log \Delta$, for $\alpha = 3/4$. Furthermore, every iteration can be implemented in $O(1)$ rounds as discussed above.

Space requirement We first discuss the space required to implement the process until the rank becomes $O\left(n/\log^{10} n\right)$. By Lemma 3.2 we have that after the graph up to rank r_i is simulated, the maximum degree in the remaining graph is $O(n \log n / r_i)$ w.h.p. Observe that it also trivially holds in the first iteration, i.e., the initial graph has maximum degree $O(n)$. Let G_i be the graph induced by the ranks between r_i and r_{i+1}. Then, a neighbor u of vertex v appears in G_i with probability $(r_{i+1} - r_i)/(n - r_i) \le r_{i+1}/n$. Hence, the expected degree of every vertex in this graph is at most

$$\mu \stackrel{\text{def}}{=} \Theta(n \log n / r_i \cdot r_{i+1}/n) = \Theta\left(\Delta^{(1-\alpha)\alpha^i} \log n\right).$$

Since $\mu \ge \log n$, by the Chernoff bound we have that every vertex in G_i has degree $O(\mu)$ w.h.p. Now, since there are $O(r_{i+1})$ vertices in G_i, we have that G_i contains

$$O\left(r_{i+1} \Delta^{(1-\alpha)\alpha^i} \log n\right) = O\left(n \Delta^{-\alpha^i/2} \log n\right) \tag{3.1}$$

many edges w.h.p., where we used that $\alpha = 3/4$. Recall that the algorithm iterates over the ranks until the maximum degree becomes less than $\log^{10} n$. Also, $\Theta(n \log n / r_i)$ upper-bounds the maximum degree (see Lemma 3.2). Hence, we have

$$\Theta(n \log n / r_i) \ge \log^{10} n \implies \Delta^{\alpha^i} \ge \Omega\left(\log^9 n\right).$$

Combining the last implication with (3.1) provides that G_i contains $O(n)$ edges w.h.p.

After the rank becomes $n/\log^{10} n$ or greater, we run the CONGESTED-CLIQUE algorithm of [Gha17] for $O(\log\log\Delta)$ iterations. That algorithm performs only simple local decisions with low communication. Hence, by using standard techniques, every iteration of the algorithm can be implemented in $O(1)$ MPC rounds with $\Theta(n)$ space per machine. Finally, the graph remaining after running $O(\log\log\Delta)$ iterations of that algorithm contains $O(n)$ edges w.h.p. (see Lemma 2.11 of [Gha17]). Hence, we deliver the remaining graph to one machine and construct its MIS.

□

4 Network Routing under Link Failures

This chapter is based on a joint work with Marco Chiesa, Andrei Gurtov, Aleksander Mądry, Ilya Nikolaevskiy, Michael Schapira, and Scott Shenker. It has been accepted to the 43rd International Colloquium on Automata, Languages, and Programming (**ICALP**) 2016 [CGM+16] under the title

On the Resiliency of Randomized Routing Against Multiple Edge Failures.

The techniques that we develop in this chapter are applicable in a context broader than presented in the sequel. For details, we point a reader to our work [CNM+16] and [CNM+17].

4.1 Introduction

Routing on the Internet (within organizational networks and between them) typically involves computing a set of *destination-based* routing tables (i.e., tables that map the destination IP address of a packet to an outgoing link). Whenever a link or node fails, routing tables are recomputed by invoking the routing protocol to run again (or having it run periodically, independent of failures). This produces well-formed routing tables, but results in relatively long outages after failures as the protocol is recomputing routes.

As many critical applications rely on the Internet, such outages became unacceptable. As a result, *fast failover* techniques have been employed to facilitate immediate recovery from failures. The most well-known of these is Fast Reroute in MPLS where, upon a link failure, packets are sent along a precomputed alternative path without waiting for the global re-computation of routes [PSA05]. This, and other similar forms of fast failover thus enable rapid response to failures but are limited to the set of precomputed alternate paths. Most of the existing approaches protect only from single failure, however in many scenarios (e.g. overlay networks [EGR14], highly-connected large datacenter networks [GJN11]) multiple failures at the same time may be a common occurrence.

The fundamental question that we ask in this chapter is:

How resilient can failover routing be?

That is, how many link failures can failover routing schemes tolerate before connectivity is interrupted (i.e., packets are trapped in a forwarding loop, or hit a dead end)? The answer

to this question depends on both the structural properties of the graph, and the limitations imposed on the design of routing scheme.

Clearly, if it is possible to store arbitrary amount of information in the packet header, perfect resiliency can be achieved by collecting information about every failed link that is hit by a packet [LCR+07, SCR13]. Such approaches are not feasibly deployable in modern networks as the header of a packet may be too large for the routing tables. Our focus is thus on failover routing schemes that do not involve any change in the packet headers. Another traditional approach to achieving high resiliency is implementing stateful routing, i.e., storing information at a node every time a packet is being received from a different incoming link (see, e.g., link reversal [GB81] and other approaches [LPS+13, LYSS11]). As current routing protocols do not allow network operators to implement such stateful failover routing, our goal is to design protocols that correspond to a stateless, or *static*, failover routing.

Specifically, we consider a particularly simple and practical form of static failover routing: for each incoming link, a router maintains a destination-based routing table that maps the destination address of a packet and the set of non-failed ("active") links, to an output link. The router can locally detect which outgoing links are down and forwards packets accordingly. One should note that maintaining such per-incoming-link destination-based routing tables is necessary; not only is destination-based routing unable to achieve robustness against even a single link failure [KGGZ11], but it is even computationally hard to devise failover routing schemes that maximize the number of nodes that are protected [AZ08, BS13, KGGZ11, SCK+03]. We only consider link failures, not router failures (which are not always detectable by neighboring routers, and so such fast failover techniques may not apply).

A failover routing algorithm is responsible for computing, for each node (vertex) of a network (graph), a *routing function* that *matches* an incoming packet to an outgoing edge. A set of such routing functions for each vertex guarantees *reachability* between a pair of vertices, u and v, for which there exists a connecting path in the graph, if any packet directed to node v originated at node u is correctly routed from u to v.

We are interested in routing functions that rely solely on information that is locally available at a node (e.g., the set of non-failed edges, the incoming link along which the packet arrived, and any information stored in the header of the packet).

While it is known that every k-connected network cannot be partitioned by deleting at most $k-1$ links, it is not known whether any static "deterministic" routing (i.e., the outgoing port of each packet is always uniquely determined at a vertex v by its incoming link and the failed edges incident at v) achieves such resiliency.

On the other hand, routing based on random walks, i.e., choosing the outgoing link at random, achieves the best possible resilience as they will eventually deliver a packet to the destination as long as the network is connected. But, it comes with a huge cost. Namely, a random walk might traverse the whole network even when there is no single failed link. In fact, the expected delivery time of a packet would be as large as $\Theta(|V|^3)$ in some network topologies [BW90]. Furthermore, a random walk almost never reaches the destination following the shortest path. So, although extremely robust, when it comes to the time needed to deliver a packet to the destination, the behavior of random walks is undesirable.

4.1.1 Model

We represent our network as an undirected multigraph $G = (V(G), E(G))$, where each router in the network is modeled by a vertex in $V(G)$ and each link between two routers is modeled by an undirected edge in the multiset $E(G)$. When it is clear from the context, we simply write V and E instead of $V(G)$ and $E(G)$. We denote an (undirected) edge between x and y by $\{x, y\}$. A graph is *k-edge-connected* if there exist k edge-disjoint paths between any pair of vertices of G.

Each vertex v routes packets according to a *routing function* that matches an incoming packet to a sequence of forwarding actions. Packet *matching* is performed according to the set of active (non-failed) edges incident at v, the incoming edge, and any information stored in the packet header (e.g., destination label, extra bits), which all are *locally* available at a vertex.

Since our focus is on per-destination routing functions, we assume that there exists a unique destination $d \in V$ to which every other vertex wishes to send packets and, therefore, that the destination label is not included in the header of a packet. Forwarding *actions* consist of routing packets through an outgoing edge, rewriting some bits in the packet header, and creating duplicates of a packet.

In our work we consider *randomized* routing functions, in which a vertex forwards a packet through an outgoing edge with a probability based only on the incoming port and the set of active outgoing edges. We present the formal definitions of the randomized routing model in Section 4.1.2.

The STATIC-ROUTING-RESILIENCY (SRR) problem. Given a graph G, a routing function f is *k-resilient* if, for each vertex $v \in V$, a packet originated at v and routed according to f reaches its destination d as long as at most k edges fail and there still exists a path between v and d. The input of the SRR problem is a graph G, a destination $d \in V(G)$, and an integer $k > 0$, and the goal is to compute a set of resilient routing functions that is k-resilient.

4.1.2 Our Results

In this chapter, we show how randomness can be used to achieve $k-1$ resilient routing in k-connected networks while significantly outperforming random walks in terms of number of traversed nodes. Namely, we introduce **Randomized failover routing** (`RND`) in which outgoing edge is chosen for packets in a probabilistic manner based on the destination label, the incoming edge, and the set of non-failed edges. The randomized protocol that we present provides bound on the expected delivery time that gracefully grows with the number of actual link failures.

Our randomized routing functions provide delivery in case of any $k-1$ link failures for any k-connected graph. We achieve that by leveraging the standard decomposition of k-connected graphs into k arc-disjoint spanning arborescences $\mathcal{T}$ [Edm72]. We also provide a bound on the expected number of hops that our algorithm performs, which is $O(Hk)$ for any $k-1$ failures and $O(H)$ for αk failures, where H is the length of the longest branch of any arborescence of $\mathcal{T}$ and $\alpha < 1$ is a constant. Furthermore, our routing functions are deterministic as long as the routing does not encounter any failure. Hence, packets belonging to the same logical connection are routed along the same path, minimizing reordering complexity at the receiver side. More precisely, we show the following theorem.

Theorem 4.1

Given a k-connected graph G and a destination d, there exists a $(k-1)$-resilient algorithm that delivers a packet to d after $O\left(\frac{k}{k-f}H\right)$ hops in expectation, where H is the length of a longest path of any arborescence of $\mathcal{T}$ and f the number of failed edges. The algorithm uses randomization only when encounters a failed edge. In particular, if $f = 0$, the algorithm is deterministic.

4.1.3 Related Work

Past work [AZ06, YLS+14] (1) designed such routing functions with guaranteed robustness against *only* a single link/node failure [ERC07, FGP+12, NLY+07, WN07, ZWB13, ZNY+05], (2) achieved robustness against $\lfloor \frac{k}{2} - 1 \rfloor$ edge failures for k-connected graphs [EGR14], and (3) proved that it is impossible to be robust against any set of edge failures that does not partition the network [FGP+12].

Thanks to its flexibility and oblivious behavior, another line of study was motivated by randomization. Namely, some of the previous work developed randomized routing schemes, usually to directly or indirectly achieve low congestion and/or balance the network load. In particular, Busch et al. [BHW00] use randomization to adjust packet priorities, which in turn allows them to control deflection of packets.

Valiant [Val82] proposed a randomized routing algorithm with the goal to balance the load of the underlying network. Since then, that scheme is called *Valiant Load-Balancing* (VLB), whose one of the main ingredients is randomization. VLB was extensively used in designing networks. Zhang-Shen et al. [ZSM08] employed VLB to design fault-tolerant networks with guaranteed no congestion under few router or link failures. Greenberg et al. [GHJ+09] adopt VLB to reduce volatility of traffic and failure pattern of their data centers. In [SW06], Shepherd et al. extend VLB in order to build cost-effective networks robust to changes in demand patterns.

Beraldi [Ber09] presents a search protocol for mobile networks that is based on modified random walks, i.e. based on biased random walks with look-ahead. Motivated by the success of ant-colonies in their search for food, Günes et al.[GKB03] studied ant algorithms, which in their heart rely on randomization, as an approach to designing on-demand ad-hoc routing algorithms.

Chiesa et al. [CNM+16] studied resilience under link failures in k-connected networks. They devise static routing schemes that are resilient under $k-1$ failures in the following regimes: (1) if the routers are allowed to use three bits in the packet header for read/write operation, or (2) if the network supports broadcasting. A building block of those schemes is the result that every k-connected graph contains k arc-disjoint arborescences rooted at the same vertex [Edm72].

4.2 Preliminaries

We denote a directed arc from x to y by (x, y) and by $\vec{G}$ the directed copy of G, i.e. a directed graph such that $V(\vec{G}) = V$ and $\{x, y\} \in E$ if and only if $(x, y), (y, x) \in E(\vec{G})$.

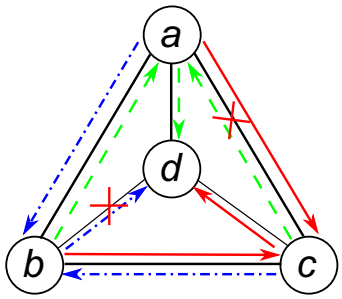

Figure 4.1 – A 3-connected graph with 3 arc-disjoint arborescences colored red, blue, and green.

A subgraph T of $\vec{G}$ is an *r-rooted arborescence* of $\vec{G}$ if (i) $r \in V$, (ii) $V(T) \subseteq V$, (iii) r is the only vertex without outgoing arcs and (iv), for each $v \in V(T) \setminus \{r\}$, there exists a single directed path from v to r that only traverses vertices in $V(T)$. If $V(T) = V$, we say that T is a r-rooted *spanning* arborescence of $\vec{G}$. When it is clear from the context, we use the word "arborescence" to refer to a d-rooted spanning arborescence, where d is the destination vertex. We say that two arborescences T_1 and T_2 are *arc-disjoint* if $(x,y) \in E(T_1) \implies (x,y) \notin E(T_2)$. A set of l arborescences $\{T_1, \ldots, T_l\}$ is *arc-disjoint* if the arborescences are pairwise arc-disjoint. We say that two arc-disjoint arborescences T_1 and T_2 do not *share* an edge $\{x,y\} \in E$ if $(x,y) \in E(T_1) \implies (y,x) \notin E(T_2)$.

For example, consider Figure 4.1, in which each pair of nodes is connected by an edge (ignore the red crosses) and three arc-disjoint (d-rooted spanning) arborescences `Red`, `Green`, and `Blue` are depicted by colored arrows.

4.3 Overview and Organization

Throughout this section, unless specified otherwise, we let $\mathcal{T} = \{T_1, \ldots, T_k\}$ denote a set of k d-rooted arc-disjoint spanning arborescences of $\vec{G}$. All our routing techniques are based on a decomposition of $\vec{G}$ into $\mathcal{T}$. The existence of k arc-disjoint arborescences in any k-connected graph was proven in [Edm72], while fast algorithms to compute such arborescences can be found in [BHKP08]. We say that a packet is routed in *canonical* mode along an arborescence T if a packet is routed through the unique directed path of T towards the destination. If the packet hits a failed edge at vertex v along T, it is processed by v (e.g., duplication, header-rewriting) according to the capabilities of a specific routing function and it is rerouted along a different arborescence. We call such routing technique *arborescence-based* routing. One crucial decision that must be taken is the next arborescence to be used after a packet hits a failed edge. In our work, we propose two natural choices that represent the building blocks of all our routing functions. When a packet is routed along T_i and it hits a failed arc (v,u), we consider the following two possible actions:

- **Reroute along some available arborescence**, e.g., reroute along T', where T' is chosen randomly from distribution that we define in the sequel. Observe that, if the outgoing arc belonging to T' failed, we randomly pick another arborescence T'', and so on.
- **Bounce on the reversed arborescence**, i.e., we reroute along the arborescence T_{next}

that contains arc (u, v).

To grasp how bouncing enters in our picture for obtaining $k-1$ resiliency, consider the following case. Assume that in the network there are $k/2$ failed links, such that every single out of k arborescences contains one of the links. (As a reminder, arborescences that we construct might share links, but not arcs.) So, this example might suggest that there are scenarios in which already $k/2$ failed links make all the arborescences not very useful, and that no algorithm can cope with that. But, there is a twist. Let $k = 2$, and T_i and T_j be the two arborescences and let them share the same failed edge a. Furthermore, let a be the only failed edge T_i and T_j contain. If a packet hits a while routed along T_i or T_j, then after bouncing on a the packet will reach d without any further interruption! So, we have just found a way to resolve a case in which every arborescence contains one failed link, and that is not an isolated scenario, as we discuss in the sequel.

From a different point of view, bouncing is a way of recycling arborescences that contain one failed link. This observation is crucial to obtain an efficient and a simple randomized $(k-1)$-resilient routing scheme, which we are now ready to present. The algorithm is parametrized by q that we define later. In the following sections, we first study the connection between

Algorithm 8: RAND-BOUNCING-ALGO $(\mathcal{T}, d)$
A resilient-routing scheme for G

Input:
- A set of arborescences of G rooted at the same vertex $\mathcal{T} = \{T_1, \ldots, T_k\}$
- destination d

```
1 T ← an arborescence from 𝒯 sampled uniformly at random.
2 while While d is not reached do
3     Route along T (canonical mode)
4     if a failed link is hit then
5         With probability q, replace T by an arborescence from 𝒯 sampled uniformly at
          random.
6         Otherwise, bounce the failed edge and update T correspondingly.
```

arborescences of $\mathcal{T}$ and failed links, and show how a part of their intricate interaction can be represented in a simple and an elegant way via, so-called, *meta-graph*. Afterward, we show the RAND-BOUNCING-ALGO is $(k-1)$-resilient and we analyze its efficiency.

Organization

The rest of this chapter is organized as follows. In Section 4.4, we focus on studying the relation between arborescences our input graph decomposes into and failed links. Section 4.5 builds on Section 4.4 and is devoted to designing an algorithm that, for any k-connected graph, computes randomized routing functions that are robust to $k-1$ edge failures and have bounded expected delivery time.

4.4 Meta-graph, Good Arcs, and Good Arborescences

The goal of this section is to provide an understanding of the structural relation between the arborescences of $\mathcal{T}$ when the underlying k-connected network has at most $k-1$ failed edges. The perspective that we are building here drives the construction of our randomized algorithm.

We start by introducing the notion of a *meta-graph*. To that end, we fix an arbitrary set of failed edges F. Throughout the section, we assume $|F| < k$, and define $f \stackrel{\text{def}}{=} |F|$. Then, we define a meta-graph $H_F = (V_F, E_F)$ as follows:

- $V_F = \{1, \ldots, k\}$, where vertex i is a representative of arborescence T_i.
- For each failed edge $e \in E$ belonging to at least one arborescences of $\mathcal{T}$ we define the corresponding edge e_F in H_F in the following way:
  - $e_F = \{i, j\}$, if e belongs to two different arborescences T_i and T_j;
  - $e_F = \{i, i\}$, i.e., e_F is a self-loop, if e belongs to a single arborescence T_i only.

Note that in our construction H_F might contain parallel edges. Intuitively, the meta-graph represents a relation between arborescences of $\mathcal{T}$ for a fixed set of failed edges. We provide the following lemma as the first step towards understanding the structure of H_F.

Lemma 4.2. *The set of connected components of H_F contains at least $k - f$ trees.*

Proof. We give a proof by contradiction. To that end, assume that the set of connected components of H_F, denoted by $\mathscr{C}$, contains at most $k - f - 1$ trees. Now, if $C \in \mathscr{C}$ is a tree, we have $|E(C)| = |V(C)| - 1$, and $|E(C)| \geq |V(C)|$ otherwise. We also have

$$\begin{aligned} \sum_{C \in \mathscr{C}} |E(C)| &= \sum_{C \in \mathscr{C} \text{ is not a tree}} |E(C)| + \sum_{C \in \mathscr{C} \text{ is a tree}} |E(C)| \\ &\geq \sum_{C \in \mathscr{C} \text{ is not a tree}} |V(C)| + \sum_{C \in \mathscr{C} \text{ is a tree}} (|V(C)| - 1). \end{aligned} \tag{4.1}$$

Next, following our assumption that $\mathscr{C}$ contains at most $k - f - 1$ trees, from (4.1) we obtain

$$\sum_{C \in \mathscr{C}} |E(C)| \geq \sum_{C \in \mathscr{C}} |V(C)| - (k - f - 1). \tag{4.2}$$

Furthermore, as by the construction we have $\sum_{C \in \mathscr{C}} |V(C)| = |V_F| = k$, (4.2) implies

$$\sum_{C \in \mathscr{C}} |E(C)| \geq |V_F| - (k - f - 1) = f + 1. \tag{4.3}$$

On the other hand, from the construction of H_F we have

$$\sum_{C \in \mathscr{C}} |E(C)| = f,$$

which leads to a contradiction with (4.3). □

Lemma 4.2 implies that the fewer failed edges there are, the larger fraction of connected components of the meta-graph H_F are trees. Note that an isolated vertex is a tree as well.

In the sequel, we show that each tree-component of H_F contains at least one vertex corresponding to an arborescence from which any bounce on a failed edge leads to the destination d without hitting any new failed edge. To that end, we introduce the notion of good arcs and good arborescences. We say that an arc (u,v) is a *good arc* of an arborescence T if on the (unique) v-d path in T there is no failed edge. Let $a=(i,j)$, for $i \neq j$, be an arc of $\vec{H}_F$, $\{u,v\}$ be the edge that corresponds to a, and w.l.o.g. assume (u,v) is an arc of T_j. Then, we say a is a *well-bouncing* arc if (u,v) is a good arc of T_j. Intuitively, a well-bouncing arc (i,j) of $\vec{H}_F$ means that by bouncing from T_i to T_j on the failed edge $\{v,u\}$ the packet will reach d via routing along T_j without any further interruption. Finally, we say that an arborescence T_i is a *good arborescence* if every outgoing arc of vertex $i \in V_F$ is well-bouncing.

Lemma 4.3. *Let T be a tree-component of H_F s.t. $|V(T)|>1$. Then, $\vec{T}$ contains at least $|V(T)|$ well-bouncing arcs.*

Proof. Let T_i be an arborescence of $\mathcal{T}$ such that $i \in V(T)$. Then, by the construction of H_F we have that T_i contains a failed link. Next, a failed link closest to the root of T_i is a good arc of T_i. Therefore, for every $i \in V(T)$, we have that T_i contains an arc which is both good and failed. Furthermore, by the construction of H_F and the definition of well-bouncing arcs, we have that for every good, failed link of T_i there is the corresponding well-bouncing arc of $\vec{T}$. Also, observe that the construction of H_F implies that a well-bouncing arc corresponds to exactly one good-arc.

Now, putting all the observations together, we have that each T_i, for every $i \in V(T)$, has a good failed link which further corresponds to a well-bouncing arc of $\vec{T}$. As all the arborescences are arc-disjoint, and there are $|V(T)|$ many of them represented by the vertices of T, we have that $\vec{T}$ contains at least $|V(T)|$ well-bouncing arcs. □

Now, building on Lemma 4.3, we prove the following.

Lemma 4.4. *Let T be a tree-component of H_F. Then, there is an arborescence T_i such that $i \in V(T)$ and T_i is good.*

Proof. Consider two cases: $|V(T)|=1$, and $|V(T)|>1$. In the case $|V(T)|=1$, T is an isolated vertex which implies that it has no outgoing arcs. Therefore, T represents a good arborescence.

If $|V(T)|>1$, then from Lemma 4.3 we have that $\vec{T}$ contains at most $2(|V(T)|-1)-|V(T)|<|V(T)|$ arcs which are not well-bouncing. This implies that there is at least one vertex in T from which every outgoing arc is well-bouncing. □

Let us understand what this implies. Consider an arborescence T_i, and a routing of a packet along it. In addition, assume that the routing hits a failed edge e, such that e is shared with some other arborescence T_j. Now, if e corresponds to a well-bouncing arc of $\vec{H}_F$, then by bouncing on e and routing solely along T_j, the packet will reach d without any further interruption. Lemma 4.4 claims that for each tree-component T of H_F there always exists an arborescence T_i, with $i \in V(T)$, which is good, i.e. every failed edge of T_i corresponds to a well-bouncing arc of $\vec{H}_F$.

We can now state the main lemma of this section.

Lemma 4.5. *If G contains at most $k-1$ failed edges, then $\mathcal{T}$ contains at least one good arborescence.*

Proof. We prove that there exists an arborescence T_i such that if a packet bounces on any failed edge of T_i it will reach d without any further interruption. Let F be the set of failed edges, at most $k-1$ of them. Then, by Lemma 4.2 we have that H_F contains at least $k-f \geq 1$ tree-components. Let T be one such component.

By Lemma 4.4, we have that there exists at least an arborescence T_i such that every outgoing arc from i is well-bouncing. Therefore, bouncing on any failed arc of T_i the packet will reach d without any further interruption. □

4.5 Randomized Routing via Good Arborescences

In this section, we show that a set of routing functions for G obtained by RAND-BOUNCING-ALGO is $(k-1)$-resilient. Note that our routing function (`RND`) maps an incoming edge and the set of active edges incident at v to a set of pairs (e,q), where e is an outgoing edge and q is the probability of forwarding a packet through e. A packet is forwarded through a unique outgoing edge.

The section is structured as follows. As a prelude, we show a simple, yet inefficient, randomized routing algorithm, called RAND-ALGO, that although is $(k-1)$-resilient, fails to achieve low expected number of hops in case of $k-1$ failed edges. We then apply our results from Section 4.4 to show that RAND-BOUNCING-ALGO is both $(k-1)$-resilient and requires up to an order fewer number of hops, compared to RAND-ALGO, to reach the destination.

4.5.1 A Simple (Inefficient) Randomized Routing

Consider the following randomized algorithm RAND-ALGO for routing along arborescences. A packet is routed along the same arborescence until it either reaches its destination or hits a failed edge. In the latter case, it is rerouted along another arborescences chosen uniformly at random. We first show that there exists a k-connected graph and a set of failed edges such that the expected number of tree switches that RAND-ALGO makes is $\Omega(k^2)$. We then exhibit an instance of k-connected graph and a set of failed edges for which RAND-ALGO makes $\Omega(|V|k^2)$ hops. This number of hops is at least k times larger than $O(kH)$, that is guaranteed by Theorem 4.1.

RAND-ALGO Performs $\Omega(k^2)$ Tree Switches

To prove the promised bound, we start by defining a $2k$ edge connected graph $G=(V,E)$ and its set of $2k$ arc disjoint spanning trees $T_0,\ldots,T_{2k-1}$ as follows.

- Set V consists of a destination vertex d and $4k$ additional vertices arranged into two equal-sized layers $L_1=\{v_0^1,\ldots,v_{2k-1}^1\}$ and $L_2=\{v_0^2,\ldots,v_{2k-1}^2\}$.
- Set E is defined by the following four subgraphs: (1) L_2 is a clique of size $2k$; (2) (L_1,L_2) is a complete bipartite graph; (3) for each $k=0,\ldots,k-1$, there is an edge (v_{2i}^1, v_{2i+1}^1) and (4) vertex d is connected to each vertex of L_1. There is no other edge included in G.

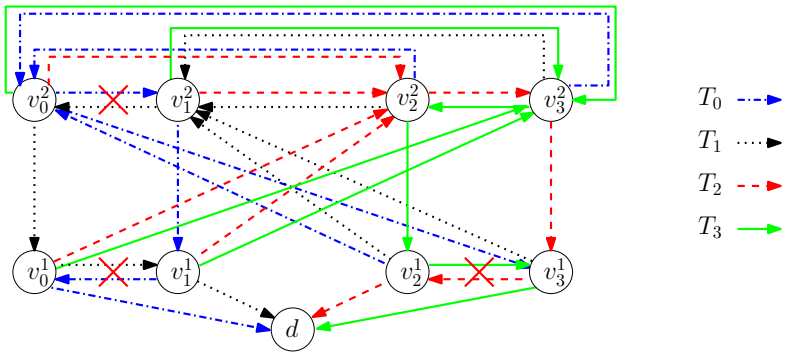

Figure 4.2 – Graph used in the proof of Theorem 4.8 for $k = 2$.

Next, we construct $2k$ arc-disjoint spanning trees $T_0, \ldots, T_{2k-1}$ (see Figure 4.2 for an example with $k = 2$). We use $[t]_0$ to denote set $\{0, 1, 2, \ldots, t-1\}$.

- For each $i \in [k]_0$, add the following arcs:
  - (v_{2i+1}^2, v_{2i}^2), (v_{2i}^2, v_{2i}^1), (v_{2i}^1, v_{2i+1}^1), and (v_{2i+1}^1, d) into T_{2i+1};
  - arcs (v_{2i}^2, v_{2i+1}^2), (v_{2i+1}^2, v_{2i+1}^1), (v_{2i+1}^1, v_{2i}^1), and (v_{2i}^1, d) into T_{2i}.
- For each $i \in [k]_0$, and for each $j \in [2k]_0 \setminus \{2i, 2i+1\}$, add the following arcs:
  - (v_j^2, v_{2i}^2), and (v_j^1, v_{2i}^2) into T_{2i};
  - (v_j^2, v_{2i+1}^2), and (v_j^1, v_{2i+1}^2) into T_{2i+1}.

Finally, consider a scenario in which edges (v_0^2, v_1^2), (v_2^2, v_3^2), $\ldots$, (v_{2k-4}^2, v_{2k-3}^2) and (v_0^1, v_1^1), (v_2^1, v_3^1), $\ldots$, (v_{2k-4}^1, v_{2k-3}^1), (v_{2k-2}^1, v_{2k-1}^1) failed.

We say that a packet is routed *downwards* (*upwards*) if it is routed from a vertex in L_2 (L_1) to a vertex in L_1 (L_2). Let E_d be the expected number, minimized over all the vertices, of tree switches of a packet that is routed downwards, E_u be the expected number of tree switches of a packet that is routed along T_i and is currently located at v_i^2, for some $i \in [2k-2]_0$, and E_2 be the expected number of tree switches of a packet that is originated by a vertex in L_2. Then, we can show.

Lemma 4.6. *It holds* $E_u \geq \frac{3}{2k-1} E_d + \frac{2k-4}{2k-1} E_u + 1$.

Proof. Let p be routed along T_h and located at v_h^2, for some $h \in [2k-2]_0$. W.l.o.g, let $h = 0$. By the construction of T_0, from v_0^1 packet p should be forwarded to v_1^2 but (v_0^2, v_1^2) has failed. So, from v_0^2, p is forwarded downwards along T_1, T_{2k-2} or T_{2k-1} with probability $\frac{3}{2k-1}$ and routed along any other tree T_j to a vertex v_j^2 in L_2 with probability at least $\frac{2k-4}{2k-1}$. Hence, the lemma follows. □

Lemma 4.7. *We have* $E_d \in \Omega(k^2)$.

Proof. By the construction, a packet routed downwards traverses arc (v_i^2, v_i^1) of T_i. W.l.o.g, let p be routed along (v_{2i}^2, v_{2i}^1) of T_{2i}. As (v_{2i}^1, v_{2i+1}^1), which belongs to T_{2i}, has failed, p is rerouted along T_j for some $j \in [2k]_0 \setminus \{2i\}$. Among them, only T_{2i+1} has a path from v_{2i}^1 to d that does

not contain any failed link. T_{2i+1} is chosen with probability $\frac{1}{2k-1}$.
If any other tree T_j is chosen except T_{2k-2} and T_{2k-1}, which happens with probability $\frac{2k-4}{2k-1}$, then p is rerouted through T_j from v_{2i}^1 to a vertex v_j^2 in L_2, and hence

$$E_d \geq \frac{2k-4}{2k-1} E_u + 1. \tag{4.4}$$

Putting together with (4.4) and Lemma 4.6 we obtain $E_d \in \Omega(k^2)$. □

We finally observe that any packet originated at a vertex of L_2 is routed downwards at least once before reaching the destination vertex, i.e., $E_2 \geq E_d = \Omega(k^2)$, which proves the following theorem.

Theorem 4.8

For any $k > 0$, there exists a $2k$ edge-connected graph, a set of $2k$ arc-disjoint spanning trees, and a set of $2k-1$ failed edges, such that the expected number of tree switches with RAND-ALGO is $\Omega(k^2)$.

RAND-ALGO Performs $\Omega(|V|k^2)$ Hops

We now a more involved construction than the one in Theorem 4.8 that shows that there are examples for which if we apply only bouncing, in addition to the number of tree switches, they have big stretch.

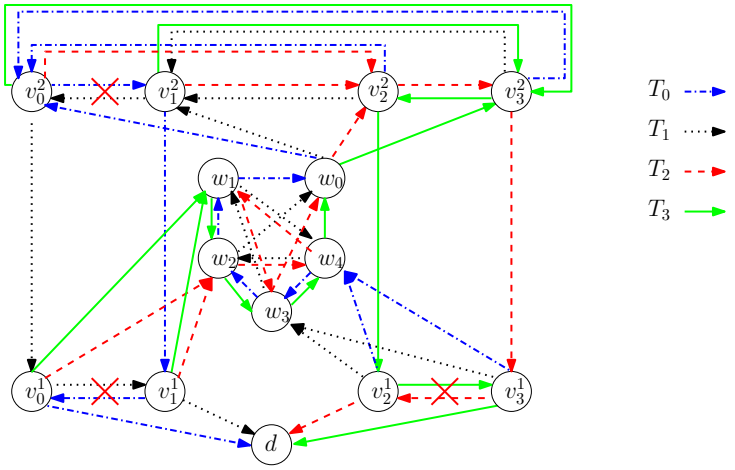

Figure 4.3 – Graph used in the proof of Theorem 4.9 for $k = 2$ and $|V| = 5$.

Theorem 4.9

For any $k > 0$, there exists a $2k$ edge-connected graph $G = (V, E)$, a set of $2k$ arc-disjoint spanning trees, and a set of $k-1$ failed edges, such that the expected number of tree switches with RAND-ALGO is $\Omega(k^2)$. Furthermore, the routing makes $\Omega(k^2|V|)$ hops in expectation.

Proof. To prove the promised bound, we start by defining a $2k$ edge connected graph $G = (V, E)$ and its set of $2k$ arc disjoint spanning trees $T_0, \ldots, T_{2k-1}$ as follows.

- Set V consists of a destination vertex d and $4k + p$ additional vertices arranged into three layers L_1, L_2, and W.
- Layers $L_1 = \{v_0^1, \ldots, v_{2k-1}^1\}$ and $L_2 = \{v_0^2, \ldots, v_{2k-1}^2\}$ are equal-sized.
- Layer $W = \{w_0, w_1, \ldots, w_{p-1}\}$ is placed "in between" L_1 and L_2. Number p is a prime such that $\max\{|V|, 2k+1\} \leq p \leq 2\max\{|V|, 2k+1\}$. Note that such p always exist.
- Set E is defined to be the edge support of the arborescences that we define in the sequel. Other than that, G does not contain additional edges.

Next, we construct $2k$ arc-disjoint spanning trees $T_0, \ldots, T_{2k-1}$ (see Figure 4.3 for an example with $k = 2$ and $|V| = 5$). We use $[t]_0$ to denote set $\{0, 1, 2, \ldots, t-1\}$.

- For each $i \in [k]_0$, add the following arcs:
  - (v_{2i+1}^2, v_{2i}^2), (v_{2i}^2, v_{2i}^1), (v_{2i}^1, v_{2i+1}^1), and (v_{2i+1}^1, d) into T_{2i+1};
  - arcs (v_{2i}^2, v_{2i+1}^2), (v_{2i+1}^2, v_{2i+1}^1), (v_{2i+1}^1, v_{2i}^1), and (v_{2i}^1, d) into T_{2i}.
- For each $i \in [k]_0$, and for each $j \in [2k]_0 \setminus \{2i, 2i+1\}$, add the following arcs:
  - (v_j^2, v_{2i}^2), $(v_j^1, w_{(2i+1)(p-1) \mod p})$, and (w_0, v_{2i}^2) into T_{2i};
  - (v_j^2, v_{2i+1}^2), $(v_j^1, w_{(2i+2)(p-1) \mod p})$, and (w_0, v_{2i+1}^2) into T_{2i+1}.
- For each $i \in [2k]_0$ and each $a \in [p-1]_0$ add arc $(w_{(i+1)(a+1) \mod p}, w_{(i+1)a \mod p})$ to T_i.

Finally, consider a scenario in which edges (v_0^2, v_1^2), (v_2^2, v_3^2), $\ldots$, (v_{2k-4}^2, v_{2k-3}^2) and (v_0^1, v_1^1), (v_2^1, v_3^1), $\ldots$, (v_{2k-4}^1, v_{2k-3}^1), (v_{2k-2}^1, v_{2k-1}^1) failed.

Since p is a prime, it is easy to show that the described arborescences are valid and arc-disjoint.

For each $a = 1, \ldots, 2k$ the i-th vertex of the vertex-cloud in the middle layer arborescence a walks over has index $x_i^a = a \cdot (i-1) \mod p$. In order to show that this vertex ordering can indeed be part of $2k$ arc-disjoint arborescences we will prove that: $x_i^a \neq x_j^a$ whenever $i \neq j$; and, (x_i^a, x_{i+1}^a) is different than (x_j^b, x_{j+1}^b) for any $a \neq b$, and any valid i and j. These claims follow from the fact that $\gcd(p, i) = 1$, but for completeness we provide short proofs.

Towards a contradiction, assume that $x_i^a = x_j^a$ for some $i \neq j$. Furthermore, by the definition, that implies $a \cdot (i-1) \equiv a \cdot (j-1) \mod p$, and hence $a(i-j) \equiv 0 \mod p$. However, as $0 \leq a, |i-j| < p$ and p is a prime, $a(i-j)$ is not divisible by p and hence a contradiction.

Again towards a contradiction, assume that $(x_i^a, x_{i+1}^a) = (x_j^b, x_{j+1}^b)$ for some $a \neq b$, and some valid i and j. Then, from $x_i^a = x_j^b$ we have

$$ai \equiv bj \mod p. \tag{4.5}$$

On the other hand, $x_{i+1}^a = x_{j+1}^b$ implies $a(i+1) \equiv b(j+1) \mod p$, which can be written as

$$ai + a \equiv bj + b \mod p. \tag{4.6}$$

Putting together (4.5) and (4.6) we obtain $a \equiv b \mod p$, which contradicts the fact that $a \neq b$ and $1 \leq a, b \leq k$.

Observe that whenever packet reaches vertex of W, it visits all the vertices of W before leaving that layer. Then, the rest of the proof, i.e., computing the number of expected hops and stretch, is analogous to the proof of Theorem 4.8.

This concludes the analysis. □

4.5.2 Correctness of Randomized-Bouncing Routing

In this section we prove that RAND-BOUNCING-ALGO eventually delivers a packet to d, i.e. it avoids loops, and in the next section we analyze its efficiency.
Assume that we, magically, know whether the arborescence we are routing along is a good one or not. Then, on a failed edge we could bounce if the arborescence is good, or switch to the next arborescence otherwise. And, we would not even need any randomness. However, we do not really know whether an arborescence is good or not since we do not know which edges will fail. To alleviate this lack of information we use a random guess. So, each time we hit a failed edge we take a guess that the arborescence is good, where the parameter q estimates this likelihood. Notice that RAND-BOUNCING-ALGO implements exactly this approach. As an example, consider Fig. 4.1. If a packet originated at a is first routed through `Red` and the corresponding outgoing edge $\{a, c\}$ is failed, then the packet is forwarded with probability q to `Blue` or `Green` chosen u.a.r., and with probability $1-q$ it is bounced to `Green`, which shares the outgoing failed edge $\{a, c\}$ with `Red`. By the following lemma we show that this approach leads to $(k-1)$-resilient routing.

Lemma 4.10. RAND-BOUNCING-ALGO *produces a set of* $(k-1)$*-resilient routing functions.*

Proof. By Lemma 4.5 we have that there exists at least one arborescence T_i of $\mathcal{T}$ such that bouncing on any failed edge of T_i the packet will reach d without any further interruption. Now, as on a failed edge algorithm RAND-BOUNCING-ALGO will switch to T_i with positive probability, and on a failed edge of T_i the algorithm will bounce with positive probability, we have that the algorithm will eventually reach d. □

4.5.3 Number of Switches of RAND-BOUNCING-ALGO

In this subsection we analyze the expected number of times I the packet is rerouted from one arborescence to another one in RAND-BOUNCING-ALGO. As we are interested in providing an upper bound on I, we make the following assumptions. First, we assume that bouncing from an arborescence which is not good the routing always bounces to an arborescence which is not good as well. Second, we assume that only by bouncing from a good arborescence the routing will reach d without switching to any other arborescence. Third, we assume that there are exactly $k-f$ good arborescences, which is the lower bound provided by Lemma 4.2 and Lemma 4.4. Clearly, these assumptions can only lead to an increased number of iterations compared to the real case. Finally, for the sake of brevity we define $t \stackrel{\text{def}}{=} \frac{f}{k}$.

Now, we are ready to start with the analysis. As the first step we define a random variable, where in the definitions T is the arborescence variable from algorithm RAND-BOUNCING-

ALGO,

$$X \stackrel{\text{def}}{=} \text{number of times a failed edge is hit before reaching } d \text{ if routing on } T$$

Let T_{init} be the first arborescence that we consider in RAND-BOUNCING-ALGO. Then, $\mathbb{E}[I]$ is upper-bounded by

$$\mathbb{E}[I] \leq \Pr\left[T_{init} \text{ is not good}\right]\mathbb{E}\left[X|T_{init} \text{ is not good}\right] + \Pr\left[T_{init} \text{ is good}\right]\mathbb{E}\left[X|T_{init} \text{ is good}\right], \quad (4.7)$$

where from our assumptions we have

$$\Pr\left[T_{init} \text{ is not good}\right] = t, \text{ and } \Pr\left[T_{init} \text{ is good}\right] = 1-t.$$

To simplify calculations, let X_P and Y_P be *pessimistic* upper bound on conditional expected values. That is, let X_P be the same as $\mathbb{E}\left[X|T_{init} \text{ is not good}\right]$ and Y_P as $\mathbb{E}\left[X|T_{init} \text{ is good}\right]$ under assumption that: the packet always hits a failed edge unless it bounces on a good arborescence; and, whenever packet bounces on a non-good arborescence it switches to a non-good one.

Now, let us express X_P and Y_P as functions in X_P, Y_P, q, and t, while following our assumptions. If T is not a good arborescence, then a routing along T will hit a failed edge. If it hits a failed edge, with probability $1-q$ the routing will bounce and switch to a non good arborescence. With probability qt the routing scheme will set T to be a non good arborescence, and with probability $q(1-t)$ it will set T to be a good arborescence. Formally, we have

$$X_P = 1 + qtX_P + q(1-t)Y_P + (1-q)X_P. \quad (4.8)$$

Applying an analogous reasoning about Y_P, we obtain

$$Y_P = 1 + qtX_P + q(1-t)Y_P. \quad (4.9)$$

Observe that the equations describing X_P and Y_P differ only in the term $(1-q)X_P$. This comes from the fact that bouncing on a good arborescences the packet will reach d without hitting any other failed edge.

By some simple calculations (see [CNP+14]), we obtain

$$\mathbb{E}[I] \leq U(q) \stackrel{\text{def}}{=} \frac{t}{(1-q)q(1-t)} + \frac{1}{1-q}. \quad (4.10)$$

Now we can prove the following lemma.

Lemma 4.11. *We have that*

$$\mathbb{E}[I] \leq 2 + 4\frac{t}{1-t} = 2 + 4\frac{f}{k-f}.$$

Proof. From (4.10) we have $\mathbb{E}[I] \leq U(q)$. Setting $q = 1/2$ we obtain

$$U(1/2) \leq 2 + 4\frac{t}{1-t},$$

and by plugging $t = f/k$ the lemma follows. □

Note that if we know f in advance, or have some guarantee in terms of an upper bound on f, we can derive parameter q that improves the running time of RAND-BOUNCING-ALGO, as provided by the following lemma.

Lemma 4.12. *$U(q)$ is minimized for $q = q^\star = 1-(1+\sqrt{t})^{-1}$, and equal to*

$$U(q^\star) = \frac{1+\sqrt{t}}{1-\sqrt{t}}. \tag{4.11}$$

Proof. Consider $U(q)'$, which is

$$U(q)' = \frac{t(1-q)^2 - q^2}{(1-q)^2 q^2 (t-1)}.$$

In order to find the value of q that minimizes $U(q)$, denote it by $q^\star$, we find the roots of $U(q)' = 0$ with respect to q. There is only one positive solution of equation $U(q)' = 0$, which is also the minimizer $q^\star$, and is equal to $q^\star = 1 - \frac{1}{1+\sqrt{t}}$, as desired.

Finally, substituting $q^\star$ into (4.10) and simplifying the expression we obtain (4.11). □

Observe that

$$U(q^\star) \leq \frac{4}{1-\frac{f}{k}}.$$

Therefore, if $f = \alpha k$, i.e., only a fraction of the edges fail, we obtain $U(q^\star) \leq \frac{4}{1-\alpha}$. This means that the expected number of arborescence switches does not depend on the number of failed edges but on the ratio between this number and the connectivity of the graph. Otherwise, if $f = k-1$, we have that the expected number of arborescence switches is bounded by $4k$, which is linear w.r.t. to the connectivity of the graph. Combining these conclusions with Lemma 4.10 the proof of Theorem 4.1 follows.

4.5.4 Extension : Rerouting in a Non-uniform Manner

In this section we briefly study non-uniform choice of arborescence used for rerouting in algorithm RAND-BOUNCING-ALGO. To motivate that discussion, consider a scenario in which a packet hits a failed edge u, v while routed along arborescence T. Wlog, assume $T = T_k$. Furthermore, assume that path v-d along every other arborescence does not contain any failed link. Therefore, switching from T_k to any other arborescence the packet will reach d without any further interruption. If the packet is rerouted at step 2.2.(a) of algorithm RAND-BOUNCING-ALGO but not bounced, then the rerouting tree is chosen uniformly at random. It further means that the expected number of edges the packet will traverse before reaching d from v is

$$E_U = \sum_{i=1}^{k-1} \frac{dist_{T_i}(v)}{k-1} = \frac{\sum_{i=1}^{k-1} dist_{T_i}(v)}{k-1},$$

where $dist_{T_i}(a)$ is the number of the edges on the unique path from a to d along arborescence T_i.[1] However, the distances from v to d along different arborescences might significantly

[1] As a remark, the best option in this scenario would be to reroute the packet along the arborescence T_i such that $i = \arg\min_i dist_{T_i}(v)$. Unfortunately, our model does not provide the information whether there is any failed edge on path v-d along T_i or not.

differ. This naturally suggests us to consider a non-uniform distribution of arborescences chosen at step 2.2.(a) of RAND-BOUNCING-ALGO, as we do in the rest of this section.

For each vertex $v \neq d$ and each arborescence T_i define probability p_v^i as

$$p_v^i \stackrel{\text{def}}{=} \frac{\frac{1}{dist_{T_i}(v)}}{\sum_{j=1}^{k-1} \frac{1}{dist_{T_j}(v)}}.$$

The expected number of the edges the packet will traverse if each arborescence is chosen with respect to the distribution given by p_v is

$$E_{NU} = \sum_{i=1}^{k-1} p_v^i dist_{T_i}(v) = \sum_{i=1}^{k-1} \frac{1}{\sum_{j=1}^{k-1} \frac{1}{dist_{T_j}(v)}} = \frac{k-1}{\sum_{i=1}^{k-1} \frac{1}{dist_{T_i}(v)}}.$$

Now we would like to show that indeed $E_U \stackrel{?}{\geq} E_{NU}$. But, it is the same as showing that

$$(k-1)^2 \stackrel{?}{\leq} \sum_{i=1}^{k-1} dist_{T_i}(v) \sum_{i=1}^{k-1} \frac{1}{dist_{T_i}(v)}.$$

However, the latter follows from Cauchy–Schwarz inequality as

$$(k-1)^2 = \left(\sum_{i=1}^{k-1} \sqrt{dist_{T_i}(v)} \sqrt{\frac{1}{dist_{T_i}(v)}} \right)^2 \leq \sum_{i=1}^{k-1} dist_{T_i}(v) \sum_{i=1}^{k-1} \frac{1}{dist_{T_i}(v)}.$$

Hence, $E_U \geq E_{NU}$, as advertised.

5 Streaming Submodular Maximization under Element Removals

This chapter is based on a joint work with Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Volkan Cevher. It has been accepted to the 31th Conference on Neural Information Processing Systems (**NIPS**) 2017 [MBNF+17] under the title

Streaming Robust Submodular Maximization: A Partitioned Thresholding Approach.

5.1 Introduction

A central challenge in many large-scale machine learning tasks is data summarization – the extraction of a small representative subset out of a large dataset. Applications include image and document summarization [TIWB14, LB11], influence maximization [KKT03], facility location [LWD16], exemplar-based clustering [KG10], recommender systems [EAG11], and many more. Data summarization can often be formulated as the problem of maximizing a *submodular* set function subject to a cardinality constraint.

On small datasets, this submodular maximization problem is solved by a greedy method [NWF78], that produces solutions provably close to optimal. However, this algorithm requires repeated access to all elements, which makes it infeasible for large-scale scenarios, where the entire dataset does not fit in the main memory. In this setting, streaming algorithms prove to be useful, as they make only a small number of passes over the data and use sublinear space.

In many settings, the extracted representative set is also required to be *robust*. That is, the objective value should degrade as little as possible when some elements of the set are removed. Such removals may arise in many applications. For instance, imagine a scenario in which a user requests a personalized recommendation of books with the goal to buy a new book. Ideally, such recommendation list should have the following two properties. First, naturally, the recommended list should represent the user's interest. Second, the list should be robust to the books that the user have already read. Namely, even when the books that the user have already read are removed from the list, the remaining books in the list should capture the user's interest.

A robustness requirement is especially challenging for large datasets, where it is prohibitively expensive to reoptimize over the entire data collection in order to find replacements for the removed elements. In some applications, where data is produced so rapidly that most of it is not being stored, such a search for replacements may not be possible at all.

5.1.1 Problem Setup

We consider a potentially large universe of elements V of size n equipped with a *normalized monotone submodular* set function $f : 2^V \to \mathbb{R}_{\geq 0}$ defined on V. It is said that f is *monotone* if for any two sets $X \subseteq Y \subseteq V$ we have $f(X) \leq f(Y)$. The set function f is *submodular* if for any two sets $X \subseteq Y \subseteq V$ and any element $e \in V \setminus Y$ it holds that

$$f(X \cup \{e\}) - f(X) \geq f(Y \cup \{e\}) - f(Y).$$

We use $f(Y \mid X)$ to denote the marginal gain in the function value due to adding the elements of set Y to set X, i.e. $f(Y \mid X) \stackrel{\text{def}}{=} f(X \cup Y) - f(X)$. We say that f is *normalized* if $f(\emptyset) = 0$.

The problem of maximizing a monotone submodular function subject to a cardinality constraint, i.e.,

$$\max_{Z \subseteq V, |Z| \leq k} f(Z), \tag{5.1}$$

has been studied extensively. It is well-known that a simple greedy algorithm (referred to as GREEDY) [NWF78] provides a $(1 - e^{-1})$-approximation. However, it requires repeated access to all the elements of a dataset, which precludes it from use in large-scale tasks.

We say that a set S is *robust* for a parameter m if, for any set $E \subseteq V$ such that $|E| \leq m$, there is a subset $Z \subseteq S \setminus E$ of size at most k such that

$$f(Z) \geq c f(\mathrm{OPT}(k, V \setminus E)),$$

where $c > 0$ is an approximation ratio. We use $\mathrm{OPT}(k, V \setminus E)$ to denote the optimal subset of size k of $V \setminus E$. Formally, $\mathrm{OPT}(k, V \setminus E)$ is defined as

$$\mathrm{OPT}(k, V \setminus E) \in \operatorname*{arg\,max}_{Z \subseteq V \setminus E, |Z| \leq k} f(Z).$$

In this work, we are interested in solving a robust version of Problem (5.1) in the streaming setting.

5.1.2 Our Results

We propose a two-stage procedure for robust submodular maximization. For the first stage, we design a streaming algorithm which makes one pass over the data and finds a summary that is robust against removal of up to m elements, while containing at most $O\left((m \log k + k) \log^2 k\right)$ elements.

In the second stage, given any set of size m that has been removed from the obtained summary, we run the standard greedy algorithm on the remaining elements to obtain a set of size at most k. We show that this set has large value as well. More precisely, we prove the following.

Theorem 5.1

For any given constant $\epsilon > 0$, there exists a single-pass streaming algorithm that under at most m removals from the stream collects $O\left((k + m \log k) \log^2 k\right)$ elements and outputs

set Z of cardinality at most k such that

$$f(Z) \geq \frac{0.149}{1+\epsilon}\left(1 - \frac{1}{\lceil \log k \rceil}\right) f(\mathrm{OPT}(k, V \setminus E)).$$

5.1.3 Related Work

A robust, non-streaming version of Problem (5.1) was first introduced in [KMGG08]. In that setting, the algorithm must output a set Z of size k which maximizes the smallest objective value guaranteed to be obtained after a set of size m is removed, that is,

$$\max_{Z \subseteq V, |Z| \leq k} \min_{E \subseteq Z, |E| \leq m} f(Z \setminus E).$$

The work [OSU16] provides the first constant (0.387) factor approximation result to this problem, valid for $m = o(\sqrt{k})$. Their solution consists of buckets of size $O(m^2 \log k)$ that are constructed greedily, one after another. Recently, in [BMSC17], a centralized algorithm has been proposed that achieves the same approximation result and allows for a greater robustness $m = o(k)$. This algorithm constructs a set that is arranged into partitions consisting of buckets whose sizes increase exponentially with the partition index. In this work, we use a similar structure for the robust set but, instead of filling the buckets greedily one after another, we place an element in the first bucket for which the gain of adding the element is above the corresponding threshold. Moreover, we introduce a novel analysis that allows us to be robust to any number of removals m as long as we are allowed to use $O(m \log^2 k)$ memory.

Submodular streaming algorithms (e.g. [KG10], [KMVV15] and [NFBEH+16]) have become a prominent option for scaling submodular optimization to large-scale machine learning applications. A popular submodular streaming algorithm SIEVE-STREAMING [BMKK14] solves Problem (5.1) by performing one pass over the data, and achieves a $(0.5-\epsilon)$-approximation while storing at most $O\left(\frac{k \log k}{\epsilon}\right)$ elements.

Recent progress. Independently of our work, [MKK17] gave a streaming algorithm for robust submodular maximization subject to the cardinality constraint. Their approach also provides a constant-factor (i.e. $1/2 - \varepsilon$) approximation guarantee. However, their algorithm uses $O(mk \log k/\varepsilon)$ memory.

5.2 Organization

We begin by presenting our main algorithm in Section 5.3. In Section 5.4, we prove that our algorithm provides a constant-factor approximation guarantee if the value of $f(\mathrm{OPT}(k, V \setminus E))$ is known. By extending standard techniques, in Section 5.5, we show how to remove this assumption.

5.3 Main Algorithm

We design a streaming algorithm *Streaming Robust submodular algorithm with Partitioned Thresholding* (Algorithm 9), that we also refer to by STAR-T. This algorithm is used to select

the elements from the stream, while Algorithm 10, which we call STAR-T-GREEDY, is used to output the set of elements after the stream is over. In the algorithm and throughout the proofs, we use a multiplicative factor w that we define in Table 5.1.

The multiplicative factor for bucket size:

$$w \stackrel{\text{def}}{=} \left\lceil \frac{4\lceil \log k \rceil m}{k} \right\rceil .$$

Table 5.1 – The parameter for scaling bucket sizes that is used in the algorithm and in the proofs.

Algorithm 9: STAR-T (V, k, τ)

Input:
- The ground set V provided in the streaming fashion
- Cardinality constraint k
- τ – the threshold value shared by all the buckets

Output: A robust summary S

```
1 B_{i,j} ← ∅, for all 0 ≤ i ≤ ⌈log k⌉ and 1 ≤ j ≤ w⌈k/2^i⌉
2 for element e from the stream do
      /* loop over partitions */
3     for i ← 0 to ⌈log k⌉ do
          /* loop over buckets */
4         for j ← 1 to w⌈k/2^i⌉ do
5             if |B_{i,j}| < min{2^i, k} and f(e | B_{i,j}) ≥ τ / min{2^i, k} then
6                 B_{i,j} ← B_{i,j} ∪ {e}
7                 proceed to the next element in the stream
8 return S ← ⋃_{i,j} B_{i,j}
```

As the input, STAR-T requires the access to the ground set V, a cardinality constraint k, and a threshold parameter τ. Think of the parameter τ is an α-approximation to $f(\mathrm{OPT}(k, V \setminus E))$, for some $\alpha \in (0,1]$ to be specified later. Hence, τ depends on $f(\mathrm{OPT}(k, V \setminus E))$, that is not known a priori. For the sake of clarity, we present the algorithm as if $f(\mathrm{OPT}(k, V \setminus E))$ is known, and in Section 5.5 we show how remove this assumption. The algorithm makes one pass over the data and outputs a set of elements S that is later used by STAR-T-GREEDY to output the final set.

The set S is divided into $\lceil \log k \rceil + 1$ partitions, where every partition $i \in \{0, \ldots, \lceil \log k \rceil\}$ consists of $w\lceil k/2^i \rceil$ buckets $B_{i,j}$, $j \in \{1, \ldots, w\lceil k/2^i \rceil\}$. Every bucket $B_{i,j}$ stores at most $\min\{k, 2^i\}$ elements. If $|B_{i,j}| = \min\{2^i, k\}$, then we say that $B_{i,j}$ is *full*.

Every partition has a corresponding threshold that is exponentially decreasing with the partition index i as $\tau/2^i$. For example, the buckets in the first partition will only store elements that have marginal value at least τ. Every element $e \in V$ arriving on the stream is assigned to the first non-full bucket $B_{i,j}$ for which the marginal value $f\left(e \,\middle|\, B_{i,j}\right)$ is at least $\tau/2^i$. If there is no such bucket, the element will not be stored. Hence, the buckets are disjoint sets that in the

end (after one pass over the data) can have a smaller number of elements than specified by their corresponding cardinality constraints, and some of them might even be empty. The set S returned by STAR-T is the union of all the buckets.

Algorithm 10: STAR-T- GREEDY

Input:
- A robust summary S
- Cardinality constraint k
- The set of removed elements E

Output: A subset of $S \setminus E$ of cardinality k

1 **return** $Z \leftarrow \text{GREEDY}(k, S \setminus E)$

In the second stage, STAR-T-GREEDY receives as input the set S constructed in the streaming stage, a set $E \subset S$ that we think of as removed elements, and the cardinality constraint k. The algorithm then returns a set Z, of size at most k, that is obtained by running the simple greedy algorithm GREEDY on the set $S \setminus E$. Note that STAR-T-GREEDY can be invoked for different sets E.

5.4 Approximation with the Access to the Optimum Value

In this section we present our results in detail. We initially assume that the value $f(\text{OPT}(k, V \setminus E))$ is known; later, in Section 5.5, we remove this assumption. We begin by stating the main result.

Theorem 5.2

Let f be a normalized monotone submodular function defined over the ground set V. Define α as

$$\alpha \stackrel{\text{def}}{=} \frac{1}{2 + \frac{(1-e^{-1})}{(1-e^{-1/3})}\left(1 - \frac{4m}{wk}\right)}.$$

Then, given a cardinality constraint k and an upper-bound m on $|E|$, the algorithm STAR-T performs a single pass over the dataset and constructs a set S of at most $O((k + m\log k)\log k)$ elements.

Furthermore, if $\tau = \alpha f(\text{OPT}(k, V \setminus E))$, then the set S is such that STAR-T-GREEDY outputs a set $Z \subseteq S \setminus E$ of size at most k such that

$$f(Z) \geq 0.149\left(1 - \frac{1}{\lceil \log k \rceil}\right) \cdot f(\text{OPT}(k, V \setminus E)).$$

5.4.1 Proof Overview

To prove the theorem, we consider three cases.

We first consider the case when there is a partition $i_\star$ in S such that at least half of its buckets are full. We show that then there is at least one full bucket $B_{i_\star,j}$ such that $f\left(B_{i_\star,j} \setminus E\right)$ is only a constant factor smaller than $f(\text{OPT}(k, V \setminus E))$, as long as the threshold τ is set close

to $f(\mathrm{OPT}(k, V \setminus E))$. We make this statement precise in the following lemma, whose proof is deferred to Section 5.4.2.

Lemma 5.3. *If there exists a partition in S such that at least half of its buckets are full, then for the set Z produced by* STAR-T-GREEDY *we have*

$$f(Z) \geq \left(1 - e^{-1}\right)\left(1 - \frac{4m}{wk}\right)\tau. \tag{5.2}$$

Next, we consider the other case, i.e., when for every partition, more than half of its buckets are not full after the execution of STAR-T. For every partition i, we let B_i denote a bucket that is not fully populated and for which $|B_i \cap E|$ is minimized over all the buckets of that partition. Then, we look at such a bucket in the last partition, denoted by $B_{\lceil \log k \rceil}$.

We provide two lemmas that depend on $f(B_{\lceil \log k \rceil})$. If the threshold parameter τ is set to be as in the statement of Theorem 5.2, then:

- Lemma 5.4 shows that if $f(B_{\lceil \log k \rceil})$ is close to $f(\mathrm{OPT}(k, V \setminus E))$, then our solution is within a constant factor of $f(\mathrm{OPT}(k, V \setminus E))$;
- Lemma 5.5 shows that if $f(B_{\lceil \log k \rceil})$ is small compared to $f(\mathrm{OPT}(k, V \setminus E))$, then our solution is again within a constant factor of $f(\mathrm{OPT}(k, V \setminus E))$.

Lemma 5.4. *If there does not exist a partition of S such that at least half of its buckets are full, then for the set Z produced by* STAR-T-GREEDY *we have*

$$f(Z) \geq \left(1 - e^{-1/3}\right)\left(f\left(B_{\lceil \log k \rceil}\right) - \frac{4m}{wk}\tau\right),$$

where $B_{\lceil \log k \rceil}$ is a not-fully-populated bucket in the last partition that minimizes $\left|B_{\lceil \log k \rceil} \cap E\right|$ and $|E| \leq m$.

Lemma 5.5. *If there does not exist a partition of S such that at least half of its buckets are full, then for the set Z produced by* STAR-T-GREEDY*,*

$$f(Z) \geq (1 - e^{-1})\big(f(OPT(k, V \setminus E)) - f(B_{\lceil \log k \rceil}) - \tau\big),$$

where $B_{\lceil \log k \rceil}$ is any not-fully-populated bucket in the last partition.

The proofs of Lemma 5.4 and Lemma 5.5 are provided in Section 5.4.3 and Section 5.4.4, respectively.

With these lemmas in hands, we are ready to prove Theorem 5.2.

Proof of Theorem 5.2

First, we prove the bound on the size of S:

$$|S| = \sum_{i=0}^{\lceil \log k \rceil} w\lceil k/2^i \rceil \min\{2^i, k\} \leq \sum_{i=0}^{\lceil \log k \rceil} w(k/2^i + 1)2^i \leq (\log k + 5)\, wk. \tag{5.3}$$

From the definition of w (see Table 5.1) we obtain $S = O((k + m\log k)\log k)$.

Next, we show the approximation guarantee. We first define $\gamma \stackrel{\text{def}}{=} \frac{4m}{wk}$, $\alpha_1 \stackrel{\text{def}}{=} \left(1 - e^{-1/3}\right)$, and $\alpha_2 \stackrel{\text{def}}{=} \left(1 - e^{-1}\right)$. Assume for a moment that τ is fixed. Lemma 5.4 and Lemma 5.5 provide two bounds on $f(Z)$, one increasing and one decreasing in $f(B_{\lceil \log k \rceil})$. By balancing out the two bounds, we derive

$$f(Z) \geq \left(\frac{\alpha_1\alpha_2}{\alpha_1 + \alpha_2}\right)(f(\text{OPT}(k, V \setminus E)) - (1+\gamma)\tau), \tag{5.4}$$

where the equality holds when $f(B_{\lceil \log k \rceil}) = \frac{\alpha_2 f(\text{OPT}(k,V\setminus E)) - (\alpha_2 - \gamma\alpha_1)\tau}{\alpha_2 + \alpha_1}$.

Observe that the bound (5.4) (that is obtained by combining Lemma 5.4 and Lemma 5.5) and the bound provided by Lemma 5.3 correspond to two complementary cases. That is, for any τ, exactly one of the two bounds holds. By setting

$$\tau = \alpha f(\text{OPT}(k, V \setminus E)),$$

where α is as provided in the statement of this theorem, we obtain that in either of the two complementary cases we have

$$f(Z) \geq \frac{1}{\frac{2}{\alpha_2(1-\gamma)} + \frac{1}{\alpha_1}} f(\text{OPT}(k, V \setminus E)). \tag{5.5}$$

Following the definition of w (see Table 5.1), we have $\gamma \leq 1/\lceil \log k \rceil$, and hence from (5.5)

$$f(Z) \geq \frac{1}{\frac{2}{\alpha_2\left(1 - \frac{1}{\lceil \log k \rceil}\right)} + \frac{1}{\alpha_1}} f(\text{OPT}(k, V \setminus E)),$$

which after substituting α_1 and α_2 proves our main result

$$\begin{aligned} f(Z) &\geq \frac{\left(1 - e^{-1/3}\right)\left(1 - e^{-1}\right)\left(1 - \frac{1}{\lceil \log k \rceil}\right)}{2\left(1 - e^{-1/3}\right) + \left(1 - e^{-1}\right)} f(\text{OPT}(k, V \setminus E)) \\ &\geq 0.149\left(1 - \frac{1}{\lceil \log k \rceil}\right) f(\text{OPT}(k, V \setminus E)). \end{aligned}$$

5.4.2 Case I – A Partition is Half-Full

In this section we prove Lemma 5.3.

Lemma 5.3. *If there exists a partition in S such that at least half of its buckets are full, then for the set Z produced by* STAR-T-GREEDY *we have*

$$f(Z) \geq \left(1 - e^{-1}\right)\left(1 - \frac{4m}{wk}\right)\tau. \tag{5.2}$$

Proof. Let $i_\star$ be a partition such that half of its buckets are full. Let $B_{i_\star,j}$ be a full bucket that minimizes $\left|B_{i_\star,j} \cap E\right|$. We first show that $f\left(B_{i_\star,j} \setminus E\right)$ is close to τ.

In STAR-T, every partition contains $w\lceil k/2^i \rceil$ buckets. Hence, the number of full buckets in partition $i_\star$ is at least $wk/2^{i_\star+1}$. That further implies

$$\left|B_{i_\star,j} \cap E\right| \leq \frac{2^{i_\star+1} m}{wk}. \tag{5.6}$$

Taking into account that $B_{i_\star,j}$ is a full bucket, we conclude

$$\left|B_{i_\star,j} \setminus E\right| \geq \left|B_{i_\star,j}\right| - \frac{2^{i_\star+1}m}{wk}. \tag{5.7}$$

From the property of our Algorithm (line 5) every element added to $B_{i_\star,j}$ increased the utility of this bucket by at least $\tau/2^{i_\star}$. Combining this with the fact that $B_{i_\star,j}$ is full, we conclude that the gain of every element in this bucket is at least $\tau/\left|B_{i_\star,j}\right|$. Therefore, from (5.7) it follows:

$$f\left(B_{i_\star,j} \setminus E\right) \geq \left(\left|B_{i_\star,j}\right| - \frac{2^{i_\star+1}m}{wk}\right)\frac{\tau}{\left|B_{i_\star,j}\right|} = \tau\left(1 - \frac{2^{i_\star+1}m}{\left|B_{i_\star,j}\right|wk}\right). \tag{5.8}$$

Taking into account that $2^{i_\star+1} \leq 4\left|B_{i_\star,j}\right|$ this further reduces to

$$f\left(B_{i_\star,j} \setminus E\right) \geq \tau\left(1 - \frac{4m}{wk}\right). \tag{5.9}$$

Finally,

$$\begin{aligned}
f(Z) = f(\textsc{Greedy}(k, S \setminus E)) &\geq (1-e^{-1}) f(\mathrm{OPT}(k, S \setminus E)) \\
&\geq \left(1-e^{-1}\right) f(\mathrm{OPT}(k, B_{i^*,j} \setminus E)) && (5.10)\\
&= \left(1-e^{-1}\right) f\left(B_{i_\star,j} \setminus E\right) && (5.11)\\
&\geq \left(1-e^{-1}\right)\left(1 - \frac{4m}{wk}\right)\tau, && (5.12)
\end{aligned}$$

where (5.10) follows from $(B_{i_\star,j} \setminus E) \subseteq (S \setminus E)$, (5.11) follows from the fact that $|B_{i_\star,j}| \leq k$, and (5.12) follows from (5.9). □

5.4.3 Case II – No Partition is Half-Full; The Last Partition is Large

We start by studying some properties of E that we use in the proof of Lemma 5.4.

Lemma 5.6. *Let B_i be a bucket in partition $i > 0$, and let $E_i \stackrel{\text{def}}{=} B_i \cap E$ denote the elements that are removed from this bucket. Given a bucket B_{i-1} from the previous partition such that $|B_{i-1}| < 2^{i-1}$ (i.e. B_{i-1} is not fully populated), the loss in the bucket B_i due to the removals is at most*

$$f(E_i \mid B_{i-1}) < \frac{\tau}{2^{i-1}}|E_i|.$$

Proof. First, we can bound $f(E_i \mid B_{i-1})$ as follows

$$f(E_i \mid B_{i-1}) \leq \sum_{e \in E_i} f(e \mid B_{i-1}). \tag{5.13}$$

Consider a single element $e \in E_i$. There are two possible cases: $f(e) < \frac{\tau}{2^{i-1}}$, and $f(e) \geq \frac{\tau}{2^{i-1}}$. In the first case, $f(e \mid B_{i-1}) \leq f(e) < \frac{\tau}{2^{i-1}}$. In the second one, as $|B_{i-1}| < 2^{i-1}$ we conclude $f(e \mid B_{i-1}) < \frac{\tau}{2^{i-1}}$, as otherwise the streaming algorithm would place e in B_{i-1}. These observations together with (5.13) imply:

$$f(E_i \mid B_{i-1}) < \sum_{e \in E_i} \frac{\tau}{2^{i-1}} = \frac{\tau}{2^{i-1}}|E_i|.$$

□

Lemma 5.7. *For every partition i, let B_i denote a bucket such that $|B_i| < 2^i$ (i.e. no partition is fully populated), and let $E_i = B_i \cap E$ denote the elements that are removed from B_i. The loss in the bucket $B_{\lceil \log k \rceil}$ due to the removals, given all the remaining elements in the previous buckets, is at most*

$$f\left(E_{\lceil \log k \rceil} \,\middle|\, \bigcup_{j=0}^{\lceil \log k \rceil - 1} \left(B_j \setminus E_j\right)\right) \leq \sum_{j=1}^{\lceil \log k \rceil} \frac{\tau}{2^{j-1}} |E_j|.$$

Proof. We proceed by induction. More precisely, we show that for any $i \geq 1$ the following holds

$$f\left(E_i \,\middle|\, \bigcup_{j=0}^{i-1} \left(B_j \setminus E_j\right)\right) \leq \sum_{j=1}^{i} \frac{\tau}{2^{j-1}} |E_j|. \tag{5.14}$$

Once we show that (5.14) holds, the lemma will follow immediately by setting $i = \lceil \log k \rceil$.

Base case $i = 1$. Since B_0 is not fully populated and the maximum number of elements in the partition $i = 0$ is 1, it follows that both B_0 and E_0 are empty. Then the term on the left hand side of (5.14) for $i = 1$ becomes $f(E_1)$. As $|B_0| < 1$ we can apply Lemma 5.6 to obtain

$$f(E_1) = f(E_1 \mid B_0) \leq |E_1| \frac{\tau}{2^0}.$$

Inductive step $i > 1$. Now we show that (5.14) holds for $i > 1$, assuming that it holds for $i - 1$. First, due to submodularity we have

$$f\left(E_{i-1} \,\middle|\, \bigcup_{j=0}^{i-2} \left(B_j \setminus E_j\right)\right) \geq f\left(E_{i-1} \,\middle|\, \bigcup_{j=0}^{i-1} \left(B_j \setminus E_j\right)\right),$$

and, hence, we can write

$$\begin{aligned} f\left(E_i \,\middle|\, \bigcup_{j=0}^{i-1} \left(B_j \setminus E_j\right)\right) &\leq f\left(E_i \,\middle|\, \bigcup_{j=0}^{i-1} \left(B_j \setminus E_j\right)\right) + f\left(E_{i-1} \,\middle|\, \bigcup_{j=0}^{i-2} \left(B_j \setminus E_j\right)\right) - f\left(E_{i-1} \,\middle|\, \bigcup_{j=0}^{i-1} \left(B_j \setminus E_j\right)\right) \\ &= f\left(E_i \cup \bigcup_{j=0}^{i-1} \left(B_j \setminus E_j\right)\right) + f\left(E_{i-1} \,\middle|\, \bigcup_{j=0}^{i-2} \left(B_j \setminus E_j\right)\right) - f\left(E_{i-1} \cup \bigcup_{j=0}^{i-1} \left(B_j \setminus E_j\right)\right). \end{aligned} \tag{5.15}$$

Due to monotonicity, the first term can be further bounded by

$$f\left(E_i \cup \bigcup_{j=0}^{i-1} \left(B_j \setminus E_j\right)\right) \leq f\left(E_i \cup B_{i-1} \cup \bigcup_{j=0}^{i-2} \left(B_j \setminus E_j\right)\right), \tag{5.16}$$

and for the third term we have

$$f\left(E_{i-1} \cup \bigcup_{j=0}^{i-1} \left(B_j \setminus E_j\right)\right) = f\left(E_{i-1} \cup B_{i-1} \cup \bigcup_{j=0}^{i-2} \left(B_j \setminus E_j\right)\right) \geq f\left(B_{i-1} \cup \bigcup_{j=0}^{i-2} \left(B_j \setminus E_j\right)\right), \tag{5.17}$$

where to obtain the identity we used that $E_{i-1} \cup (B_{i-1} \setminus E_{i-1}) = E_{i-1} \cup B_{i-1}$.

By substituting the obtained bounds (5.16) and (5.17) in (5.15) we obtain:

$$f\left(E_i \,\middle|\, \bigcup_{j=0}^{i-1}\left(B_j \setminus E_j\right)\right) \leq f\left(E_i \,\middle|\, B_{i-1} \cup \bigcup_{j=0}^{i-2}\left(B_j \setminus E_j\right)\right) + f\left(E_{i-1} \,\middle|\, \bigcup_{j=0}^{i-2}\left(B_j \setminus E_j\right)\right)$$
$$\leq f\left(E_i \mid B_{i-1}\right) + f\left(E_{i-1} \,\middle|\, \bigcup_{j=0}^{i-2}\left(B_j \setminus E_j\right)\right), \tag{5.18}$$

where the second inequality follows by submodularity.

Next, Lemma 5.6 can be used (as $|B_{i-1}| < 2^{i-1}$) to bound the first term in (5.18):

$$f\left(E_i \,\middle|\, \bigcup_{j=0}^{i-1}\left(B_j \setminus E_j\right)\right) \leq \frac{\tau}{2^{i-1}}|E_i| + f\left(E_{i-1} \,\middle|\, \bigcup_{j=0}^{i-2}\left(B_j \setminus E_j\right)\right). \tag{5.19}$$

To conclude the proof, we use the inductive hypothesis that (5.14) holds for $i-1$, which together with (5.19) implies

$$f\left(E_i \,\middle|\, \bigcup_{j=0}^{i-1}\left(B_j \setminus E_j\right)\right) \leq \frac{\tau}{2^{i-1}}|E_i| + \sum_{j=1}^{i-1}\frac{\tau}{2^{j-1}}|E_j| = \sum_{j=1}^{i}\frac{\tau}{2^{j-1}}|E_j|,$$

as desired. □

Here, we outline a technical lemma that is used in the proof of Lemma 5.4

Lemma 5.8. *For any submodular function f on a ground set V, and any sets $A, B, R \subseteq V$, we have*

$$f(A \cup B) - f(A \cup (B \setminus R)) \leq f(R \mid A).$$

Proof. Define $R_2 \stackrel{\text{def}}{=} A \cap R$, and $R_1 \stackrel{\text{def}}{=} R \setminus A = R \setminus R_2$. We have

$$\begin{aligned} f(A \cup B) - f(A \cup (B \setminus R)) &= f(A \cup B) - f((A \cup B) \setminus R_1) \\ &= f(R_1 \mid (A \cup B) \setminus R_1) \\ &\leq f(R_1 \mid (A \setminus R_1)) && (5.20) \\ &= f(R_1 \mid A) && (5.21) \\ &= f(R_1 \cup R_2 \mid A) && (5.22) \\ &= f(R \mid A), \end{aligned}$$

where (5.20) follows from the submodularity of f, (5.21) follows since A and R_1 are disjoint, and (5.22) follows since $R_2 \subseteq A$. □

Lemma 5.4. *If there does not exist a partition of S such that at least half of its buckets are full, then for the set Z produced by* STAR-T-GREEDY *we have*

$$f(Z) \geq \left(1 - e^{-1/3}\right)\left(f\left(B_{\lceil \log k \rceil}\right) - \frac{4m}{wk}\tau\right),$$

where $B_{\lceil \log k \rceil}$ is a not-fully-populated bucket in the last partition that minimizes $\left|B_{\lceil \log k \rceil} \cap E\right|$ and $|E| \leq m$.

Proof. Let B_i denote a bucket in partition i which is not fully populated ($B_i \leq \min\{2^i, k\}$), and for which $|E_i|$, where $E_i = B_i \cap E$, is of minimum cardinality. Such bucket exists in every partition i due to the assumption of the lemma that more than a half of the buckets are not fully populated.

First,

$$f\left(\bigcup_{i=0}^{\lceil \log k \rceil} (B_i \setminus E_i)\right) \geq f\left(B_{\lceil \log k \rceil}\right) - f\left(E_{\lceil \log k \rceil} \middle| \bigcup_{i=0}^{\lceil \log k \rceil - 1} (B_i \setminus E_i)\right) \tag{5.23}$$

$$\geq f\left(B_{\lceil \log k \rceil}\right) - \sum_{i=1}^{\lceil \log k \rceil} \frac{\tau}{2^{i-1}} |E_i|, \tag{5.24}$$

where (5.23) follows from Lemma 5.8 by setting $B = B_{\lceil \log k \rceil}$, $R = E_{\lceil \log k \rceil}$ and $A = \bigcup_{i=0}^{\lceil \log k \rceil - 1} (B_i \setminus E_i)$. As we consider buckets that are not fully populated, Lemma 5.7 is used to obtain (5.24). Next, we bound each term $\frac{\tau}{2^{i-1}} |E_i|$ in (5.24) independently.

From Algorithm 9 we have that partition i consists of $w\lceil k/2^i \rceil$ buckets. By the assumption of the lemma, more than half of those are not fully populated. Recall that B_i is defined to be a bucket of partition i which is not fully populated and which minimizes $|E_i|$. Let $\tilde{E}_i$ be the subset of E that intersects buckets of partition i. Then, $|E_i|$ can be bounded as follows:

$$|E_i| \leq \frac{|\tilde{E}_i|}{\frac{w\lceil k/2^i \rceil}{2}} \leq \frac{2^{i+1}|\tilde{E}_i|}{wk}.$$

Hence, the sum on the left hand side of (5.24) can be bounded as

$$\sum_{i=1}^{\lceil \log k \rceil} \frac{\tau}{2^{i-1}} |E_i| \leq \sum_{i=1}^{\lceil \log k \rceil} \frac{\tau}{2^{i-1}} \frac{2^{i+1}|\tilde{E}_i|}{wk} = \frac{4}{wk}\tau \sum_{i=1}^{\lceil \log k \rceil} |\tilde{E}_i| \leq \frac{4|E|}{wk}\tau.$$

Putting the last inequality together with (5.24) we obtain

$$f\left(\bigcup_{i=0}^{\lceil \log k \rceil} (B_i \setminus E_i)\right) \geq f\left(B_{\lceil \log k \rceil}\right) - \frac{4|E|}{wk}\tau.$$

Observe also that

$$\bigcup_{i=0}^{\lceil \log k \rceil} |B_i \setminus E_i| \leq \bigcup_{i=0}^{\lceil \log k \rceil} |B_i| \leq k + \bigcup_{i=0}^{\lfloor \log k \rfloor} 2^i \leq 3k,$$

which implies

$$f\left(\mathrm{OPT}(3k, S \setminus E)\right) \geq f\left(\bigcup_{i=0}^{\lceil \log k \rceil} (B_i \setminus E_i)\right) \geq f\left(B_{\lceil \log k \rceil}\right) - \frac{4|E|}{wk}\tau.$$

Finally,

$$\begin{aligned} f(Z) = f(\textsc{Greedy}(k, S \setminus E)) &\geq \left(1 - e^{-1/3}\right) f\left(\mathrm{OPT}(3k, S \setminus E)\right) \\ &\geq \left(1 - e^{-1/3}\right)\left(f\left(B_{\lceil \log k \rceil}\right) - \frac{4|E|}{wk}\tau\right) \\ &\geq \left(1 - e^{-1/3}\right)\left(f\left(B_{\lceil \log k \rceil}\right) - \frac{4m}{wk}\tau\right), \end{aligned} \tag{5.25}$$

as desired. □

5.4.4 Case III – No Partition is Half-Full; The Last Partition is Small

In this section we prove Lemma 5.5.

Lemma 5.5. *If there does not exist a partition of S such that at least half of its buckets are full, then for the set Z produced by* STAR-T-GREEDY,

$$f(Z) \geq (1-e^{-1})\big(f(OPT(k, V \setminus E)) - f(B_{\lceil \log k \rceil}) - \tau\big),$$

where $B_{\lceil \log k \rceil}$ is any not-fully-populated bucket in the last partition.

Proof. Let $B_{\lceil \log k \rceil}$ denote a bucket in the last partition which is not fully populated. Such bucket exists due to the assumption of the lemma that more than a half of the buckets are not fully populated.

Let X and Y be two sets such that Y contains all the elements from $\mathrm{OPT}(k, V \setminus E)$ that are placed in the buckets that precede bucket $B_{\lceil \log k \rceil}$ in S, and let $X \stackrel{\text{def}}{=} \mathrm{OPT}(k, V \setminus E) \setminus Y$. In that case, for every $e \in X$ we have

$$f\left(e \,\middle|\, B_{\lceil \log k \rceil}\right) < \frac{\tau}{k} \tag{5.26}$$

due to the fact that $B_{\lceil \log k \rceil}$ is the bucket in the last partition and is not fully populated.

We proceed to bound $f(Y)$:

$$\begin{aligned}
f(Y) &\geq f(\mathrm{OPT}(k, V \setminus E)) - f(X) && (5.27)\\
&\geq f(\mathrm{OPT}(k, V \setminus E)) - f\left(X \,\middle|\, B_{\lceil \log k \rceil}\right) - f\left(B_{\lceil \log k \rceil}\right) && (5.28)\\
&\geq f(\mathrm{OPT}(k, V \setminus E)) - f\left(B_{\lceil \log k \rceil}\right) - \sum_{e \in X} f\left(e \,\middle|\, B_{\lceil \log k \rceil}\right) && (5.29)\\
&\geq f(\mathrm{OPT}(k, V \setminus E)) - f\left(B_{\lceil \log k \rceil}\right) - \frac{\tau}{k}|X| && (5.30)\\
&\geq f(\mathrm{OPT}(k, V \setminus E)) - f\left(B_{\lceil \log k \rceil}\right) - \tau, && (5.31)
\end{aligned}$$

where (5.27) follows from $f(\mathrm{OPT}(k, V \setminus E)) = f(X \cup Y)$ and submodularity, Eq (5.28) and Eq (5.29) follow from monotonicity and submodularity, respectively. Equation (5.30) follows from (5.26), and (5.31) follows from $|X| \leq k$.

Finally, we have:

$$\begin{aligned}
f(Z) = f(\textsc{Greedy}(k, S \setminus E)) &\geq \left(1-e^{-1}\right) f(\mathrm{OPT}(k, S \setminus E))\\
&\geq \left(1-e^{-1}\right) f(\mathrm{OPT}(k, Y)) && (5.32)\\
&= \left(1-e^{-1}\right) f(Y) && (5.33)\\
&\geq \left(1-e^{-1}\right)\left(f(\mathrm{OPT}(k, V \setminus E)) - f(B_{\lceil \log k \rceil}) - \tau\right), && (5.34)
\end{aligned}$$

where (5.32) follows from $Y \subseteq (S \setminus E)$, (5.33) follows from $|Y| \leq k$, and (5.34) follows from (5.31). □

5.5 Algorithm Without the Access to the Optimum Value

In the statement of Theorem 5.2, algorithm STAR-T is invoked with the parameter τ that is a function of an unknown value $f(\mathrm{OPT}(k, V \setminus E))$. However, $f(\mathrm{OPT}(k, V \setminus E))$ is unknown.

To deal with this shortcoming, we show how to extend the idea of [BMKK14] of maintaining multiple parallel instances of our algorithm in order to approximate $f(\mathrm{OPT}(k, V \setminus E))$. For a given constant $\epsilon > 0$, compared to STAR-T, this approach increases the space requirement by a factor of $\log_{1+\epsilon} k$ and provides a $(1+\epsilon)$-approximation compared to the value obtained in Theorem 5.2. More precisely, the following statement holds.

Theorem 5.1

For any given constant $\epsilon > 0$, there exists a single-pass streaming algorithm that under at most m removals from the stream collects $O\left((k + m\log k)\log^2 k\right)$ elements and outputs set Z of cardinality at most k such that

$$f(Z) \geq \frac{0.149}{1+\epsilon}\left(1 - \frac{1}{\lceil \log k \rceil}\right) f(\mathrm{OPT}(k, V \setminus E)).$$

Proof. First, $f(\mathrm{OPT}(k, V \setminus E))$ can be bounded in the following way: $\eta \leq f(\mathrm{OPT}(k, V \setminus E)) \leq k\eta$, where η denotes the largest value of any of the elements of $V \setminus E$, i.e. $\eta = \max_{e \in (V \setminus E)} f(e)$. In case we are given η, we follow the same approach as in [BMKK14] by considering all the $O\left(\log_{1+\epsilon} k\right)$ possible values of $f(\mathrm{OPT}(k, V \setminus E))$ from the set $\{(1+\epsilon)^i \mid i \in \mathbb{Z}, \eta \leq (1+\epsilon)^i \leq k\eta\}$. For each of the thresholds independently and in parallel we then run STAR-T, and hence build $O\left(\log_{1+\epsilon} k\right)$ different summaries (see Algorithm 11). After the stream ends, on each of the summaries we run algorithm STAR-T-GREEDY and report the maximum output over all the runs (see Algorithm 12). As this approach runs $O(\log_{1+\epsilon} k)$ copies of our algorithm, it requires $O(\log_{1+\epsilon} k)$ more memory space than stated in Theorem 5.2. Furthermore, since we are approximating $f(\mathrm{OPT}(k, V \setminus E))$ as the geometric series with base $(1+\epsilon)$, our final result is an $(1+\epsilon)$-approximation of the value provided in the theorem.

Algorithm 11: STAR-T- PARALLEL
Running parallel of STAR-T

Input:
- The ground set V
- Cardinality constraint k
- $\eta \in \mathbb{R}$

1 Let α be as defined in the statement of Theorem 5.2.
2 $O \leftarrow \left\{(1+\epsilon)^i \mid \eta \leq (1+\epsilon)^i \leq k\eta\right\}$
3 Create a set of instances $\mathscr{I} \leftarrow \left\{\text{STAR-T}(V, k, \alpha\eta) \mid \eta \in O\right\}$, and run all the instances in parallel over the stream.
4 Let $\mathscr{S} \leftarrow \left\{\text{the output of instance } I \mid I \in \mathscr{I}\right\}$.
5 **return** $\mathscr{S}$

Unfortunately, the value η might also not be known a priori. However, η is some value among the $m+1$ largest elements of the stream. This motivates the following idea. At every moment, we keep $m+1$ largest elements of the stream. Let L denote that set (note that L changes during the course of the stream). Then, for different values of η belonging to the set $\{f(e) \mid e \in L\}$ we approximate $f(\mathrm{OPT}(k, V \setminus E))$ as described above. Here we make a minor difference, as also described in [BMKK14]. Namely, instead of instantiating all the copies of the algorithm corresponding to $\eta \leq (1+\epsilon)^i \leq km$, we instantiate copies of the algorithm

Algorithm 12: STAR-T- GREEDY- PARALLEL
The final algorithm for running parallel instances

Input:
- Family of sets $\mathscr{S}$
- The set E of removed elements
- Cardinality constraint k

return $Z \leftarrow \arg\max_{S \in \mathscr{S}} \textsc{Greedy}(k, S \setminus E)$

corresponding to the values of $f(\mathrm{OPT}(k, V \setminus E))$ from the set $\{(1+\epsilon)^i \mid i \in \mathbb{Z}, \eta \leq (1+\epsilon)^i \leq 2k\eta\}$. We do so as an element e can belong to an instance of our algorithm even if $f(\mathrm{OPT}(k, V \setminus E)) = 2kf(e)$.

Next, let e be a new element that arrives on the stream. If e is not among the $m+1$ largest elements of the stream seen so far, we do not instantiate any new copy of our algorithm. On the other hand, if e should replace another element $e' \in L$ because e' does not belong to the $m+1$ largest elements of the stream anymore, we redefine L to be $(L \setminus \{e'\}) \cup \{e\}$, and update the instances. The instances are updated as follows: we instantiate copies (those that do not exist already) of our algorithm for $\eta = f(e)$ as described above; and, any instance of our algorithm corresponding to $\eta = f(e')$, but not to any other element of L, we discard.

To bound the space complexity, we start with the following observation – given an element e, we do not need to add e to any instance of our algorithm corresponding to $f(\mathrm{OPT}(k, V \setminus E)) < f(e)$. This reasoning is justified by the following: if $e \in E$, then it does not matter whether we keep e in our summary or not; if $e \notin E$, then $f(\mathrm{OPT}(k, V \setminus E)) \geq f(e)$. Therefore, those thresholds that are less than $f(e)$ are not a good estimate of the optimum solution with respect to e. To keep the memory space low, we pass an element e to the instances of our algorithm corresponding to the of $f(\mathrm{OPT}(k, V \setminus E))$ being in set $\{(1+\epsilon)^i \mid i \in \mathbb{Z}, f(e) \leq (1+\epsilon)^i \leq 2kf(e)\}$. Notice that, by the structure of our algorithm, e will not be added to any instance of our algorithm with threshold more than $2kf(e)$.

Combining all the observations, we derive the following conclusion. At any point during the execution, every element of L belongs to at most $O(\log_{1+\epsilon} k)$ instances of our algorithm. Define $e_{\min} \stackrel{\text{def}}{=} \arg\min_{e \in L} f(e)$. Then by the definition, every element $a \notin L$ kept in the parallel instances of our algorithms is such that $f(a) \leq f(e_{\min})$. This further implies that a also belongs to at most $O(\log_{1+\epsilon} k)$ instances corresponding to the following set of values $\{(1+\epsilon)^i \mid i \in \mathbb{Z}, f(e_{\min}) \leq (1+\epsilon)^i \leq 2kf(e_{\min})\}$. Therefore, the total memory usage of the elements of L is $O\left(m \log_{1+\epsilon} k\right)$. On the other hand, since all the elements not in L belong to at most $O(\log_{1+\epsilon} k)$ different instances of STAR-T, the total memory those elements occupy is $O((k + m\log k)\log k \log_{1+\epsilon} k)$. Therefore, the memory complexity of this approach is $O\left((k + m\log k)\log k \log_{1+\epsilon} k\right)$. □

6 Fast Recovery for Separated Sparsity Signals

This chapter is based on a joint work with Aleksander Mądry, and Ludwig Schmidt. It has been accepted to the 21st International Conference on Artificial Intelligence and Statistics (**AISTATS**) 2018 [MMS18] under the title

A Fast Algorithm for Separated Sparsity via Perturbed Lagrangians.

6.1 Introduction

Over the past two decades, sparsity has become a widely used tool in several fields including signal processing, statistics, and machine learning. In many cases, sparsity is the key concept that enables us to capture important structure present in real-world data while making the resulting problem computationally tractable and suitable for mathematical analysis. Among the many applications of sparsity are sparse linear regression, compressed sensing, sparse PCA, and dictionary learning.

The first wave of sparsity-based techniques focused on the standard notion of sparsity that only constrains the number of non-zeros. Over time, it became apparent that extending the notion of sparsity to encompass more complex structures present in real-world data can offer significant benefits. Specifically, utilizing such additional structure often improves the statistical efficiency in estimation problems and the interpretability of the final result. There is now a large body of work on structured sparsity that has introduced popular models such as group sparsity and hierarchical sparsity [YL06, EM09, BCDH10, MJOB11, HZM11, RRN12, NRWY12, BBC14, HIS15c]. These statistical improvements, however, come at a computational cost: the resulting optimization problems are often much harder to solve. The key reason is that the combinatorial sparsity structures give rise to non-convex constraints. Consequently, many of the resulting algorithms have significantly worse running time than their "standard sparsity" counterparts. This trade-off raises an important question: can we design algorithms for structured sparsity that match the time complexity of commonly used algorithms for standard sparsity?

In our work, we address this question in the context of the *separated sparsity* model, a popular sparsity model for data with a known minimum distance between large coefficients [HDC09, DDJB10, HB11, DSRB13, FMN15]. In the one-dimensional case, such as time series data, neuronal spike trains are a natural example. Here, a minimum refractory period ensures separation between consecutive spikes. In two dimensions, separation constraints arise in the

context of astronomical images or super-resolution applications [HB11, HK15].

We introduce new algorithms for separated sparsity that run in *nearly-linear* time. This significantly improves over prior work that required at least quadratic time, which is quickly prohibitive for large data sets. An important consequence of our fast running time is that it enables methods that utilize separated sparsity yet are essentially as fast as their counterparts based on standard sparsity only. For instance, when we instantiate our algorithm in compressive sensing, the running time of our method matches that of common methods such as IHT or CoSaMP.

Our algorithms stem from a primal-dual linear programming (LP) perspective on the problem. Our theoretical findings reveal a rich structure behind the separated sparsity model, which we utilize to obtain efficient methods. Interestingly, our final algorithm has a very simple form that can be interpreted as a *Lagrangian relaxation* of the sparsity constraint. In spite of the non-convexity of the constraint, our algorithm is still guaranteed to find the globally optimal solution.

We also show that these algorithmic and theoretical contributions directly translate into empirical efficiency. Specifically, we demonstrate that, compared to the state of the art procedures, our methods yield an order of magnitude speed-up already on moderate-size inputs. We run experiments on synthetic data and real world neuronal spike train signals.

6.1.1 Problem Setup

In this section, we formally define separated sparsity and the corresponding algorithmic problems. As a concrete application of our algorithms, we instantiate them in a sparse recovery context that is representative for many statistical problems such as compressed sensing and sparse linear regression.

First, we briefly introduce our notation. As usual, $[d]$ denotes the set $\{1,\dots,d\}$. We say that a vector $\theta \in \mathbb{R}^d$ is k-sparse if θ contains at most k non-zero coefficients. We define the support of θ as the set of indices corresponding to non-zero coefficients, i.e., $\mathrm{supp}(\theta) = \{i \in [d] \mid \theta_i \neq 0\}$. We let $\|\theta\|$ denote the ℓ_2-norm of a vector $\theta \in \mathbb{R}^d$.

Separated sparsity. *Sparsity models* are a natural way to formalize structure beyond "standard" sparsity [BCDH10]. In this work we focus on the the separated sparsity model, defined as follows [HDC09]. For a support $\Omega \subseteq [d]$, let $\mathrm{sep}(\Omega) = \min_{i \neq j \in \Omega} |i - j|$ be the minimum separation of two indices in the support. We define the following two sets of supports: set $\mathbb{M}_\Delta = \{\Omega \subseteq [d] \mid \mathrm{sep}(\Omega) \geq \Delta\}$, and Δ-*separated sparsity* supports $\mathbb{M}_{k,\Delta} = \{\Omega \in \mathbb{M}_\Delta \mid |\Omega| = k\}$. That is, $\mathbb{M}_{k,\Delta}$ is the set of support patterns containing k non-zeros with at least $\Delta - 1$ zero entries between consecutive non-zeros.

In order to employ separated sparsity in statistical problems, we often want to add constraints based on the support set $\mathbb{M}_{k,\Delta}$ to optimization problems such as empirical risk minimization. A standard way of incorporating constraints into gradient-based algorithms is via a *projection operator*. In the context of separated sparsity, this corresponds to the following problem.

Problem 1. *For a given input vector $x \in \mathbb{R}^d$, our goal is to project x onto the set $\mathbb{M}_{k,\Delta}$, i.e., to*

find a vector $\hat{x}$ such that

$$\hat{x} \in \operatorname*{arg\,min}_{x' \in \mathbb{R}^d \,:\, supp(x') \in \mathbb{M}_{k,\Delta}} \|x - x'\| . \tag{6.1}$$

Problem 1 is the main algorithmic problem we address in this chapter.

Sparse recovery. Structured sparsity has been employed in a variety of machine learning tasks. In order to keep the discussion coherent, we present our results in the context of the well-known sparse linear model:

$$y = X\theta^* + e \tag{6.2}$$

where $y \in \mathbb{R}^n$ are the observations/measurements, $X \in \mathbb{R}^{n \times d}$ is the design or measurement matrix, and $e \in \mathbb{R}^n$ is a noise vector. The goal is to find a good estimate $\hat{\theta}$ of the unknown parameters θ^* up to the noise level.

The authors of [BCDH10] give an elegant framework for incorporating structured sparsity into the estimation problem outlined above. They design a general recovery algorithm that relies on a model-specific *projection oracle*. In the case of separated sparsity, this oracle is required to solve precisely the Problem 1 stated above.

6.1.2 Our Results

Our main contribution is a new algorithm for Problem 1 that we call ***L**agrangian **A**pproach to the **S**eparated **S**parsity **P**roblem* (LASSP). The pseudo code is given in Algorithm 13.

Theorem 6.1

Let $c \in \mathbb{R}^d$, and let $\gamma \in \mathbb{N}_+$ be the maximal number of bits needed to store any c_i. There is an implementation of LASSP that for every c computes a solution $\hat{c}$ to Problem 1. With probability $1 - 1/d$, the algorithm runs in time $O(d(\gamma + \log d))$.

Combined with the framework of [BCDH10], we get the following.

Corollary 6.2. *Let y, X, θ^*, and e be as in the sparse linear model in Equation (6.2). We assume that $supp(\theta^*) \in \mathbb{M}_{k,\Delta}$ and that X satisfies the model-RIP for $\mathbb{M}_{k,\Delta}$. There is an algorithm that for every y and e returns an estimate $\hat{\theta}$ such that*

$$\|\hat{\theta} - \theta^*\|_2 \leq C\|e\|_2 .$$

Moreover, the algorithm runs in time $\widetilde{O}(T_X + d)$, where T_X is the time of multiplying the matrices X and X^T by a vector.

The corollary shows that, up to logarithmic factors, the running time is dominated by $T_X + d$. This matches the time complexity of standard sparse recovery and shows that we can utilize separated sparsity without a significant increase in time complexity. Many measurement matrices in compressive sensing enable fast multiplication with X (e.g., a subsampled Fourier matrix), in which case the total running time becomes $\widetilde{O}(d)$. We validate these theoretical findings in Section 6.6 by showing that LASSP runs significantly faster than the state of the

art algorithm used for sparse recovery with separation constraints, while retaining the same accuracy of the recovered signal.

Our algorithm LASSP is randomized. However, we also design a deterministic nearly-linear time algorithm and prove the following theorem.

Theorem 6.3

Let $c \in \mathbb{R}^d$ be the input vector and let γ be as in Theorem 6.1. Then there is an algorithm that computes a solution $\hat{c}$ satisfying Equation (6.1) and runs in time $O(d(\gamma + \log k) \log d \log \Delta)$.

Further, in Section 6.5 we present a dynamic programming approach that for a specific, but also natural, family of instances solves the separated sparsity problem in even linear time.

We also consider a natural extension to the 2D-variant of the separated sparsity projection problem and show that it is NP-hard in Section 6.8. Moreover, in Section 6.9, we extend our model to allow for blocks of separated variables and show that our algorithms for $\mathbb{M}_{k,\Delta}$ also applies to the more general variant. Finally, separated sparsity can be used to model signals in which a longer pattern is repeated multiple times so that any two patterns are at least Δ apart. This model is called *disjoint pulse streams* [HB11]. Again, the algorithmic core remains the same and algorithms for $\mathbb{M}_{k,\Delta}$ can also be used for this generalization.

6.1.3 Related Work

The papers [HDC09, FMN15] are closely related to our work. The paper [HDC09] proposed the separated sparsity model, provided a sample complexity upper bound, and gave an LP-based model-projection algorithm. However, they resorted to a black-box approach for solving the LP, that lead to a fairly prohibitive $O(d^{3.5})$ time complexity. Recently, [FMN15] provided a faster dynamic program for this problem with a time complexity of $O(d^2)$ and also showed a sample complexity lower bound. The algorithmic aspect of these papers is the main difference from our work: we exploit structure in both the primal and dual formulations of the LP and give an algorithm that *provably* runs in *nearly-linear* time.

Beside the papers addressing the core algorithmic question of projecting onto separated sparse vectors, there is much of work utilizing the sparsity model for applications in neural signal processing [DDJB10, HB11, DSRB13] and recovery with coherent dictionaries [DB13, Nee15]. In the latter application, the separated sparsity constraint enforces that the signal representation only consists of incoherent dictionary atoms. We expect that our algorithmic techniques will also lead to improvements in the context of these applications.

In addition to separated sparsity, a large body of work on structured sparsity has emerged over the past few years. We refer the reader to the surveys [DE11, BJMO12, Wai14, HIS15b] for an overview. The line of work most relevant to this chapter is the *model-based compressive sensing* framework introduced in [BCDH10], which is also the starting point for [HDC09, FMN15]. While the framework provides a general recovery scheme based on a model-projection oracle satisfying Equation (6.1), it does not provide any guidance on how to design such oracles for a specific sparsity model. We address precisely this problem for separated sparsity with our nearly-linear time algorithm.

Recently, two papers have proposed a fairly general framework for deriving model-projection oracles via *graph sparsity*, i.e., sparsity structures that can be defined through connected components in a graph on the signal coefficients [HZM11, HIS15c]. This framework generalizes several previously studied sparsity models such as block sparsity, tree sparsity, and cluster sparsity. Moreover, the paper [HIS15c] gives projections that run in nearly-linear time. Interestingly, these tools do not apply to the separated sparsity model we study in our work. Intuitively, graph sparsity captures structures in which the non-zero coefficients are *clustered together*, while the separated sparsity model achieves a reduction in sample complexity for the opposite reason: the non-zero coefficients are *far apart*. Moreover, the algorithms in [HIS15c] are *approximate* and project into a sparsity model with a relaxed sparsity constraint. As we explain in Section 6.3, the separated sparsity model requires more careful control over the output sparsity in order to achieve a meaningful sample complexity improvement over "standard" sparsity. We circumvent this issue by providing an *exact* projection onto the separated sparsity model.

Recent progress. The separated sparsity projection can be reduced to the problem of finding a minimum-weight path of length k on an edge-weighted directed graph. Recently it was shown that this graph can be designed so that the edge-weights satisfy the concave Monge property [AT18]. This further implies that the separated sparsity problem can be solved in nearly-linear time [AST94, BLP92].

6.2 Organization

We start by showing, in Section 6.3, that a rather obvious *approximation* algorithm for the separated sparsity projection is not sufficient to recover separated sparsity signals. Then, in Section 6.4 we describe our randomized approach to the exact separated sparsity projection. Section 6.5 provides our refined dynamic programming result. In Section 6.6 we present the results of evaluation of our randomized algorithm. Section 6.7 is devoted to describing our nearly-linear deterministic algorithm. in Section 6.8 we provide our study of the two-dimensional variant of the separated sparsity model. We conclude this chapter by Section 6.9 in which we define a generalization of the separated sparsity model, and also show how to reduce the recovery problem of this new model to our nearly-linear time algorithms.

6.3 An Approximate-Projection Counterexample

Before we delve into details of our algorithms, we show that an obvious (and perhaps at the first sight natural) approximate-approach does not solve the separated sparsity recovery.

Multiple recent algorithms for structured sparse projections build on the *approximation-tolerant* framework of [HIS15a]. In this framework, it suffices to design approximate projections instead of solving the model projection problem exactly. For some sparsity structures, this approach has led to significantly faster algorithms [HIS14, HIS15c]. Hence it is interesting to see whether the approximation route is also helpful for separated sparsity. In fact, it is easy to design the following 2-approximation algorithm that runs in nearly-linear time.

Partition the vector c into blocks of length Δ. Then, number those blocks 1 through $\lceil d/\Delta \rceil$ in the order they appear in c. In each block, choose an index corresponding to the largest

coordinate of c within that block. Split those indices into two groups G and H based on the parity of the corresponding block. Formally, define G and H as follows

$$G \stackrel{\text{def}}{=} \left\{ \arg \max_{j=(i-1)\Delta+1 \ldots \min\{i\Delta,d\}} c_j \mid 1 \le i \le \lceil d/\Delta \rceil \text{ and } i \text{ is odd} \right\},$$

and similarly

$$H \stackrel{\text{def}}{=} \left\{ \arg \max_{j=(i-1)\Delta+1 \ldots \min\{i\Delta,d\}} c_j \mid 1 \le i \le \lceil d/\Delta \rceil \text{ and } i \text{ is even} \right\}.$$

Let $g_1, \ldots, g_{|G|}$ be an ordering of the elements of G so that $c_{g_i} \ge c_{g_j}$ whenever $i \le j$. Similarly, let $h_1, \ldots, h_{|H|}$ be an ordering of the elements of H such that $c_{g_i} \ge c_{g_j}$ whenever $i \le j$.

Next, define $G' \stackrel{\text{def}}{=} \{g_i \mid 1 \le i \le \min\{k, |G|\}\}$ and $H' \stackrel{\text{def}}{=} \{h_i \mid 1 \le i \le \min\{k, |H|\}\}$. Now, it is not hard to see that the set attaining larger value among G' and H' has a solution value that is at least $OPT/2$, where OPT is the maximum sum attainable with a Δ-separated and k-sparse vector (in the language of [HIS15a], this is an approximate head projection).

While the above algorithm runs in nearly-linear time and achieves a constant-factor approximation, there is a catch. In particular, the returned support pattern might contain fewer than k indices (as a simple example, consider the case $k = 2$, $\Delta = 2$, and $c = (1, 100, 1)$). This raises the question of the sample complexity for this relaxed sparsity model. Let k' denote the number of output indices of the 2-approximation algorithm. As we have described in the preliminaries, the sample complexity of the separated sparsity model is $O(k \log(n/k - \Delta))$. For concreteness, we now consider the case $k' = k/2$ and $d/k - \Delta \in O(1)$ (the latter is the important regime where the separated sparsity model achieves a sample complexity of $m = O(k)$). Then we have $d/k' - \Delta \in O(1) + \Delta$. Therefore, if we would like to apply the 2-approximation algorithm in the case $k' = k/2$, the sample complexity would be

$$O(k \log(2d/k - \Delta)) \;=\; O(k \log \Delta) \;=\; O(k \log d/k)$$

where we used the assumption that $d/k - \Delta \in O(1)$. It is important to note that this sample complexity is worse than $O(k)$ and falls back to the $O(k \log d/k)$ sample complexity of "standard" k-sparse recovery. So by using an approximate projection we have lost the sample complexity advantage of the separated sparsity model. Since we already know how to recover k-sparse vectors in nearly-linear time without resorting to structured sparsity, the approximation algorithm does not provide a novel trade-off.

6.4 Randomized Approach

In this section we describe our randomized algorithm and provide the underlying proofs.

6.4.1 Overview

Given an arbitrary vector $x \in \mathbb{R}^d$, Problem 1 requires us to find a vector $\hat{x}$ such that $\hat{x} \in \mathcal{M}_{k,\Delta}$ and $\|x - \hat{x}\|$ is minimized. We now slightly reformulate the problem. Let $c \in \mathbb{R}^d$ be a vector such that $c_i = x_i^2$ for all i. Then it is not hard to see that this problem is equivalent to finding a set of k entries in the vector c such that each of these entries is separated by at least Δ and the

sum of these entries is maximized. Hence our main algorithmic problem is to find a set of k entries in an non-negative input vector c so that the entries are Δ-separated and their sum is maximized. More formally, our goal is to find a support $\hat{S}$ such that

$$\hat{S} \in \operatorname*{arg\,max}_{S \in \mathbb{M}_{k,\Delta}} \sum_{i \in S} c_i \,. \tag{6.3}$$

In the following, we also consider a relaxed version of (6.3) called PROJLAGR, which is parametrized by a trade-off parameter λ and a vector $\tilde{c}$:

$$\text{PROJLAGR}(\lambda, \tilde{c}) \stackrel{\text{def}}{=} \operatorname*{arg\,max}_{S \in \mathbb{M}_{\Delta}} \sum_{i \in S} \tilde{c}_i + \lambda \left(k - |S|\right) \,.$$

Intuitively, PROJLAGR represents a Lagrangian relaxation of the sparsity constraint in Equation (6.3).

Our algorithm LASSP is a *Las Vegas* algorithm: it always returns a correct answer, but the running time of the algorithm is randomized. Concretely, LASSP repeats a main loop until a stopping criterion is reached. Every iteration of LASSP first adds a small perturbation to the coefficients c (see Line 3). This perturbation has only a small effect on the solution but improves the "conditioning" of the corresponding non-convex Lagrangian relaxation PROJLAGR so that it returns a globally optimal solution that almost satisfies the constraint. As we show in Section 6.4.4, we can solve this relaxation (Line 4) in nearly-linear time. After the algorithm has solved the Lagrangian relaxation, it obtains the final support $\hat{S}$ in line 5 by solving PROJLAGR on a slightly shifted $\hat{\lambda}$ to ensure that the constraint is satisfied with good probability.

Remark: We assume that the bit precision γ required to represent the coefficients c is finite, and we provide our results as a function of γ. For practical purposes, γ is usually a constant. Since the solution to Equation (6.3) is invariant under scaling by a positive integer and γ is finite, without loss of generality we assume that $c \in \mathbb{Z}^d$.

Algorithm 13: LASSP(c, k)

Input:
- Cost vector $c \in \mathbb{Z}^d$
- Sparsity $k \in \mathbb{N}_+$

Output: A solution $\hat{S}$ to (6.3)

1 **repeat**
2 Let $X \in \mathbb{Z}^d$ be a vector such that X_i is chosen uniformly at random from $\{0, \ldots, d^3 - 1\}$, $\forall i$
3 Define vector $\tilde{c} \leftarrow d^4 c + X$
4 Let $\hat{\lambda} \leftarrow \arg\min_{\lambda \in \mathbb{Z}} \text{PROJLAGR}(\lambda, \tilde{c})$. In case of ties, maximize $\hat{\lambda}$.
5 Choose $\hat{S} \in \text{PROJLAGR}\left(\hat{\lambda} - \frac{1}{d+1}, \tilde{c}\right)$
6 **until** $\left|\hat{S}\right| = k$
7 **return** $\hat{S}$

6.4.2 Roadmap

We first provide a simple proof (Lemma 6.6 in Section 6.4.3) that LASSP outputs the right answer if it terminates. But does LASSP terminate on every input? Answering this question

is the most intricate part in the analysis of our randomized approach. We split the proof in two main pieces. The first part is Section 6.4.4, where we provide an alternative view on the separated sparsity problem based on linear programming duality. The duality view paves the way towards proving our main results. In particular, we show that the subroutines in LASSP can be implemented quickly.

Lemma 6.4. *Single iteration of* LASSP *can be implemented to run in time* $O(d(\gamma + \log d))$.

The second part is Section 6.4.5, where we further study the duality view on separated sparsity. We show that after perturbing the input instance in Line 3 of LASSP, the support obtained with a shifted $\hat{\lambda}$ in Line 5 has cardinality k with high probability.

Lemma 6.5. *Algorithm* LASSP *runs only a single iteration with probability at least* $1 - 1/d$.

Together with Lemma 6.6, these results yield Theorem 6.1.

6.4.3 Proof of Correctness

We begin our analysis by proving that LASSP returns a correct result *if the algorithm terminates*. As we will see later, establishing termination is the crucial part of the analysis. Nevertheless, the following lemma is a useful warm-up for understanding how the different pieces of our algorithm fit together.

Lemma 6.6. *When* LASSP *terminates, it outputs a support* $\hat{S}$ *such that* x *restricted to* $\hat{S}$ *is a solution to Problem 1.*

Proof. As we have argued above, the problems in Equations (6.1) and (6.3) are equivalent. So, we show that LASSP outputs a solution to the problem in Equation (6.3).

Let $\hat{S}$ be the set returned by LASSP. By the condition of the loop in Line 6, we have $|\hat{S}| = k$. So, as $\hat{S} \in \mathbb{M}_\Delta$ (see the definition of PROJLAGR), we have $\hat{S} \in \mathbb{M}_{k,\Delta}$.

Now, towards a contradiction, assume that support $\hat{S}$ is not a solution to the problem in Equation (6.3), while support $S^\star$ is. This implies that $\sum_{i \in S^\star} c_i > \sum_{i \in \hat{S}} c_i$. Now since, without loss of generality, we assumed that $c \in \mathbb{Z}^d$, the last inequality implies $\sum_{i \in S^\star} c_i \geq 1 + \sum_{i \in \hat{S}} c_i$, and hence

$$\sum_{i \in S^\star} d^4 c_i \geq d^4 + \sum_{i \in \hat{S}} d^4 c_i \,. \tag{6.4}$$

Observe that for any support $S \in \mathbb{M}_{k,\Delta}$, the term $\lambda\,(k - |S|)$ equals zero, and recall that $\hat{S}, S^\star \in \mathbb{M}_{k,\Delta}$. Furthermore, by the definition of the random vector X in line 2 and from (6.4)

$$\begin{aligned} \sum_{i \in S^\star} \left(d^4 c_i + X_i\right) &\geq \sum_{i \in S^\star} d^4 c_i \geq d^4 + \sum_{i \in \hat{S}} d^4 c_i \\ &> \sum_{i \in \hat{S}} \left(d^4 c_i + X_i\right). \end{aligned}$$

Since $S^\star \in \mathbb{M}_\Delta$, this chain of inequalities contradicts Line 5 of LASSP which chooses $\hat{S}$ as an optimal solution to $\textsc{ProjLagr}\left(\hat{\lambda} - 1/(d+1), d^4 c + X\right)$. This further implies that $\hat{S}$ is a solution to Problem 1. □

6.4.4 Part I – To Duality and Further

We now analyze the running time of a single iteration of LASSP. We provide a series of equivalences, as illustrated in Figure 6.1, in order to exploit structure in the separated sparsity problem. More precisely, we start with a linear programming (LP) view on separated sparsity. It has already been shown that this viewpoint yields a totally unimodular LP [HDC09], which implies that the LP has an integral solution. Hence solving the LP solves the separated sparsity projection in Problem 1. However, prior work did not utilize this connection to reason about the power of the Lagrangian relaxation approach to the problem.

We begin our detailed analysis of the LP with the dual program $\mathscr{D}$. By strong duality, the value of $\mathscr{D}$ equals the value of the primal LP. Then we cast $\mathscr{D}$ as minimization of LP $\mathscr{D}_\lambda$ over λ. This reduction will play the central role in our analysis and connect $\mathscr{D}_\lambda$ to Line 4 of LASSP.

The LP perspective

We start with a linear programming view on problem (6.3) by considering its LP relaxation denoted by $\mathscr{P}$:

$$\begin{aligned}
\text{maximize} \quad & c^T u \\
\text{subject to} \quad & \sum_{i=1}^{d} u_i = k \\
& \sum_{j=i}^{\min\{i+\Delta-1,d\}} u_j \leq 1 \qquad \forall i = 1 \dots d \\
& u_i \geq 0 \qquad \forall i = 1 \dots d
\end{aligned}$$

Given an LP $\mathscr{A}$ we use $\mathrm{VAL}_{\mathscr{A}}$ to denote its optimal objective value. (Note that we can restrict our attention to the case of c being integral since $\mathscr{P}$ is invariant under shifting and scaling the vector c by a positive integer.) (A variant of) $\mathscr{P}$ was first formulated in [HDC09] and it was already observed there that $\mathscr{P}$ has a very important property: it is *totally unimodular* and thus there always exists an optimal solution to it that is integral [NW88]. The proof of total unimodularity provided in [HDC09] applied to slightly different variant of this LP, thus for completeness we provide a proof of total unimodularity of our LP $\mathscr{P}$.

Lemma 6.7. *The constraint matrix of problem $\mathscr{P}$ is* totally unimodular *(TUM).*

Proof. We first rewrite the constraints of $\mathscr{P}$ to be in the form $Bu \leq b, u \geq 0$, where $Bu \leq b$ is defined as

$$\begin{aligned}
\sum_{i=1}^{d} u_i &\leq k \\
\sum_{i=1}^{d} -u_i &\leq -k \\
\sum_{j=i}^{\min\{i+\Delta-1,d\}} u_j &\leq 1 \qquad \forall i = 1 \dots d
\end{aligned}$$

Let D be a square submatrix of B. If we show that $\det D \in \{-1, 0, 1\}$, then by the definition of TUM the lemma follows.

Now, D is either a binary *interval matrix*, or it has one row where all the non-zero entries are consecutive and equal -1 and the other rows constitute a binary interval matrix. Let D' be a matrix defined as $D'_{i,j} \stackrel{\text{def}}{=} |D_{i,j}|$. Then, we have $|\det D| = |\det D'|$. Also, we have that D' is a binary interval matrix. However, it is well known that binary interval matrices are TUM, see [NW88]. Hence, $\det D \in \{-1, 0, 1\}$ □

Remark: This implies that a solution to $\mathscr{P}$ can be used to obtain a solution to the separated sparsity model projection: if $u^\star$ is an optimal solution to $\mathscr{P}$, we can derive the optimal support of (6.3) from the non-zero entries among the LP variables $u_1^\star, \ldots, u_d^\star$. It is unclear, however, if there is a way to directly solve this LP fast, e.g. it is not known how to solve $\mathscr{P}$ directly in time matching the running time of our algorithm LASSP.

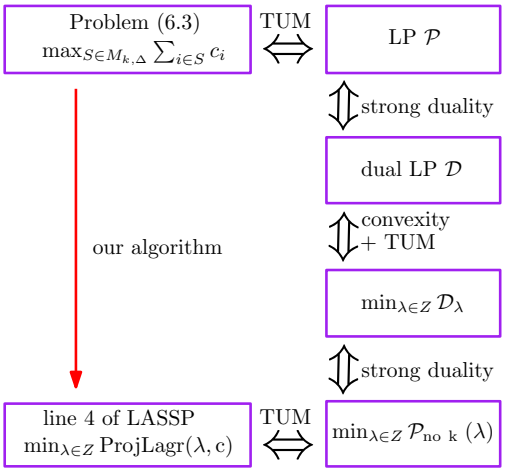

Figure 6.1 – The values of the problems in the diagram are equal. Every equivalence relation carries structural information that we utilize in our analysis.

A key step in our approach is understanding the separated sparsity structure from the dual point of view. The dual LP to $\mathscr{P}$, denoted by $\mathscr{D}$, is given as follows

$$\begin{aligned}
\text{minimize} \quad & w_0 k + \sum_{i=1}^{d} w_i \\
\text{subject to} \quad & w_0 + \sum_{\substack{j \,:\, j \geq 1 \text{ and} \\ j \leq i \leq j+\Delta-1}} w_j \geq c_i \qquad \forall i = 1 \ldots d \\
& w_i \geq 0 \qquad \forall i = 1 \ldots d \\
& w_0 \in \mathbb{R}
\end{aligned}$$

Then, as $\mathscr{P}$ is integral, so is $\mathscr{D}$.

Corollary 6.8. *For an integer k and a vector of integers c, there exists $\hat{w}$ such that $\hat{w}$ is an optimum of $\mathscr{D}$ and $\hat{w}_0 \in \mathbb{Z}$.*

Proof. By Lemma 6.7, problem $\mathscr{P}$ is totally unimodular. By [Sch02] we have that $\mathscr{D}$ is totally unimodular as well. This implies that $\mathscr{D}$ has an integral optimal solution. □

We define $\mathscr{D}_v$ as the LP $\mathscr{D}$ in which the variable w_0 is set to v. Also, in what follows, we will be interested in cost vectors and sparsity parameters other than c and k, respectively. Hence, whenever this is the case, we will write $\mathscr{P}(c',k')$ to refer to the program $\mathscr{P}$ for cost vector c' and sparsity k'. Similarly, whenever we consider some different cost vector c' and sparsity parameter k', we will denote the corresponding dual LP by $\mathscr{D}(c',k')$. Now, it is not hard to show the following lemma.

Lemma 6.9. *$\mathscr{D}_v(c',k')$ is convex with respect to v.*

Proof. Let $\hat{w}$ and $\tilde{w}$ be w-solution to $\mathscr{D}_{\hat{v}}(c',k')$ and $\mathscr{D}_{\tilde{v}}(c',k')$, respectively. Note that $\hat{w}_0 = \hat{v}$ and $\tilde{w}_0 = \tilde{v}$. Now if we show that

$$\mathscr{D}_{\frac{\hat{v}+\tilde{v}}{2}}\left(c',k'\right) \leq \frac{\mathscr{D}_{\hat{v}}(c',k') + \mathscr{D}_{\tilde{v}}(c',k')}{2},$$

the convexity will follow. We start by showing that $\frac{\hat{w}+\tilde{w}}{2}$ is a feasible w-vector for the dual, assuming that both $\hat{w}$ and $\tilde{w}$ are feasible. We consider the feasibility of each of the constraints.

(1) From

$$\hat{w}_0 + \sum_{j\,:\, j\leq i\leq j+\Delta-1 \text{ and } j\geq 1} \hat{w}_j \geq c'_i$$

and

$$\tilde{w}_0 + \sum_{j\,:\, j\leq i\leq j+\Delta-1 \text{ and } j\geq 1} \tilde{w}_j \geq c'_i$$

we have

$$(\hat{w}_0 + \tilde{w}_0) + \sum_{j\,:\, j\leq i\leq j+\Delta-1 \text{ and } j\geq 1} (\hat{w}_j + \tilde{w}_j) \geq 2c'_i,$$

which implies

$$\frac{\hat{w}_0 + \tilde{w}_0}{2} + \sum_{j\,:\, j\leq i\leq j+\Delta-1 \text{ and } j\geq 1} \frac{\hat{w}_j + \tilde{w}_j}{2} \geq c'_i.$$

(2) Also, from $\hat{w}_j \geq 0$ and $\tilde{w}_j \geq 0$ we have $\frac{\hat{w}_j+\tilde{w}_j}{2} \geq 0$.

(3) Trivially, $\frac{\hat{w}_0+\tilde{w}_0}{2} \in \mathbb{R}$.

So, indeed $\frac{\hat{w}+\tilde{w}}{2}$ is feasible. Hence, we have

$$\begin{aligned}
\mathscr{D}_{\frac{\hat{v}+\tilde{v}}{2}} &\leq \frac{\hat{w}_0 + \tilde{w}_0}{2} k' + \sum_{i=1}^{d'} \frac{\hat{w}_j + \tilde{w}_j}{2} \\
&= \frac{\left(\hat{w}_0 k' + \sum_{i=1}^{d'} \hat{w}_j\right) + \left(\tilde{w}_0 k' + \sum_{i=1}^{d'} \tilde{w}_j\right)}{2} \\
&= \frac{\mathscr{D}_{\hat{v}}(c',k') + \mathscr{D}_{\tilde{v}}(c',k')}{2}.
\end{aligned}$$

This completes the proof. □

From the definition of $\mathscr{D}$, the following equality holds $\mathrm{VAL}_{\mathscr{D}} = \min_{\lambda\in\mathbb{R}} \mathrm{VAL}_{\mathscr{D}_\lambda}$. Furthermore, Corollary 6.8 implies that it is sufficient to consider λ in $\mathbb{Z}$ only, i.e.

$$\mathrm{VAL}_{\mathscr{D}} = \min_{\lambda\in\mathbb{Z}} \mathrm{VAL}_{\mathscr{D}_\lambda}. \tag{6.5}$$

Now, Lemma 6.9 implies that we can obtain $\mathrm{VAL}_{\mathscr{D}}$ by applying ternary search over λ on function $\mathrm{VAL}_{\mathscr{D}_\lambda}$.

Implementing one iteration of LASSP efficiently

We now derive the final connection between $\mathscr{D}$ and LASSP which will enable us to obtain $\hat{\lambda}$ at line 4 in nearly-linear time. To that end, consider $\mathscr{P}_{\text{no-}k}(\lambda)$ defined as

$$\begin{aligned} \text{maximize} \quad & c^T u + \lambda(k - \mathbb{1}^T u) \\ \text{subject to} \quad & \sum_{j=i}^{\min\{i+\Delta-1,d\}} u_j \le 1 \qquad \forall i = 1\dots d \\ & u_i \ge 0 \qquad \forall i = 1\dots d \end{aligned}$$

Observe that compared to $\mathscr{P}$, $\mathscr{P}_{\text{no-}k}(\lambda)$ does not contain the sparsity constraint. Furthermore, the LP $\mathscr{P}_{\text{no-}k}(\lambda)$ is a relaxed version of $\textsc{ProjLagr}(\lambda, c)$. Also, as $\mathscr{P}$ is, then $\mathscr{P}_{\text{no-}k}(\lambda)$ is totally unimodular. Now this sequence of conclusions results in the following.

Corollary 6.10. *Problems $\mathscr{P}_{no\text{-}k}(\lambda)$ and $\textsc{ProjLagr}(\lambda, c)$ are equivalent.*

To obtain the final connection, we consider $L_{\mathscr{P}}$ given by

$$L_{\mathscr{P}} \stackrel{\text{def}}{=} \min_{\lambda\in\mathbb{R}} \mathscr{P}_{\text{no-}k}(\lambda).$$

Next, note that the dual of $\mathscr{P}_{\text{no-}k}(\lambda)$ is $\mathscr{D}_\lambda$. Hence, the strong duality implies $\mathrm{VAL}_{\mathscr{D}_\lambda} = \mathrm{VAL}_{\mathscr{P}_{\text{no-}k}(\lambda)}$. That together with $\mathrm{VAL}_{\mathscr{D}} = \min_{\lambda\in\mathbb{R}} \mathrm{VAL}_{\mathscr{D}_\lambda}$ yields that $\mathscr{D}$ and $L_{\mathscr{P}}$ coincide as functions in λ. Furthermore, since we can solve $\mathscr{D}$ by applying ternary search over integral values of λ and $\mathscr{D}_\lambda$, we can solve $L_{\mathscr{P}}$ by applying ternary search over integral values of λ and function $\mathscr{P}_{\text{no-}k}(\lambda)$. But since $\mathscr{P}_{\text{no-}k}(\lambda)$ and $\textsc{ProjLagr}(\lambda, c)$ are equivalent, we can also obtain $\hat{\lambda}$ at line 4 of LASSP by applying ternary search over λ.

Now, it is very easy to see that for an optimal solution $w^\star$ of $\mathscr{D}$ we have $w_0^\star \le \max_i |c_i|$. It is also not hard to show that there is an optimal solution such that $w_0^\star \ge -(k-1)\max_i |c_i|$ (see Lemma 6.21). Therefore, in order to find optimal $\hat{\lambda}$ it suffices to execute $O\left(\log \max_i |c_i| + \log k\right) = O(\gamma + \log d)$ iterations of ternary search.

Every iteration of the ternary search invokes $\textsc{ProjLagr}$, which can be implemented to run in linear time.

Lemma 6.11. *Given λ and $\hat{c} \in \mathbb{R}^d$, there is an algorithm that finds support $\hat{S} \in \textsc{ProjLagr}(\lambda, \hat{c})$ in time $O(d)$.*

Proof. Observe that for a fixed λ, solving $\textsc{ProjLagr}(\lambda, \hat{c})$ is equivalent to finding support $S' \in \mathbb{M}_\Delta$ that maximize $\sum_{i\in S'} (\hat{c}_i - \lambda)$. Hence, we can reinterpret $\textsc{ProjLagr}(\lambda, \hat{c})$ as follows: given a vector $\tilde{c} = \hat{c} - \lambda\mathbb{1}$, select a subset of $[d]$ of indices (not necessarily k of them) so that (i)

every two indices are at least Δ apart, and (ii) the sum of the values of $\tilde{c}$ at the selected indices is maximized.

This task can be solved by standard dynamic programming in the following way. For every i, we define s_i to be the maximum value of the described task restricted to the first i indices of $\tilde{c}$. Then, it is easy to see that $s_{i+1} = \max\{s_i, s_{i+1-\Delta} + \tilde{c}_{i+1}\}$. Namely, we can either decide not to select index $i+1$, in which case the best is already contained in s_i; or, we can decide to select index $i+1$ which has value $\tilde{c}_{i+1}$ and for the rest we consider $s_{i+1-\Delta}$. Therefore, s_d can be obtained in time $O(d)$. Now it is easy to reconstruct the corresponding support in linear time. □

Putting all together proves Lemma 6.4.

6.4.5 Part II – Active Constraints

Note that the chain of equivalences present in Figure 6.1 shows that $\min_\lambda \textsc{ProjLagr}(\lambda, c)$ outputs the sum of coordinates of an optimal solution of problem 6.1. Prior to our work, it was not even known how to obtain this value in time faster than $O(dk)$, while our result shows we can compute it in nearly-linear time. So, it is natural to ask whether the same relaxation also outputs a support of size k? The answer is, unfortunately, no. To see that, consider the example: $c = (4,7,5,0,0,5,8,5)$, $\Delta = 2$, and $k = 3$. For $\lambda > 2$ every solution $u^\star_\lambda$ to $\textsc{ProjLagr}(\lambda, c)$ is such that $u^\star_\lambda$ has less than k non-zeros. On the other hand, for every $\lambda \le 2$ the exists a solution $u^\star_\lambda$ such that $u^\star_\lambda$ contains more than k non-zeros. Therefore, there is no λ, neither $\lambda - 1/(d+1)$, for which $\textsc{ProjLagr}(\lambda, c)$ provably outputs a support of cardinality k. This also suggests that the perturbation we apply in lines 2-3 is essential!

Instead of studying lines 2-5 of LASSP directly, we shift our focus to $\mathscr{D}_\lambda$. In particular, we exhibit very close connection between its structure and the sparsity of the primal solution, which we present via the notion of "active constraints". Then we use these findings in our analysis to show that slight perturbation of the input instance, while not affecting the value of the solution, makes it possible to obtain a solution to problem 6.3 by applying Lagrangian relaxation.

Solving $\mathscr{D}_\lambda$

The crucial property of the LP $\mathscr{D}$ is the following: for a fixed value of the variable w_0, we can solve $\mathscr{D}$ in *linear* time. Observe that once we fixed the value of w_0, all remaining constraints in $\mathscr{D}_\lambda$ are "local" since they only affect a known interval of length Δ. They are also ordered in a natural way. As a result, we can solve $\mathscr{D}_\lambda$ by making a single pass over these variables. Starting with w_1 and all variables set to 0, we consider each constraint from left to right and increase the variables to satisfy these constraints in a lazy manner. That is, if in our pass we reach a constraint with index i that is still not satisfied, we increase the value of w_i until that constraint becomes satisfied and then move to the next constraint. Given c and λ, algorithm Dual-Greedy, i.e. Algorithm 14, formalizes this approach.

Lemma 6.12. *For any c', and k', the algorithm Dual-Greedy(c', v) computes an optimal solution $\mathscr{D}_v(c', k')$ in linear time.*

Algorithm 14: Dual-Greedy(c, λ)

Input:
- $c \in \mathbb{Z}^d$
- $\lambda \in \mathbb{R}$

Output: an optimal solution w to $\mathscr{D}$ such that $w_0 = \lambda$

```
w ← ;   w_0 ← λ                                          // initialize the output
sum_Δ ← 0                  // store the sum of the most recent Δ variables of w
for i ← 1...d do
    if i − Δ ≥ 1 then
        sum_Δ ← sum_Δ − w_{i−Δ}
    diff ← c_i − (w_0 + sum_Δ)      // compute "how far" constraint i is from being
    tight
    if diff > 0 then
        w_i ← diff              // if constraint i is not satisfied, make it tight
    sum_Δ ← sum_Δ + w_i   // update the sum of the Δ most recent variables of w
return w                                          // solution to 𝒟 s.t.   w_0 = λ
```

Proof. Let $c' \in \mathbb{N}_+^{d'}$. We show that the algorithm Dual-Greedy(c', v) outputs vector $\tilde{w}$ in $O(d')$ time such that $\mathscr{D}_v(c', k')$ is minimized and $\tilde{w}_0 = v$. To show the running time, observe that every iteration in the for loop takes $O(1)$ time, so the total algorithm runs in $O(d')$ time.

Next, it is easy to see that $\tilde{w}$ is a feasible solution to $\mathscr{D}_v(c', k')$. We show that it is a minimal as well.
Towards a contradiction, assume there is a vector w' such that $w'_0 = \tilde{w}_0 = v$, w' is a feasible solution to $\mathscr{D}(c', k')$, and $\|w'\|_1 < \|\tilde{w}\|_1$. Then, there exists an index j such that $w'_j \neq \tilde{w}_j$ and $w'_i = \tilde{w}_i$ for all $i < j$. In case there are multiple vectors w', let w' be one that maximizes the location of mismatch j. Consider the two possible cases: $w'_j < \tilde{w}_j$, and $w'_j > \tilde{w}_j$.

Case $w'_j < \tilde{w}_j$. Observe that $\tilde{w}_j$ is chosen as a function of $\tilde{w}_0, \ldots, \tilde{w}_{j-1}$ as a minimal value so that $\tilde{w}$ is feasible. Therefore, if $w'_j < \tilde{w}_j$ and $w'_i = \tilde{w}_i$ for all $i < j$, then w' could not be feasible.

Case $w'_j > \tilde{w}_j$. First, note that $j < d'$, as otherwise $\|w'\|_1 > \|\tilde{w}\|_1$. Now, we construct w'' as follows. We set $w''_i = w'_i$ for all i different than j and $j+1$. Set $w''_j = \tilde{w}_j$ and $w''_{j+1} = w'_{j+1} + \tilde{w}_j - w'_j$. Clearly, w'' is also a feasible solution to $\mathscr{D}(c', k')$. Furthermore, $\|w''\|_1 = \|w'\|_1$ and $\tilde{w}$ and w'' agrees on first j coordinates, contradicting our choice of w'.

This concludes the proof. □

Now, to obtain an algorithm that is also able to solve the original dual LP $\mathscr{D}$ (instead of only $\mathscr{D}_v$) it suffices to provide a procedure for choosing the optimal value of v. A priori, there can be many possible choices of v and thus an exhaustive search would be prohibitive. Fortunately, as shown by Lemma 6.9, $\mathscr{D}_v$ is actually convex in v. Then, as discussed, we can use a ternary search over v to find such an optimal value.

Putting these pieces together yields the main theorem of this section. As a reminder, we assume that c is an integral vector with non-negative entries.

Theorem 6.13

There exists an algorithm Opt-Value-of-$\mathscr{D}(c', k')$ that, for any c' and k', outputs an optimal solution $w^\star$ to $\mathscr{D}(c', k')$ in time $O\left(d'(\log c'_{max} + \log k')\right)$.

Proof. **Bounding $w_0^\star$.** Following Lemma 6.9, we can use ternary search to find $w_0^\star$. However, to be able to do that, we have to know the interval $[a, b]$ we are searching over for $w_0^\star$. By the following lemma we give an upper bound on b. However, in order to provide a lower bound on a, we need to develop some more machinery. So, we defer its proof to later sections, and in Lemma 6.21 we show that a can be lower bounded by $-(k-1)c_{max}$.

Lemma 6.14. *Let $w^\star$ be a vector that minimizes $\mathscr{D}$. Then, $w_0^\star \le c_{max}$.*

Proof. Towards a contradiction, let $w^\star$ be an optimal vector such that $w_0^\star > c_{max}$. Now, as $w_i \ge 0$ for all $i \ge 1$, we have that the corresponding objective is at least $kw_0^\star > kc_{max}$. On the other hand, consider vector $\hat{w}$ such that $\hat{w}_0 = c_{max}$ and $\hat{w}_i = 0$ for all $i \ge 1$. Clearly, $\hat{w}$ is a feasible solution to $\mathscr{D}$. However, the objective function corresponding to $\hat{w}$ is $kc_{max} < kw_0^\star$, which contradicts our assumption that $w^\star$ is a minimizer of $\mathscr{D}$. □

A nearly-linear time algorithm. Now we provide an algorithm that computes the optimal value of $\mathscr{D}(c', k')$ in nearly-linear time (see Algorithm 15). It employs ternary search over the interval provided by Lemma 6.21 and Lemma 6.14 in order to find w_0 that optimizes $\mathscr{D}(c', k')$. At every step of the search, it uses the result from Lemma 6.12 to find an optimal solution to $\mathscr{D}_v(c', k')$, for v chosen at the current search step.

Algorithm 15: Opt-Value-of-$\mathscr{D}$

Input:
- $c' \in \mathbb{N}_+^{d'}$
- Sparsity k'
- Lower bound lb (if not specified, the default value is $-(k'-1)c'_{max}$)
- Upper bound ub (if not specified, the default value is c'_{max})

Output: A minimizer w^{best} to $\mathscr{D}(c', k')$ constrained to $w_0^{\text{best}} \in [lb, ub]$; if lb and ub are not specified, w^{best} is an optimal solution to $\mathscr{D}(c', k')$

$s \leftarrow lb$, $e \leftarrow ub$
while $s \le e$ **do**
  $l \leftarrow s + \lfloor \frac{e-s}{3} \rfloor$, $r \leftarrow e - \lfloor \frac{e-s}{3} \rfloor$
  $(active^l, w^l) \leftarrow$ Dual-Greedy(c, l)
  $(active^r, w^r) \leftarrow$ Dual-Greedy(c, r)
  if $kw_0^l + \sum_{i=1}^{d'} w_i^l \le kw_0^r + \sum_{i=1}^{d'} w_i^r$ **then**
    $e \leftarrow r-1$, $w^{\text{best}} \leftarrow w^l$
  else
    $s \leftarrow l+1$, $w^{\text{best}} \leftarrow w^r$
return w^{best}

From Lemma 6.12, Lemma 6.9, Corollary 6.8 and bounds on a and b, the rest of the proof follows directly. □

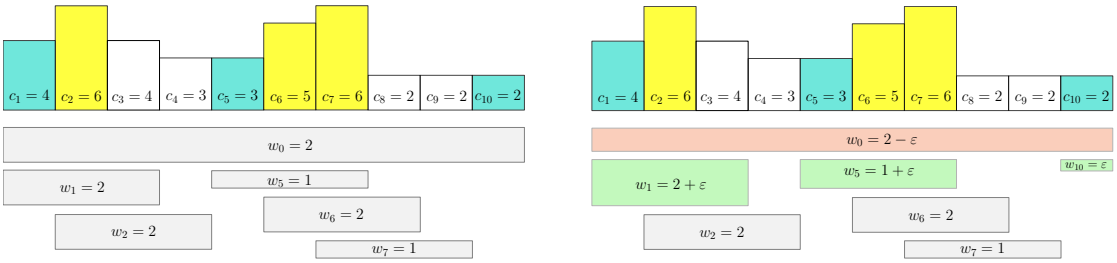

Figure 6.2 – A sketch of an instance of $\mathscr{D}$: $c = (4,6,4,3,3,5,6,2,2,2)$ and $\Delta = 3$. The left and right figure depicts w obtained by Dual-Greedy$(c,2)$ and Dual-Greedy$(c,2-\varepsilon)$ for $0 < \varepsilon < 1$, respectively. Constraints 1, 5, and 10 are active, i.e., if w_0 on the left is decreased by ε then only w_1, w_5, and w_{10} increase by ε, as shown on the right. Every w_i, for $i \geq 1$, covers Δ c-poles to its right. Colored c-poles correspond to tight constraints.

Tracking the change of $\mathscr{D}_\lambda$

Next we introduce the key concept that we need for relating the solution of dual to the sparsity of Lagrangian relaxation of the primal: the notion of *active constraints*. Let w be a vector obtained by Dual-Greedy(c,λ) and let w' be a vector obtained by Dual-Greedy$(c,\lambda-\varepsilon)$, for some fixed $\lambda \in \mathbb{Z}$ and small $\varepsilon \in (0,1)$. Then, the set of coordinates that are for ε larger in w' than in w are called active constraints. Figure 6.2 provides an illustration of this concept. Intuitively, the active constraints correspond to those variables of $\mathscr{D}$ that increase when w_0 decreases by some small value. Hence, one can interpret active constraints as gradients of $\mathscr{D}_\lambda$ with respect to the variable λ. This concept appears to be very useful in characterizing the optimal solution of $\mathscr{D}$ in an alternative way. In particular, the following lemma holds.

Lemma 6.15. *Let $c \in \mathbb{Z}^d$, $\lambda \in \mathbb{Z}$, and $w \leftarrow$ Dual-Greedy(c,λ). Then, if w has exactly k active constraints the vector w is an optimal solution to $\mathscr{D}$.*

In the interest of keeping the focus on the main ideas behind our approach, in this section we provide only a proof sketch of Lemma 6.15. A full proof of a statement stronger than Lemma 6.15 along with its proof appears in Lemma 6.17, Section 6.4.6, while in this section we provide a proof sketch. Let $w(\varepsilon) =$ Dual-Greedy$(c,\lambda-\varepsilon)$, for some small $\varepsilon \in (0,1)$. By the definition of active constraints and the fact that w has k many, there are exactly k coordinates that are larger by ε in $w(\varepsilon)$ than in w. In addition, $w(\varepsilon)_0 = \lambda - \varepsilon$ and $w_0 = \lambda$. It is not hard to show that all the other coordinates of $w(\varepsilon)$ and w are the same, which we can express as $\sum_{i=1}^{d} w(\varepsilon)_i = k\varepsilon + \sum_{i=1}^{d} w_i$. Now, recall that the objective function of dual $\mathscr{D}$ with respect to vector w equals $w_0 k + \sum_{i=1}^{d} w_i$. Then we have

$$w_0 k + \sum_{i=1}^{d} w_i = (w_0 - \varepsilon)k + k\varepsilon + \sum_{i=1}^{d} w_i = w(\varepsilon)_0 k + \sum_{i=1}^{d} w(\varepsilon)_i.$$

Hence, the objective values of dual $\mathscr{D}$ for vectors $w(\varepsilon)$ and w are equal, for **all** the values $\varepsilon \in (0,1)$. As $\mathscr{D}$ is convex in the value of variable w_0 and Dual-Greedy(c,λ) provides an optimal solution to $\mathscr{D}$ such that $w_0 = \lambda$, then w is an optimal solution to $\mathscr{D}$.

Wrapping up – perturbation and optimal sparsity

Now we use Lemma 6.15 to prove the following, which essentially justifies line 5 of LASSP.

Lemma 6.16. *Let $c \in \mathbb{Z}^d$, $\lambda \in \mathbb{Z}$, and $\tilde{w} \leftarrow$ Dual-Greedy(c, λ). Assume that $\tilde{w}$ has exactly k active constraints. Then, any optimal support $S^\star$ of* ProjLagr$(\lambda - \varepsilon, c)$*, for $0 < \varepsilon < 1/d$, has cardinality exactly k.*

Proof. Recall that by our assumption there are exactly k active constraints defined by $\tilde{w}$. Then from Lemma 6.15 it follows that $\tilde{w}$ is a minimizer of $\mathscr{D}$, i.e. $\mathrm{VAL}\mathscr{D}_\lambda = \mathrm{VAL}\mathscr{D}$. By the integrality of $\mathscr{D}$ we have that $\mathrm{VAL}\mathscr{D} \in \mathbb{Z}$. Let $\lambda' = \lambda - \varepsilon$, for some $0 < \varepsilon < 1/d$. Then, it holds $\mathrm{VAL}\mathscr{D}_{\lambda'} = \mathrm{VAL}\mathscr{D}$ as only the k variables correponding to active constraints increased by ε while $w_0 = \lambda'$ decreased by ε. From the strong duality we also have $\mathrm{VAL}\mathscr{P}_{\text{no-}k}(\lambda') = \mathrm{VAL}\mathscr{D}_{\lambda'} = \mathrm{VAL}\mathscr{D}$.

Let $\tilde{u}$ be an integral optimal solution of $\mathscr{P}_{\text{no-}k}(\lambda')$. Following the definition we have

$$\mathrm{VAL}\mathscr{P}_{\text{no-}k}(\lambda') = (c^T - \lambda \mathbb{1}^T)\tilde{u} + \lambda k + \varepsilon(\mathbb{1}^T \tilde{u} - k).$$

Now we have the following properties: $\tilde{u}$ is integral; $0 < \varepsilon|\mathbb{1}^T \tilde{u} - k| < 1$ whenever $1 \le |\mathbb{1}^T \tilde{u} - k| \le d$; $c \in \mathbb{Z}^d$; $v \in \mathbb{Z}$; and $\mathrm{VAL}\mathscr{P}_{\text{no-}k}(\lambda') \in \mathbb{Z}$. Therefore, we have $\mathbb{1}^T \tilde{u} - k = 0$, and hence $\mathbb{1}^T \tilde{u} = k$. Since we showed the equivalence between $\mathscr{P}_{\text{no-}k}(\lambda')$ and ProjLagr(λ', c) the lemma follows. □

So, if we produce λ as in Lemma 6.16, we will solve problem (6.3). These steps are implemented by lines 4-6 of LASSP. However, as illustrated in the beginning of the section, λ as in Lemma 6.16 might not exist. Intuitively, this situation happens when the number of active constraints jumps from a value smaller than k to a value larger than k for a very small change of λ. In such a case, we are unable to obtain λ as in Lemma 6.16. A key component of our analysis is showing that there is an efficient way of randomly altering c, and obtaining $\tilde{c}$, so that with high probability $\tilde{c}$ is such that: $\hat{\lambda}$ is obtained as at line 4; and, $\tilde{w} \leftarrow$ Dual-Greedy$(\tilde{c}, \hat{\lambda})$ has the same property as in Lemma 6.16. Lines 2 and 3 of LASSP implement this random perturbation. Intuitively, the perturbation achieved by X variables adds noise to our input instance so that the number of active constraints changes by at most one as λ slides over the integer domain. Following this intuition we obtain a proof of Lemma 6.5. As the proof depends on some properties of active constraints that we prove in Section 6.4.6, we provide the proof of Lemma 6.5 at the end of Section 6.4.6.

6.4.6 Active Constraints – Cont'd

In Section 10 we introduced the notion of active constraints and provided the intuition behind them. In this section we continue our study of this structure, state them algorithmically, and give a full proof of Lemma 6.15. We begin by introducing some notation. In what follows, we will be interested in cost vectors and sparsity parameters other than c and k, respectively. Hence, whenever this is the case, we will denote the corresponding dual LP by $\mathscr{D}(c', k')$.

Next, by Algorithm 16, we define active constraints algorithmically.

Active constraints provide insight into the structure of $\mathscr{D}(c', k')$ that we can leverage to optimally distribute our sparsity budget over the recursive subproblems. The optimality condition of the dual program can be described in the language of active constraints as follows.

Algorithm 16: Active-Constraints(c', w)

Input:
- $c' \in \mathbb{N}_+^{d'}$
- $w \in \mathbb{R}^{d'+1}$ a feasible vector of $\mathscr{D}(c', k')$

Output: Active constraints of $\mathscr{D}(c', k')$ for w

```
active ← ∅;   sum_Δ ← 0;   last_active ← −∞
for i ← 1 ... d' do
    if i − Δ ≥ 1 then
        sum_Δ ← sum_Δ − w_{i−Δ}
    sum_Δ ← sum_Δ + w_i
    if last_active ≤ i − Δ and c'_i − (w_0 + sum_Δ) = 0 then
        active ← active ∪ {i}
        last_active ← i
return active
```

Lemma 6.17. *Let v be an integer. Define $w^1 \leftarrow$ Dual-Greedy(c', v) and $w^2 \leftarrow$ Dual-Greedy$(c', v+1)$. Next, define $active^1 \leftarrow$ Active-Constraints(c', w^1) and $active^2 \leftarrow$ Active-Constraints(c', w^2). Then, w^1 is an optimal solution of $\mathscr{D}(c', k')$ iff $|active^1| \geq k' \geq |active^2|$.*

Proof. We first prove two properties of actives constraints. First, observe that from the way algorithm Active-Constraints(c', w) outputs $active$, it corresponds to tight constraints of $\mathscr{D}(c', k')$ for a given w. More precisely, every such tight constraint either is in $active$, or there is another tight constraint in $active$ which is at most Δ "far to the left". Furthermore, as c' is integral, it is easy to see that tight constraints for $w_0 = v$, for some integer v, and for $w_0 = v - \delta$, for $\delta \in [0, 1)$, are the same. Hence, active constraints for $w_0 = v$ and $w_0 = v - \delta$ are also the same. Putting these observations together, we get the following claim.

Lemma 6.18. *Let v be an integer and $\delta \in [0, 1)$. Also, let $\hat{w} \leftarrow$ Dual-Greedy(c', v) and $\hat{w}' \leftarrow$ Dual-Greedy$(c', v - \delta)$. Then,*

$$\textit{Active-Constraints}(c', \hat{w}) = \textit{Active-Constraints}(c', \hat{w}').$$

We point out that one can show even stronger statement about tight constraints, not necessarily active tough. Namely, it holds that if a constraint i is tight for $\hat{w}$ being Dual-Greedy(c', v), then it is tight for any Dual-Greedy(c', v') such that $v' \leq v$. It follows from the property that for any value v' there is at most one active constraint j in Active-Constraints$(c',$Dual-Greedy$(c', v'))$ such that $0 \leq i - j \leq \Delta$. Therefore, if $w_0 = v'$ gets decreased by "a very small" δ, then the variable, e.g. w_j, corresponding to active constraint will decrease by δ as well. Which in turn results i still being a tight constraint. Hence, the following lemma holds, which we utilize in the sequel.

Lemma 6.19. *Let $\hat{w} \leftarrow$ Dual-Greedy(c', v) and $\tilde{w} \leftarrow$ Dual-Greedy(c', v'), for $v' \leq v$. Then, if a constraint i is tight with respect to $\hat{w}$, it is tight with respect to $\tilde{w}$ as well.*

Using Lemma 6.18 we can show how Dual-Greedy(c', v') changes for $v' \in (v - 1, v]$.

Lemma 6.20. *Let v be an integer and $\delta \in [0,1)$. By $\hat{w}$ denote the output of Dual-Greedy(c', v), and by $\hat{w}^\delta$ the output of Dual-Greedy$(c', v-\delta)$. Let $active$ be returned by Active-Constraints$(c', \hat{w})$. Then*

$$k'\hat{w}_0^\delta + \sum_{i\geq 1} \hat{w}_i^\delta = k'\hat{w}_0 + \sum_{i\geq 1} \hat{w}_i + \delta(|active| - k').$$

Proof. From Lemma 6.18 we have that Active-Constraints$(c', \text{Dual-Greedy}(c', v-\delta)) = active$ for *all* $\delta \in [0,1)$. Now, by the construction of $active$, it holds

$$\begin{aligned} k'\hat{w}_0^\delta + \sum_{i\geq 1} \hat{w}_i^\delta &= k'(\hat{w}_0 - \delta) + \sum_{i\in active} (\hat{w}_i + \delta) + \sum_{i\geq 1 \text{ and } i\notin active} \hat{w}_i \\ &= k'\hat{w}_0 + \sum_{i\geq 1} \hat{w}_i + \delta(|active| - k'), \end{aligned}$$

as desired. □

We are now ready to finalize the proof of the lemma. Let us break the equivalence stated in the lemma into two implications, and show they are true.

($\Longleftarrow$) Let $|active^1| \geq k' \geq |active^2|$ be true. By Lemma 6.20 and the choice of v, we have that $\mathscr{D}_z(c', k')$ is non-decreasing for $z \in \left[v, v+\frac{1}{2}\right]$ and non-increasing for $z \in [v-\frac{1}{2}, v]$. As $\mathscr{D}_z(c', k')$ is convex, we have $\mathscr{D}_z(c', k')$ is minimized for $z = v$.[1]

($\Longrightarrow$) Let w^1 is an optimal solution of $\mathscr{D}(c', k')$. Recall that $\mathscr{D}_z(c', k')$ is a convex function in z. Then, as w^1 is an optimum of $\mathscr{D}(c', k')$, $\mathscr{D}_x(c', k')$ is non-increasing in z on interval $(-\infty, v]$ and non-decreasing on $[v, \infty)$. But then from Lemma 6.20 we conclude that it can only happen if $|active^1| \geq k' \geq |active^2|$.

□

Proving a claim from Theorem 6.13

Recall that in Theorem 6.13, by Lemma 6.14, we provided an upper bound on any $w_0^\star$ such that $w^\star$ is an optimum of $\mathscr{D}$. In the case of lower bound, we stated a claim without any proof. In this section we will prove that claim. First, observe that there is actually no lower bound on $w_0^\star$. To see that, consider a very simple example $c = (1)$, $\Delta = k = 1$. Nevertheless, we can provide a lower bound in the following form.

Lemma 6.21. *There exists an optimal solution $w^\star$ to $\mathscr{D}(c', k')$ such that $w_0^\star \geq -(k'-1)c'_{max}$.*

Proof. To prove the lemma, we utilize the optimality condition described by Lemma 6.17 and the following claim.

Lemma 6.22. *Let $active^1$ be the output of Active-Constraints$(c', \text{Dual-Greedy}(c', v))$ and $active^2$ be the output of Active-Constraints$(c', \text{Dual-Greedy}(c', v-t))$, for any integer v and a positive integer t. Then, it holds*

$$|active^1| \leq |active^2|.$$

[1] We use the fact that from the convexity of $\mathscr{D}_z(c', k')$ we have that $\mathscr{D}_z(c', k')$ is continuous.

Proof. Let S and S' be tight constraints for Dual-Greedy(c', v) and Dual-Greedy$(c', v-t)$, respectively. Then, as we have discussed, $S \subseteq S'$. So, all we have to show is that there are at least as many active constraints formed from S' as there are formed from S. We do that by induction on the size of S under the assumption that $S \subseteq S'$.

Base of induction: $|S| = 0$**.** As the number of active constraints is non-negative, and if $|S| = 0$ there is no active constraint, the claim follows.

Inductive step: $|S| > d'$ **and** $S \subseteq S'$**, for** $d' \geq 0$**.** Let i_{min} and i'_{min} be the smallest index of S and S', respectively. Constraint i_{min} is active for Dual-Greedy(c', v), and i'_{min} is active for Dual-Greedy$(c', v-t)$. Define $T = S \setminus \{i_{min}, \ldots, i_{min} + \Delta - 1\}$ and $T' = S' \setminus \{i'_{min}, \ldots, i'_{min} + \Delta - 1\}$. Since $i'_{min} \leq i_{min}$ it holds $T \subseteq T'$. Now, as for every active constraint j there is no other active one in the neighborhood of $\Delta - 1$ around j and except that neighborhood the other constraints are not affected by j, we have that

$$\begin{aligned} active^1 &= \{i_{min}\} \cup \{\text{active constraints for } T\}, \text{ and} \\ active^2 &= \{i_{min'}\} \cup \{\text{active constraints for } T'\}. \end{aligned}$$

Now, as $T \subseteq T'$ and $|T| < |S|$, by inductive hypothesis we have

$$|\{\text{active constraints for } T\}| \leq |\{\text{active constraints for } T'\}|,$$

and hence the lemma follows. □

Let $\hat{w} \leftarrow$ Dual-Greedy$(c', -(k-1)c_{max})$ and $active \leftarrow$ Active-Constraints$(c', \hat{w})$. We show that $|active| \geq k'$. Furthermore, we show that $l\Delta + 1 \in active$, for all $l = 0, \ldots, k' - 1$. Once it is shown, the claim follows by Lemma 6.22 and Lemma 6.17. Precisely, Lemma 6.22 shows that the number of active constraints for solutions corresponding to Dual-Greedy$(c', -(k'-1)c'_{max} - t)$, for $t \geq 0$, is at least k'. But then, from Lemma 6.17 we have that Dual-Greedy$(c', -(k'-1)c'_{max})$ or Dual-Greedy$(c', -(k'-1)c'_{max} + t)$, for some $t > 0$, outputs an optimal solution.

So, it only remain to show $|active| \geq k$. We prove that by induction, showing the following property. Let $i = l\Delta + 1$ be an index for some integer l such that $0 \leq l \leq k' - 1$. If $\hat{w}_0 = -(k'-1)c'_{max}$, $\sum_{j=\max\{1, i-\Delta+1\}}^{i-1} \hat{w}_j \leq lc'_{max}$ and no constraint in $\{(l-1)\Delta + 2, \ldots, l\Delta\}$ is active, then constraints $l\Delta + 1, (l+1)\Delta + 1, \ldots, (k'-1)\Delta + 1$ are active. The induction is applied in a downward fashion on l, i.e. the base case is $l = k' - 1$, and our goal is to show it holds for $l = 0$.

Base of induction: $l = k' - 1$**.** Let $i = (k'-1)\Delta + 1$. As $\sum_{j=\max\{1, i-\Delta+1\}}^{i-1} \hat{w}_j \leq (k'-1)c'_{max}$ and $c'_i \geq 0$, we have $\sum_{j=\max\{1, i-\Delta+1\}}^{i-1} \hat{w}_j \leq \hat{w}_0 + c'_i$, and hence constraint i is tight. Furthermore, as no constraint in $\{i - \Delta + 1, \ldots, i-1\}$ is active, constraint i is an active one.

Inductive step: $0 \leq l < k' - 1$**.** Let $i = l\Delta + 1$. First, if $\sum_{j=\max\{1, i-\Delta+1\}}^{i-1} \hat{w}_j \leq lc'_{max}$ we have that constraint i is tight, and as before active as well, and also we have $\hat{w}_i \geq -\hat{w}_0 - lc'_{max}$.

Next, we want to show that $\sum_{j=i+1}^{i+\Delta} \hat{w}_j \leq (l+1)c'_{max}$ (note that $i + \Delta \leq d'$), so that we can use the inductive hypothesis for $l + 1$. First, we show that for every index i' it holds

$$\sum_{j=\max\{1, i'-\Delta+1\}}^{i'} \leq c'_{max} - \hat{w}_0. \tag{6.6}$$

Recall that $\hat{w}_0$ is negative. Towards a contradiction, assume that there exists index q such that $\sum_{j=\max\{1,q-\Delta+1\}}^{q} > c'_{max} - \hat{w}_0$. In case of tie, let q be the smallest such index, which implies $\hat{w}_q > 0$. But then, it contradicts the greedy choice of Dual-Greedy algorithm, as by the greedy choice we have $\hat{w}_q = 0$ or $\sum_{j=\max\{1,q-\Delta+1\}}^{q} = c'_q - \hat{w}_0 \leq c'_{max} - \hat{w}_0$.

Now we combine (6.6), for $i' = i + \Delta - 1$ and $\hat{w}_i \geq -\hat{w}_0 - l c'_{max}$ to obtain

$$\sum_{j=i+1}^{i+\Delta} \hat{w}_j \leq c'_{max} - \hat{w}_0 - \hat{w}_i \leq (l+1)c'_{max},$$

as desired. In addition, as constraint i is active, we have that no constraints in $\{i+1,\ldots,i+\Delta-1\}$ is active, and hence we can use the inductive hypothesis.

This concludes the proof. □

A proof of Lemma 6.5

In this section we finalize the proof of the correctness of our randomized algorithm. Before we delve into details, we introduce some notation. In what follows, we will be interested in cost vectors and sparsity parameters other than c and k, respectively. Hence, whenever this is the case, we will write $\mathscr{P}(c', k')$ to refer to the program $\mathscr{P}$ for cost vector c' and sparsity k'. Similarly, whenever we consider some different cost vector c' and sparsity parameter k', we will denote the corresponding dual LP by $\mathscr{D}(c', k')$.
Also, as pointed out in other sections, our running-time results are given with respect to γ, where γ is the maximal number of bits needed to store any c_i. As $\mathscr{P}$ and $\mathscr{D}$ are invariant under shifting and multiplication of c, for the sake of clarity of our exposition and without loss of generality we assume c is an integral vector with non-negative entries.

Lemma 6.5. *Algorithm* LASSP *runs only a single iteration with probability at least* $1 - 1/d$.

Proof. As pointed out already, without loss of generality in this proof we assume that c is an integral non-negative vector. We also recall that we showed equivalence between problem (6.1) and $\mathscr{P}$, and also between $\mathscr{P}_{\text{no-}k}$ and PROJLAGR, so in this proof we work with the LP formulations.

The choice of λ is optimal. Consider $\tilde{w}$ as in Lemma 6.16. First we want to show that if such $\tilde{w}$ exists, then λ obtained at line 4 of Algorithm 13 is such that it also defines k active constraints. This in turn would imply, by Lemma 6.16, that support $\hat{S}$ obtained at line 5 has cardinality k, and hence is an optimal solution to $\mathscr{P}(\tilde{c})$.

If $\tilde{w}_0 = \lambda$, then we are done. Otherwise, assume that $\tilde{w}_0 \neq \lambda$. By Lemma 6.17, $\tilde{w}$ is an optimal solution of $\mathscr{D}(\tilde{c})$. On the other hand, as we discussed in Section 6.4.4, λ is such that $\text{VAL}\mathscr{D}_\lambda(\tilde{c}) = \text{VAL}\mathscr{D}(\tilde{c})$, and $\tilde{w}_0 < \lambda$ by the choice of λ. Therefore, by Lemma 6.17 it holds that Dual-Greedy$(\tilde{c}, \lambda)$ defines at least k active constraints. Furthermore, as $\mathscr{D}(\tilde{c})$ is convex w.r.t. to the variable w_0, then for every $\lambda' \in [\tilde{w}_0, \lambda]$ we have $\text{VAL}\mathscr{D}_{\lambda'}(\tilde{c}) = \text{VAL}\mathscr{D}(\tilde{c})$. In other words, $\text{VAL}\mathscr{D}_{\lambda'}(\tilde{c})$ remains constant over the given interval. Hence, from Lemma 6.20 we conclude that Dual-Greedy$(\tilde{c}, \lambda)$ defines exactly k active constraints.

However, for $\tilde{c}$ given on the input there might not exists any λ such that Dual-Greedy$(\tilde{c}, \lambda)$ defines exactly k active constraints. Our goal is to show that the randomization we apply as-

sures that for the obtained $\tilde{c}$ it is always the case that there is some λ so that Dual-Greedy$(\tilde{c}, \lambda)$ has exactly k active constraints.

The evolution of active constraints. Now we want to show that the randomization we apply will result in an existence of $\tilde{w}$ as described in Lemma 6.16. We start by studying the evolution of active constraints defined by the output of Dual-Greedy$(\tilde{c}, w_0)$ as w_0 decreases.

First, recall that by Lemma 6.19 we have that if a constraint becomes tight with respect to some $\hat{w} \leftarrow$ Dual-Greedy$(\tilde{c}, \lambda)$, it remains tight with respect to $\hat{w}' \leftarrow$ Dual-Greedy$(\tilde{c}, \lambda')$ for every $\lambda' \leq \lambda$. Let T denote the set of tight constraints with respect to $\hat{w}$, and T' with respect to $\hat{w}'$. By our discussion $T \subseteq T'$.
Let A and A' be the set of active constraints with respect to Dual-Greedy$(\tilde{c}, \lambda)$ and Dual-Greedy$(\tilde{c}, \lambda')$, respectively. Clearly $A \subseteq T$ and $A' \subseteq T'$. We claim that $|T \cap A'| \leq |A|$. Observe that A is a minimum set of constraints so that every tight constraint is covered (covered in the natural way). However, A is also a maximum set of constraints of T that can be chosen so that no two of them overlap, i.e. so that every two of them are at least Δ apart. That means if one would choose a subset of T larger than $|A|$ then some two constraints would overlap. Hence, such a subset can not consist of only active constraints, and therefore $|T \cap A'| \leq |A|$.

This implies that if the number of active constraints increases by $a > 0$ at certain point, then there are at least some a constraints that became active, but also tight, for the first time. Now we want to study what is the probability that two or more non-tight constraints become tight with respect to Dual-Greedy$(\tilde{c}, \lambda)$, for any λ.

Let us focus on a single constraint j. Fix randomness of all the X_i for $i \neq j$, i.e. fix $\tilde{c}_i$ for all $i \neq j$. For $X_j = 0$, there are at most $d-1$ different values of w_0 when any of those constraints becomes tight for the first time. Let W denote the set of these w_0 values. So $|W| < d$. Now, construct set W_j as follows. For each $\lambda \in W$:

- If constraint j is already tight with respect to Dual-Greedy$(\tilde{c}, \lambda)$ for $X_j = 0$, do nothing.
- Define $\hat{c}_i = \tilde{c}_i$ for $i \neq j$, and $\hat{c}_j = \tilde{c}_j + x_\lambda$, where x_λ is defined as the least value so that constraint j becomes tight for the first time with respect to Dual-Greedy$(\hat{c}, \lambda)$. Observe that $x_\lambda \geq 0$ and x_λ is integral. If $x_\lambda < d^3$, add x_λ to W_j.

We have $|W_j| \leq |W| < d$. Each of the value of W_j correspond to some value of X_j. Also, observe that for given λ, a constraints can become tight for the first time for at most one value of X_j. In addition, as long as constraint j is not tight it does not affect when the other constraints will become tight, as $w_j = 0$ so w_j has no affect on other constraints. This all implies that there are at most $d-1$ distinct values of X_j, out of d^3 of them, when constraint j and some other constraint become tight. Therefore,

$$Pr\left[\text{constraint } j \text{ and any other constraint become tight for the same value of } w_0 = \lambda\right] < \frac{d}{d^3} = \frac{1}{d^2}.$$

Now we can apply union bound to conclude

$$Pr\left[\text{two constraints become tight for the same value of } w_0 = \lambda\right] < d\frac{1}{d^2} = \frac{1}{d}.$$

Therefore, after applying randomness, for every value of $w_0 = \lambda$ at most one constraint becomes tight with probability at least $1 - 1/d$. Following our discussion above, this in turn implies that the number of active constraints increases by at most 1 after decreasing the value of w_0 by 1. Therefore, there is $\tilde{w}$ as in Lemma 6.16 with probability $1 - 1/d$ at least.

Required randomness. Every c_i we perturb by one out of d^3 different values, for which $O(\log d)$ random bits suffices. Therefore, in total we need $O(d \log d)$ random bits.

This concludes the proof. □

6.5 Dynamic Programming

Before we introduce our dynamic programming (DP) algorithm for the separated sparsity problem, we briefly review a variant of the DP given in [FMN15]. We remark that the variant below has a time complexity of $O(dk)$, which is already an improvement over the $O(d^2)$ of [FMN15] when $k = o(d)$. However, the authors are not aware of a work the describes the $O(dk)$ DP approach, and hence refer to it as folklore. We improve the analysis of this DP further and give a faster variant that runs in time $O(k(d-(k-1)\Delta))$. In the regime where where the slack $d/k - \Delta$ is constant, the running time simplifies to $O(d+k^2)$. So for the very sparse case $k \in \widetilde{O}(\sqrt{d})$, our improved DP already achieves a nearly-linear running time.

6.5.1 The Basic Dynamic Programming

The folklore dynamic programming fills the following table $DP[i][j]$. Value $DP[i][j]$ can be interpreted as follows: Maximal value of choosing j coordinates of c with positions in $1 \dots i$, such that any two chosen coordinates are at distance Δ at least.

$$\begin{aligned} DP[i][0] &= 0 \\ DP[i][j] &= -\infty, && \text{if } i < 1 \\ DP[i][j] &= \max\left\{DP[i-1][j], c_i + DP[i-\Delta][j-1]\right\}, && \text{otherwise} \end{aligned}$$

The following claim follows easily:

Theorem 6.23: Folklore

$DP[n][k]$ is an optimal value to the Δ-separated problem and can be computed in time $O(dk)$. The elements that constitute the optimal value can be output from DP in time $O(d+k)$.

6.5.2 An Improved Dynamic Programming

The dynamic programming outlined above already achieves an improvement over the algorithms in [HDC09, FMN15]. However, the dynamic programming is still wasteful with the state space it considers. Consider case in which there is only one possible configuration, i.e. $d = (k-1)\Delta + 1$ for some $k, \Delta \geq 1$. Regardless of the cost vector c, an input with such parameters has only one valid solution. Nevertheless, the natural implementation of the dynamic program above runs in $\Theta(dk)$ time. Note that the definition of DP does not take into account the *mandatory distance* required by the separated sparsity model. By mandatory we refer to Δ distance between any two chosen coordinates of c. This observation gives raise to new dynamic programming definition, that we denote by $\overline{DP}$, which directly implements mandatory distances. Let

$$s \stackrel{\text{def}}{=} d - ((k-1)\Delta + 1),$$

which can be seen as the *slack* distance that is not mandatory. For instance, if we choose two coordinates of c with indices i and $i+\Delta+5$, and no other coordinate between them, then we say that Δ distance between them is mandatory, and the remaining distance of 5 is slack. Now, $\overline{DP}$ is defined as

$$\begin{aligned}
&\overline{DP}[i][0] = 0 \\
&\overline{DP}[i][j] = -\infty, && \text{if } i < 0 \\
&\overline{DP}[i][j] = \max\left\{\overline{DP}[i-1][j], c_{pos(i,j)} + \overline{DP}[i][j-1]\right\}, && \text{otherwise}
\end{aligned}$$

where $pos(i,j) = d-(s-i)-(k-j)\Delta$. Value $pos(i,j)$ gives us the smallest index of c that we can chose if to the left of that index there are j other chosen indices, and slack distance of i is used. With the definition of $\overline{DP}$ in hands one can easily show the following theorem.

Theorem 6.24

Value $\overline{DP}[s][k]$ is an optimal value to the Δ-separated problem and can be computed in time $O(ks)$. The elements that constitute the optimal value can be output from $\overline{DP}$ in time $O(d+k)$.

Comparing Theorem 6.23 and Theorem 6.24, we see that for certain input parameters, e.g. when $s \in O(d)$, we did not make much progress by devising $\overline{DP}$. However, as we have already mentioned, the setting we care the most is when $d/k-\Delta$ is a constant, as with such a choice of parameters the sample complexity is $O(k)$. But in that case we do achieve a significant improvement over Theorem 6.23.

Corollary 6.25. *If $d/k-\Delta \in O(1)$, then there is an algorithm that solves the Δ-separated problem in $O(k^2+d)$ time.*

6.6 Experiments

We empirically validate the claims outlined in the previous sections. To that end, we compare LASSP with the $O(dk)$-time dynamic program (DP) described in Section 6.5 as a baseline. Note that this DP already has a better time complexity than the best previously published algorithm from [FMN15]. Both algorithms are implemented in the Julia programming language (version 0.5.0), which typically achieves performance close to C/C++ for combinatorial algorithms.

6.6.1 Synthetic Data

We perform experiments with synthetic data in order to investigate how the algorithms scale as a function of the input size. We study two different setups: (i) the running time of the projection algorithms on their own, and (ii) the overall running time of a sparse recovery algorithm using the projection algorithms as a subroutine. For the latter, we use the structure-aware variant of the popular CoSaMP algorithm [NT09, BCDH10].

Figures 6.3(a)-(b). Given a problem size d, we set the sparsity to $k = d/50$ and generate a random separated sparse vector with parameter $\Delta = (d-5(k+1))/k-1$. The non-zero

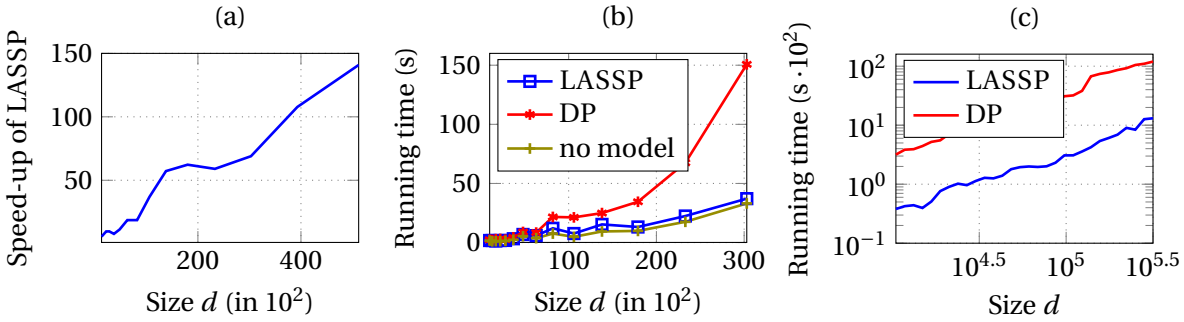

Figure 6.3 – Separated sparsity experiments on synthetic data. We plot running times of our algorithm LASSP relative to the previous work. Plots (a) and (c) are obtained by projecting signals on the separated sparsity model. Plot (b) compares the running times of CoSaMP with three different projection operators. The variant "no-model" makes no structural assumptions and uses hard thresholding as projection operator.

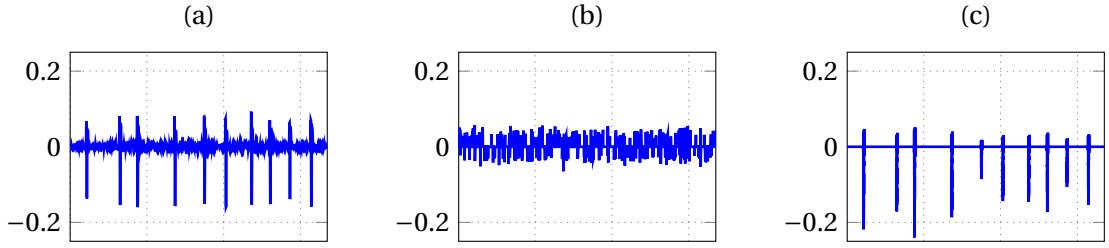

Figure 6.4 – Separated sparsity experiments for a recovery of real signals. In (b) we plot the recovery of the signal (a) obtained by the algorithm "no model", i.e. by the algorithm that makes no structural assumptions and uses hard thresholding as projection operator. Plot (c) shows the recovery of LASSP. The both procedures use the same number of measurements.

coefficients are i.i.d. ± 1. For the projection-only benchmark, we add Gaussian noise with $\sigma = 1/10$ to all coordinates in order to make the problem non-trivial. For each problem size, we run 10 independent trials and report their mean.

Figure 6.3(a) shows the speed-up obtained by our nearly-linear time projection relative to the DP baseline. We observe that LASSP is up to 150× faster. This confirms our expectation that LASSP scales gracefully with the problem size, while the DP essentially becomes a quadratic-time algorithm.

Figure 6.3(b) compares the running times of CoSaMP with three different projection operators. The first variant makes no structural assumptions and uses hard thresholding as projection operator. The other two variants use a projection for the separated sparsity model, relying on the DP baseline and LASSP, respectively. The results show that the version of CoSaMP using LASSP instead of the DP is significantly faster. Moreover, CoSaMP with a simple sparse projection has similar running time to CoSaMP with our structured projection. We note that CoSaMP with separated sparsity requires 1.5× fewer measurements to achieve the same recovery quality as CoSaMP with standard sparsity.

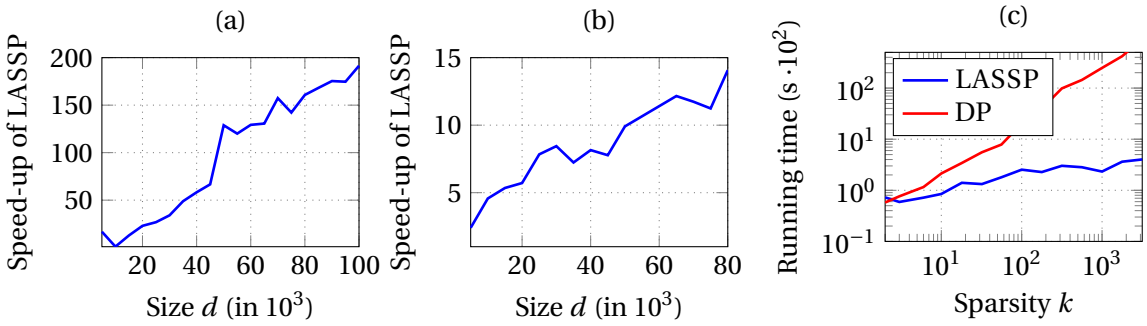

Figure 6.5 – The plots compare the running times of our algorithm LASSP and the DP baseline. Plots (a) and (c) are obtained by projecting signals directly on the separated sparsity model. Plot (b) shows the speed-up of CoSaMP recovery obtained by using LASSP as a projection operator relative to using the DP for projection. Plots (a) and (b) are run on real signals, obtained by the concatenation of the signal given in Figure 6.4(a).

Figure 6.3(c). We fix the sparsity $k = 100$ and vary the length d of the signal. The signal is obtained in the same way as for plots Figure 6.3(a)-(b). We observe that our algorithm runs 10x faster than the baseline. Furthermore, the plot shows that the running times of both the baseline DP and LASSP scale linearly with the signal length d. This behavior is expected for the DP. On the other hand, our theoretical findings predict that the running time of LASSP scales as $d \log d$. This suggests that the empirical performance of our algorithm is even (slightly) better than what our proofs state.

For the plot Figure 6.5(c) we make the setup similar to the one for Figures 6.3(a)-(b). Namely, given a problem size d, sparsity to k, we generate a random separated sparse vector with parameter $\Delta = (d - 5(k+1))/k - 1$. The non-zero coefficients are i.i.d. ± 1. We add Gaussian noise with $\sigma = 0.5$ to all coordinates in order to make the problem non-trivial. For each problem size, we run 10 independent trials and report their mean. In Figure 6.5(c) we compare the running times of projections of the baseline DP and our algorithm LASSP on synthetic data for $d = 5 \cdot 10^4$ and varying the sparsity k. This plot confirms our theoretical findings that the running time of LASSP scales more gracefully with the growth of the sparsity than the running time of DP does.

6.6.2 Neuronal Signals

We also test our algorithm on neuron spike train data from [HB11]. See Figure 6.4(a) for this input data. First, we run CoSaMP with a "standard sparsity" projection. The recovered signal is depicted in Figure 6.4(b). Next, we run the convolutional sparsity CoSaMP of [HB11] and use our fast projection algorithm. The recovered signal is given in Figure 6.4(c). For the both experiments we use $n = 250$ measurements.

We also run the convolutional sparsity CoSaMP on neuron spike train data of length 10^5, comparing the running time of LASSP and DP as projection operators. CoSaMP with our algorithm runs $2\times$ faster in this context. We do not compare the running time relative to CoSaMP with a standard sparsity projection as it requires $10\times$ more measurements to achieve accurate recovery.

In plots Figure 6.5(a)-(b) we run experiments on real signals. The signals are obtained from signal in Figure 6.4(a) by concatenating its copies. Figure 6.5(a) is obtained by projecting the signal directly to the separated sparsity model. Then we plot the ratio of the running times of the baseline DP and our algorithm LASSP. Figure 6.5(b) represents the same type of speed-up ratio but this time for the CoSaMP recovery of signal measurements.

6.7 Deterministic Approach

In this section we describe our deterministic nearly-linear time algorithm. For the sake of clarity, we repeat some of the notation presented in earlier sections.

6.7.1 Overview

Our algorithm stems from a linear programming view on the separated sparsity recovery. It has already been shown that this LP is totally unimodular [HDC09], which implies that solving the LP provides an integral solution and hence solves separated sparsity recovery. However, the previous work resorts to a black-box approach for solving this LP leading to a prohibitive $O(d^{3.5})$ time complexity. We make a step forward, and study the dual LP, obtaining a method that takes nearly-linear time to find its optimal solution. By the strong duality, the value of the dual is the value of the primal LP as well. However, unfortunately, it is not hard to see that a generic relation between primal and dual LP solutions known as "complementary slackness" does not lead to recovery of the primal optimal solution itself.

To cope with that shortcoming, we further analyze the properties of dual solution. We exhibit very close connection between its structure and the sparsity of the primal solution, which we present via the notion of "active constraints". Intuitively, this structure allows us to characterize the cases in which the complementary slackness in fact provides the primal from a dual optimal solution. Then we use these findings in our algorithm to slightly perturb the input instance, while not affecting the value of the solution, so that it is possible to obtain a primal from a dual solution in the general case, and hence solve the separated sparsity recovery.

In Section 6.4.4 we defined LP $\mathscr{P}$, which corresponds to the problem (6.3), as follows:

$$
\begin{aligned}
\text{maximize} \quad & c^T u \\
\text{subject to} \quad & \sum_{i=1}^{d} u_i = k \\
& \sum_{j=i}^{\min\{i+\Delta-1,n\}} u_j \le 1 \qquad \forall i = 1 \dots d \\
& u_i \ge 0 \qquad \forall i = 1 \dots d
\end{aligned}
$$

We also defined the dual LP to $\mathscr{P}$, denoted by $\mathscr{D}$, as

$$\begin{aligned}
\text{minimize} \quad & w_0 k + \sum_{i=1}^{d} w_i & & \\
\text{subject to} \quad & w_0 + \sum_{\substack{j \,:\, j \geq 1 \text{ and} \\ j \leq i \leq j+\Delta-1}} w_j \geq c_i & & \forall\, i = 1 \ldots d \\
& w_i \geq 0 & & \forall\, i = 1 \ldots d \\
& w_0 \in \mathbb{R} & &
\end{aligned}$$

As Theorem 6.13, which proofs appears in Section 6.4.5, this dual LP has a combinatorial structure that enables us to solve it in nearly-linear time.

Before delving into details, we introduce some notation. In what follows, we will be interested in cost vectors and sparsity parameters other than c and k, respectively. Hence, whenever this is the case, we will write $\mathscr{P}(c', k')$ to refer to the program $\mathscr{P}$ for cost vector c' and sparsity k'. Similarly, whenever we consider some different cost vector c' and sparsity parameter k', we will denote the corresponding dual LP by $\mathscr{D}(c', k')$.
As already mentioned, our running-time results are given with respect to γ, where γ is the maximal number of bits needed to store any c_i. As $\mathscr{P}$ and $\mathscr{D}$ are invariant under shifting and multiplication of c, for the sake of clarity of our exposition and without loss of generality we assume c is an integral vector with non-negative entries.

At a high-level, we develop our algorithm in three main steps. In Section 6.4.4 we considered the dual of the LP $\mathscr{P}$ and show that its combinatorial structure can be exploited to compute an optimal solution to it in nearly-linear time. By strong duality, this optimal dual solution gives us then the *value* of the optimal primal solution. But, unfortunately, it does not give us the optimal primal solution itself.

To alleviate this issue, we develop a divide-and-conquer procedure for extracting that optimal primal solution. First, in Section 6.7.2, we demonstrate that by analyzing answers of the dual oracle on perturbed versions of the original problem we can quickly recover a single non-zero entry of the optimal primal solution. That entry can be used to partition our instance into two smaller subproblems. Then, in Section 6.7.3, we show how to make these two subproblems fully independent by devising an optimal split of the sparsity constraint that they share. This optimal split is extracted from the structure of the dual solutions.

With these components in place, we assemble our final algorithm in Section 6.7.4.

6.7.2 Recovering a Segment of an Optimal Solution

At this point, we developed a way of computing the optimal value OPT of $\mathscr{P}$. However, this is not sufficient for our purposes as the separated sparsity problem also requires us to provide the corresponding solution, i.e., a binary vector $u^\star$ corresponding to OPT.

One might hope that this primal solution $u^\star$ can be inferred from the dual solution $\hat{w}$ that our algorithm Opt-Value-of-$\mathscr{D}(c', k')$ (see Algorithm 14) provides. It is not hard to see, however, that the generic relationship between the optimal primal and optimal dual solutions that the so-called "complementary slackness" provides is not sufficient here.

Therefore, we instead design a problem-specific algorithm for finding the desired primal solution vector $u^\star$. As a first step, we focus on the task of recovering a single segment of $u^\star$.

Specifically, given some target segment S of at most Δ consecutive entries, we want to either find a *single* index $i \in S$ such that $u_i^\star = 1$, for some optimal primal solution $u^\star$; or to conclude that $u_i^\star = 0$ for all $i \in S$.

To recover such a segment, we define in an adaptive manner a family of cost vectors $c^1, \ldots, c^t$, for some $t \in O(\log \Delta)$ and invoke our dual LP solver Opt-Value-of-$\mathscr{D}(c', k')$ on these cost vectors. As we show, the solutions to these perturbed instances allow us to infer $u_j^\star$ for all $j \in S$.

To provide more details, let us fix some Δ-length segment $S = [j_s, j_e]$, i.e., $j_e = \min\{j_s + \Delta - 1, n\}$. We now want to decide whether there is an optimal solution $\hat{u}$ to $\mathscr{P}$ such that $\hat{u}_i = 1$ for some $i \in [j_s, j_e]$. Observe that there might be another optimal solution $\tilde{u}$ to $\mathscr{P}$. Furthermore, it might be the case that $\tilde{u}_j = 0$ for every index $j \in [j_s, j_e]$, while there is an index $i \in [j_s, j_e]$ such that $\hat{u}_i = 1$. So while designing the algorithm, we distinguish two cases. First, we analyse the case in which there exists an optimal solution $\hat{u}$ to $\mathscr{P}$ so that $\hat{u}_i = 0$ for every $i \in [j_s, j_e]$. Next, we consider the complementary case in which for every optimal solution $\hat{u}$, we have that $\hat{u}_i = 1$ for some $i \in [j_s, j_e]$. In the latter case, we recover an index j such that there is a solution $\hat{u}$ for which $\hat{u}_j = 1$ holds.

The first case is captured by the following claim.

Lemma 6.26. *Let $S \subseteq [d']$ be a set of indices, let $c' \in \mathbb{N}_+^{d'}$ be a coefficient vector, and let k' be the sparsity. Define a vector $c'' \in \mathbb{N}_+^{d'}$ as follows:*

$$c_i'' = \begin{cases} 1 & \text{if } i \in S \\ 1 + c_i' & \text{otherwise} \end{cases}.$$

Then, Opt-Value-of-$\mathscr{D}(c'', k')$ equals Opt-Value-of-$\mathscr{D}(c', k') + k'$ iff there exists an optimal solution $u^\star$ to $\mathscr{P}(c', k')$ such that $u_i^\star = 0$ whenever $i \in S$.

Proof. Let us show the two direction of equivalence separately.

($\Longrightarrow$) Assume that Opt-Value-of-$\mathscr{D}(c'', k')$ equals Opt-Value-of-$\mathscr{D}(c', k') + k'$. Let $\hat{u}$ be an optimal solution to $\mathscr{P}(c', k')$ for the cost vector given by c''. Now we want to show that $\hat{u}_i = 0$ for every $i \in S$. Towards a contradiction, assume it is not the case, i.e. there exists $j \in S$ such that $\hat{u}_j = 1$. But then, $\sum_{i\,:\,\hat{u}_i = 1}(c_i + 1) > \sum_{i\,:\,\hat{u}_i = 1} c_i''$ as $c_i' + 1 > c_i''$ for $i \in S$, and hence Opt-Value-of-$\mathscr{D}(c'', k') <$ Opt-Value-of-$\mathscr{D}(c', k') + k'$, contradicting our assumption.

($\Longleftarrow$) First, observe that Opt-Value-of-$\mathscr{D}(c'', k') \leq$ Opt-Value-of-$\mathscr{D}(c', k') + k'$ as $c'' \leq c' + 1$. On the other hand, if there exists an optimal solution $u^\star$ to $\mathscr{P}(c', k')$ such that $u_i^\star = 0$ whenever $i \in S$, then $u^\star$ achieves value Opt-Value-of-$\mathscr{D}(c', k') + k'$ in $\mathscr{P}(c', k')$ for the costs given by c''. In other words, restricted to the set of indices outside of S, c' is a shifted by 1 variant of c''. Therefore, we also have Opt-Value-of-$\mathscr{D}(c'', k') \geq$ Opt-Value-of-$\mathscr{D}(c', k') + k'$. Now this implies Opt-Value-of-$\mathscr{D}(c'', k') =$ Opt-Value-of-$\mathscr{D}(c', k') + k'$, as desired. □

We are now ready to prove the following result.

Algorithm 17: Δ-recovery(c', k', j_s)

Input:
- $c' \in \mathbb{N}_+^{d'}$
- Sparsity k'
- Index j_s

Output: index r as described in Theorem 6.27

1 $j_e \leftarrow \min\{j_s + \Delta - 1, d'\}$
2 $s \leftarrow j_s$; $e = j_e$; $OPT \leftarrow$ Opt-Value-of-$\mathscr{D}(c', k')$
3 $c''_i \leftarrow \begin{cases} 1 & \text{if } i \in [j_s, j_e] \\ 1 + c'_i & \text{otherwise} \end{cases}$
4 **if** *Opt-Value-of-*$\mathscr{D}(c'', k') = OPT + k'$ **then**
5 **return** *-1*
6 **while** $s < e$ **do**
7 $mid \leftarrow \lfloor \frac{s+e}{2} \rfloor$
8 $c''_i \leftarrow \begin{cases} 1 & \text{if } i \in [j_s, j_e] \setminus [s, mid] \\ 1 + c'_i & \text{otherwise} \end{cases}$
9 **if** *Opt-Value-of-*$\mathscr{D}(c'', k') = OPT + k'$ **then**
10 $e \leftarrow mid$
11 **else**
12 $s \leftarrow mid + 1$
13 **return** s

Theorem 6.27

There exists an algorithm that given $c' \in \mathbb{N}_+^{d'}$, sparsity k', and an index $j_s \in [1, d']$, outputs an integer r having the following properties:

- If for every optimal solution $\hat{u}$ to $\mathscr{P}(c', k')$ there is an index i such that $\hat{u}_i = 1$ and $i \in [j_s, \min\{j_s + \Delta - 1, d'\}]$, then r is set to be an index in $[j_s, \min\{j_s + \Delta - 1, d'\}]$ such that there is an optimal solution $u^\star$ for which we have $u^\star_r = 1$.
- Otherwise, r is set to -1.

Furthermore, if the algorithm runs in time $O\big((\log \Delta) d' (\log(c'_{max} + 1) + \log k')\big)$.

Proof. Algorithm Δ-recovery (see Algorithm 17) is used in the proof of this theorem.

On line 4, Δ-recovery first checks whether we can simply ignore all the entries indexed by $\{i, \ldots, \min\{i + \Delta - 1, d'\}\}$. And if yes, it returns -1. (This step is formalized in the statement of Theorem 6.27.) However, entries from those interval can only be disregarded if there exists an optimal solution $u^\star$ to $\mathscr{P}(c', k')$ such that for every $j \in \{i, \ldots, \min\{i + \Delta - 1, d'\}\}$ we have $u^\star_j = 0$. So, if there is no such $u^\star$, i.e. the entries corresponding to that interval have to be considered, the lines following line 4 of Δ-recovery serve to pinpoint an entry $j \in \{i, \ldots, \min\{i + \Delta - 1, d'\}\}$ so that there exists an optimal solution $\hat{u}$ to $\mathscr{P}(c', k')$ for which holds $\hat{u}_j = 1$. The way it is done is by applying a binary search over the interval that *can not* be ignored, tracked via variables s

and e of the algorithm.

Correctness. Algorithm Δ-recovery outputs -1 correctly by Lemma 6.26 for $S = \{j_s, \ldots, j_e\}$.

Next, we show that when the binary search loop starting at line 6 ends we have $s = e$, i.e. $[s, e]$ corresponds to a single index. As long as $\Delta \geq 1$ and $j_s \leq d'$, at line 1 and line 2 values s and e are initialized so that $s \leq e$, so initially $[s, e]$ is indeed a non-empty interval. Furthermore, as we have $s < e$ in each iteration, it holds $mid < e$. But it also holds $mid \geq s$ (in fact mid equals s only when $s+1$ equals e). So, updating e to mid at line 9, or s to $mid+1$ at line 12, $[s, e]$ remains a non-empty interval in any iteration. Notice that the update rules also guarantee that we either increase s or decrease e at each iteration, and therefore $[s, e]$ shrinks its size by 1 at least at each iteration. Putting this together, and taking into account the loop-termination condition at line 6, we conclude that after the loop ends it holds $s = e$.

We say that interval $[s, e]$ has property $R_{s,e}$ if there is an optimal solution $\hat{u}$ to $\mathscr{P}(c', k')$ such that $\hat{u}_i = 1$ for some $i \in [s, e]$. Now, if we show that after line 4 at every step of the algorithm interval $[s, e]$ has property $R_{s,e}$, then the proof will follow immediately. That is exactly how we proceed. Namely, we show that if $[s, e]$ has property $R_{s,e}$, then after updating s or e at line 9 or line 12 obtaining s' and e', respectively, then interval $[s', e']$ will have property $R_{s',e'}$.

Observe that at the beginning of the very first iteration of the loop property $R_{s,e}$, equivalent to R_{j_s,j_e}, holds as by Lemma 6.26 *every* optimal solution $\hat{u}$ is such that $\hat{u}_i = 1$ for some $i \in [j_s, j_e]$. Next, let $[s, e]$ be updated to $[s', e']$. We want to show that property $R_{s',e'}$ holds as well.

First, assume that line 9 gets executed, i.e. $s' = s$ and $e' = mid$. Then, by Lemma 6.26, we have that there exists an optimal solution $\hat{u}$ to $\mathscr{P}(c', k')$ such that $\hat{u}_i = 0$ for every $i \in [j_s, j_e] \setminus [s, mid]$. But we also have that there is $i \in [j_s, j_e]$ such that $\hat{u}_i = 1$. So, putting it together, we conclude that property $R_{s',e'}$ holds.

Next, assume that the if condition at line 9 does not hold. So, line 12 is executed, i.e. $s' = mid + 1$ and $e' = e$. Then again by Lemma 6.26, and as a consequence of both line 4 and line 9, we have that for every optimal solution $\hat{u}$ we have $\hat{u}_i = 1$ for some $\in [j_s, j_e] \setminus [s, mid]$. Also, as $j_e - j_s + 1 \geq \Delta$, we also have $\hat{u}_j = 0$ for every $j \in [s, mid]$. But then, as by our assumption there is an optimal solution $\tilde{u}$ and an index i such that $\tilde{u}_i = 1$ and $i \in [s, e]$, we have that $i \in [mid + 1, e]$. Hence property $R_{mid+1,e}$, which is equivalent to $R_{s',e'}$, holds.

Running time. If Δ-recovery outputs -1 at line 4, it invokes Opt-Value-of-$\mathscr{D}$ for the cost vector c' and c''. Note that $c'_{max} \leq c'_{max} + 1$. Then, by Theorem 6.13, this case takes running time $O\big(d'(\log(c'_{max} + 1) + \log k')\big)$.

If the method does not output -1, then it enters the while loop. The loop applies a standard binary search over j_s and j_e which are set so that it holds $j_e - j_s + 1 \leq \Delta$. So, the loop iterates for $O(\log \Delta)$ times. Every iteration invokes Opt-Value-of-$\mathscr{D}$ for vector c'' as defined at line 8, resulting in the total running time of $O((\log \Delta) d'(\log(c'_{max} + 1) + \log k'))$. □

6.7.3 Distributing Sparsity

The algorithm we presented in the previous section enables us to quickly recover a target length-Δ segment of some optimal primal solution $u^\star$. We now would like to build on this procedure to develop a divide-and-conquer approach to recovering the primal solution $u^\star$ in full.

Our intention is to use the procedure from Section 6.7.2 to split our input problem into two (smaller and approximately equally sized) subproblems by recovering a "middle" segment of the solution $u^\star$ and then to proceed recursively on each of these subproblems. The difficulty here, however, is that these two resulting subproblems are not really independent, even though they correspond to separate cost vectors. The subproblems still share the sparsity constraint. That is, to make these subproblems truly independent, we must also specify the split of our sparsity "budget" k' between them.

In this section, we analyze properties of the dual LP and exhibit very close ties between its structure, as captured by active constraints, and the optimal sparsity distribution for our two subproblems. Specifically, we establish the following theorem.

Theorem 6.28

Let $c' \in \mathbb{N}_+^{d'}$, let k'' be a sparsity parameter, and let indices s and e be such that $e-s+1 \geq \Delta$ or e equals d'. Define a vector $c'' \in \mathbb{N}_+^{d'}$ as follows:

$$c''_i = \begin{cases} -\infty & \text{if } i \in [s,e] \\ c'_i & \text{otherwise} \end{cases}$$

Assume that there is a Δ-separated choice of coordinates of c'' such that k'' coordinates are chosen, and none of them has an index in $[s,e]$, i.e., the instance is feasible when restricted to coordinates outside of $[s,e]$. Then, there is an algorithm that outputs k'_L and k'_R with the property that there exists an optimal Δ-separated solution for cost vector c'' and sparsity k'' such that it chooses k'_L coordinates of c'' with indices less than s, and k'_R coordinates of c'' with indices greater than e.

The algorithm runs in time $O\big(d'(\log c'_{max} + \log k'')\big)$.

Observe that Theorem 6.28 essentially gives an algorithm that distributes sparsity in an optimal way and thus enables us to implement the desired divide-and-conquer approach.

We now illustrate how to use the algorithms Δ-recovery (Algorithm 17) and Active-Constraints (Algorithm 16) to recover an optimal solution for the example $c = (3,2,1,4,1,1,2,1)$, $k = 4$ and $\Delta = 2$. Let $u^\star$ be an optimal solution to $\mathscr{P}$. As described above, we proceed by splitting c into two subvectors and solve the separated sparsity problem on each of them independently. The first step is to find the exact splitting point in c. To achieve this, we invoke Δ-recovery$(c,4,k)$. As $c_4 = 4$ is part of any optimal solution, the method returns index 4 and hence we have $u^\star_4 = 1$. So at this moment we have learned one index of the optimal solution. In fact, it also implies that $u^\star_3 = u^\star_5 = 0$. Hence it remains to learn the remaining $k-1$ indices that define $u^\star$. However, we have to distribute the remaining sparsity $k-1$ over the two subproblems. To that end, we define a new vector c' as follows

$$c' = \begin{cases} -\infty & \text{if } i \in \{3,4,5\} \\ c_i & \text{otherwise.} \end{cases}$$

The definition of c' enforces that no optimal solution to the separated sparsity problem for sparsity $k-1$ will choose c_3, c_4, or c_5 (setting c'_i to the dummy value $-\infty$ is for illustration purposes only). Next, we invoke Active-Constraints$(c',$Opt-Value-of-$\mathscr{D}(c',k-1))$ and denote

its output by $active'$. Observe that, as long as Opt-Value-of-$\mathscr{D}(c,k)$ has finite value, an optimal solution to Opt-Value-of-$\mathscr{D}(c',k-1)$ is finite as well. We have that $active' = \{1,6,8\}$. Now we count the number of elements in $active'$ that are "to the left" and "to the right" of $u_4^\star$, denoting these quantities by $\hat{k}$ and $\tilde{k}$ respectively, i.e.,

$$\hat{k} = |\{i \in active' \mid i < 4\}|, \quad \text{and} \quad \tilde{k} = |\{i \in active' \mid i > 4\}| .$$

In a similar way, we split the vector c' into $\hat{c}$ and $\tilde{c}$ as follows:

$$\hat{c} = (c_1, c_2), \text{ and } \tilde{c} = (c_6, c_7, c_8) .$$

Finally, we solve two Δ-separated instances independently, one for $\hat{c}$ and $\hat{k}$, and the second one for $\tilde{c}$ and $\tilde{k}$.

Proof of Theorem 6.28. In this proof we analyze algorithm Distribute-Sparsity (see Algorithm 18). This algorithm is used to distribute the sparsity.

Algorithm 18: Distribute-Sparsity(c', k'', s, e)

Input:
- $c' \in \mathbb{N}_+^{d'}$
- Sparsity k'' to be distributed
- Indices s and e as described in Theorem 6.28

Output: sparsity k'_L for the left side, sparsity k'_R for the right side

1 $shift \leftarrow 1 + (k'' c'_{max} + 1)$
2 $c''_i \leftarrow \begin{cases} 1 & \text{if } i \in [s,e] \\ c'_i + shift & \text{otherwise} \end{cases}$
3 $w^1 \leftarrow$ Opt-Value-of-$\mathscr{D}(c'', k'', -(k''-1)c'_{max} + shift, c'_{max} + shift)$
4 $active^1 \leftarrow$ Active-Constraints(c'', w^1)
5 $active^2 \leftarrow$ Active-Constraints$(c'',$ Dual-Greedy$(c'', w_0^1 + 1))$
6 $k_L^1 \leftarrow |\{i \in active^1 \mid i < s\}|$
7 $k_L^2 \leftarrow |\{i \in active^2 \mid i < s\}|$
8 $k_R^2 \leftarrow |\{i \in active^2 \mid i > e\}|$
9 $k'_L \leftarrow k_L^2 + \min\{k_L^1 - k_L^2, k'' - k_L^2 - k_R^2\}$
10 $k'_R \leftarrow k'' - k'_L$
11 **return** (k'_L, k'_R)

Optimality for $\mathscr{D}(c'', k'')$. We argue that w^1 defined on line 3 is an optimal solution to $\mathscr{D}(c'', k'')$ although w_0^1 is restricted to belong to interval $[-(k''-1)c'_{max} + shift, c'_{max} + shift] = [c'_{max} + 2, c'_{max} + shift]$. First, observe that for $w_0^1 \geq c'_{max} + 2$ no constraint with index in $[s,e]$ is tight, and hence no such constraint can be active. Furthermore, as no constraints in $[s,e]$ is tight, we have $w_i^1 = 0$ for all $i \in [e,s]$. Now, consider $c_L = (c'_1, \ldots, c'_{s-1})$ and $c_R = (c'_{e+1}, \ldots, c'_{d'})$ as defined in the algorithm. Then, by following the proof of Lemma 6.21, and as $e - s + 1 \geq \Delta$, we conclude that Dual-Greedy$(c'', c'_{max} + 2)$ has at least $\min\left\{k', \left\lceil \frac{s-1}{\Delta} \right\rceil\right\} + \min\left\{k', \left\lceil \frac{n-e}{\Delta} \right\rceil\right\}$ active constraints which, by the construction and the assumption that the input instance is feasible, is k'' at least. So, by the monotonicity of the size of active constraints given by Lemma 6.22 and the optimality condition provided via Lemma 6.17 we have that w^1 is an optimal solution to $\mathscr{D}(c'', k'')$.

Correctness of distributed sparsity. Let $active^1$ denote Active-Constraints(c'', w^1) and $active^2$ denote Active-Constraints(c'', w^2), where $w^2 \leftarrow$ Dual-Greedy$(c'', w_0^1 + 1)$. Next, split w^1 and w^2 as follows. Let w_L^1 be a zero-indexed s-dimensional and w_R^1 be a zero-indexed $(d' - e + 1)$-dimensional vector defined as

$$w_L^1 \leftarrow (w_0^1, w_1^1, \ldots, w_{s-1}^1), \text{ and } w_R^1 \leftarrow (w_0^1, w_{e+1}^1, \ldots, w_{d'}^1).$$

Intuitively, letter 'L' stands for left to s and letter 'R' stands for right to e. Similarly to w_L^1 and w_R^1, define w_L^2 and w_R^2 as

$$w_L^2 \leftarrow (w_0^2, w_1^2, \ldots, w_{s-1}^2), \text{ and } w_R^2 \leftarrow (w_0^2, w_{e+1}^2, \ldots, w_{d'}^2).$$

Also, split $active^1$ and $active^2$ with respect to s and e in the obvious way

$$active_L^1 \leftarrow \{i \in active^1 \mid i < s\}, \text{ and } active_R^1 \leftarrow \{i \in active^1 \mid i > e\},$$

and

$$active_L^2 \leftarrow \{i \in active^2 \mid i < s\}, \text{ and } active_R^2 \leftarrow \{i \in active^2 \mid i > e\}.$$

As a reminder, there is no $i \in [s, e]$ such that $i \in active^1$. Furthermore, as there is no tight constraint in $[s, e]$ for w^1, we have $w_i^1 = 0$ for every $i \in [s, e]$. Observe that the same holds for $active^2$ and w^2. In addition, we have $e - s + 1 \geq \Delta$. From this, we can derive the following list of equalities, which essentially allows us to split the input problem into two independent subproblems. So, we have: Dual-Greedy(c_L, w_0^1) equals w_L^1; Dual-Greedy$(c_L, w_0^1 + 1)$ equals w_L^2; Dual-Greedy(c_R, w_0^1) equals w_R^1; and, Dual-Greedy$(c_R, w_0^1 + 1)$ equals w_R^2. But also, we have: Active-Constraints(c_L, w_L^1) equals $active_L^1$; Active-Constraints(c_L, w_L^2) equals $active_L^2$; Active-Constraints(c_R, w_R^1) equals $active_R^1$; and, Active-Constraints(c_R, w_R^2) equals $active_R^2$.

Now, by Lemma 6.22 we conclude that $|active_L^1| \geq |active_L^2|$ and $|active_R^1| \geq |active_R^2|$. This in turn implies that k'_L and k'_R in Distribute-Sparsity are derived correctly. (Note that $k'_L + k'_R = k''$.) But then, by Lemma 6.17 we have that w_L^1 is an optimal solution to $\mathscr{D}(c_L, k'_L)$ and w_R^1 is an optimal solution to $\mathscr{D}(c_R, k'_R)$, so we can independently solve Recover(c_L, k'_L) and Recover(c_R, k'_R) knowing that the optimal solution, i.e. its value, remains the same. We point out that $w'_L \leftarrow$ Opt-Value-of-$\mathscr{D}(c_L, k_L)$ might differ from w_L^1, and similarly $w'_R \leftarrow$ Opt-Value-of-$\mathscr{D}(c_R, k_R)$ might differ from w_R^1. In fact, w'_L and w'_R might be such that $w'_{L0} \neq w'_{R0}$, although we have that w_{L0}^1 is equal to w_{R0}^1.

Running time. Every line, except maybe line 3, take $O(d')$ time. On the other hand, line 3 takes time $O\big(d' \log\{c'_{max} + shift - (-(k'' - 1)c'_{max} + shift)\}\big)$ that equals $O\big(d'(\log c'_{max} + \log k'')\big)$. □

6.7.4 Separated Sparsity in Nearly-Linear Time

We now have all components in place to state our final algorithm Recover(c', k') that produces an optimal integral solution for $\mathscr{P}(c', k')$ (see Algorithm 19). We prove the following result for our algorithm.

Algorithm 19: Recover(c', k')

Input:
- $c' \in \mathbb{N}_+^{d'}$
- Sparsity k'

Output: a primal solution d'-dimensional vector $u^\star$

1 **if** $k' = 0$ **then**
2 **return**
3 $r \leftarrow \Delta$-recovery$(c', \left\lfloor \frac{d'+1}{2} \right\rfloor, k')$
4 **if** $r = -1$ **then**
5 $s \leftarrow \left\lfloor \frac{d'+1}{2} \right\rfloor, \quad e \leftarrow \min\{s + \Delta - 1, d'\}$
6 $k'' \leftarrow k'$
7 **else**
8 $s \leftarrow \max\{r - \Delta + 1, 1\}, \quad e \leftarrow \min\{r + \Delta - 1, d'\}$
9 $k'' \leftarrow k' - 1$
10 $(k'_L, k'_R) \leftarrow$ Distribute-Sparsity(c', k'', s, e)
11 $c_L \leftarrow (c'_1, \ldots, c'_{s-1}), \quad c_R \leftarrow (c'_{e+1}, \ldots, c'_{d'})$
12 $u^L \leftarrow$ Recover$(c_L, k'_L), \quad u^R \leftarrow$ Recover(c_R, k'_R)
13 $u_i^\star = \begin{cases} u_i^L & \text{if } i < s \\ 1 & \text{if } i \in [s, e] \text{ and } r = i \\ u_{i-e}^R & \text{if } i > e \\ 0 & \text{otherwise} \end{cases}$
14 **return** $u^\star$

Theorem 6.29

The algorithm Recover(c', k') solves the model projection problem for separated sparsity in time $O\left(d(\log(c_{max} + 1) + \log k) \log d \log \Delta\right)$.

Proof. The algorithm Recover(c', k') utilizes Δ-recovery and Distribute-Sparsity to split the input problem into two subproblems and then recurses on them. Let us now analyze the correctness and the running time of this algorithm.

Base case. If the input sparsity is 0, the algorithm outputs a zero-vector at line 1.

Δ-middle entries. Next, the algorithm invokes Δ-recovery over the Δ middle entries. Depending on r, it sets s and e to correspond to the interval of c' that should be removed from consideration in the recursive calls. In the same time, that intervals serves to break the input problem into two independent subproblems. It also sets k'' that represents the sparsity distributed outside interval $[s, e]$. Correctness of this call is guaranteed by Theorem 6.27. Value $shift$, and in turn vector c'', is set so that we have a guarantee that no element from the interval $[s, e]$ is chosen as part of an optimal solution. To see that, consider vector $c^{(3)} \leftarrow c'' - shift \cdot \mathbb{1}$. Then, $c_i^{(3)} = c'_i$ for $i \notin [s, e]$ and $c_i^{(3)} = -(k' c'_{max} + 1)$. In other words, as long as $k'' \leq k'$ entries can be chosen outside of the interval $[s, e]$, no optimal solution should choose any entry within

the interval $[s, e]$.

Distributing the sparsity. The correctness of line 10 follows by Theorem 6.28.

Running time. Line 3 of algorithm Recover runs in time $O\big((\log\Delta)d'(\log(c'_{max}+1)+\log k')\big)$ which is a subset of $O\big((\log\Delta)d'(\log(c_{max}+1)+\log k)\big)$. Vector c' is split into c_L and c_R at line 11 in $O(d')$ time. Afterwards, the method recurses on the two subproblems, and combines their outputs into $u^\star$. Obtaining $u^\star$ at line 13 also takes $O(d')$. So, it only to remain to discuss the recursion.

From Theorem 6.28 we have that line 10 take $O\big(d'(\log c'_{max}+\log k'')\big)$. Furthermore, as $k'' \le k$ and $c'_{max} \le c_{max}$, we have $O\big(d'(\log c'_{max}+\log k')\big) = O\big(d'(\log c_{max}+\log k)\big)$.

Every recursive step shrinks the corresponding c' vector by half at least. Hence, the recursion has depth of $O(\log d)$. At every level of the recursion are considered vectors c' of total length $O(d)$ – this follows from the recursive call at line 12 and the fact the c_L and c_R represent disjoint pieces of c'. This implies that the total running time of the algorithm is

$$O\big(\log d(d\left(\log c_{max}+\log k\right)+(\log\Delta)d(\log(c_{max}+1)+\log k))\big),$$

which compactly can be written as $O\big(d(\log(c_{max}+1)+\log k)\log d\log\Delta\big)$. □

6.8 Two-Dimensional Δ-Separated Problem is NP-Hard

In this section we consider a natural extension of the Δ-separated problem in which vector c is two dimensional, i.e. $c \in \mathbb{R}^{d\times m}$. Formally, given c, sparsity parameter k, and integer Δ, the goal is to output k pairs of integers (i^t, j^t) such that:

- $1 \le i^t \le d$, and $1 \le j^t \le m$, for all $t = 1 \dots k$;
- for every $t \ne s$ we have $\min\{|i^t - i^s|, |j^t - j^s|\} \ge \Delta$; and
- $\sum_{t=1}^{k} c_{i^t,j^t}$ is maximized.

We refer by 2DΔ-separated(c,k,Δ) to that problem, and show it is NP-hard by reducing it to problem Box-Pack studied in [FPT81]. Let us start by recalling the definition of Box-Pack.

Let k be a number of identical 3×3 squares.[2] Our goal is to pack the k squares into a region of plane defined by set R. Set R consists of pairs of integers, where every pair represents the point at which a square can be placed, e.g. the upper-left corner of a square. Every square can only be placed so that its sides are parallel to the axis. By Box-Pack(k,R) we refer to the problem of answering whether for given k and R one can place all the k squares in the described way, such that no two squares overlap. In [FPT81] is proved the following result.

[2] We choose sides to be of length 3 as already that setting is sufficient to prove NP-hardness of Box-Pack, see the proof of Theorem 2 in [FPT81].

Theorem 6.30: Theorem 2 in [FPT81]

Box-Pack is NP-complete.

Now we utilize Theorem 6.30 to show that 2DΔ-separated(c,k,Δ) is NP-hard as well.

Theorem 6.31

Problem 2DΔ-separated(c,k,Δ) is NP-hard.

Proof. We provide a polynomial time reduction of Box-Pack(k,R) to a sequence of 2DΔ-separated instances.

Let R^E, denoting "extended R", be the set of all integers points that can be occupied by a square, not necessarily its corners. Observe that $\left|R^E\right| \leq 9|R|$. We say that two points in R^E are adjacent if they share the same x- or the same y-coordinate. Consider a connected component F of R^E.

Assume that Box-Pack(i,F) can be reduced to an 2DΔ-separated instance. Then, let us show that Box-Pack(k,R) can be reduced to polynomially many 2DΔ-separated instances.

Let $\mathscr{F}$ be the set of all connected components of R^E. Then, for every $C \in \mathscr{F}$ define val_F as follows

$$val_F \stackrel{\text{def}}{=} \arg \max_{i \in \{0,\ldots,\min\{|F|,k\}\}} \text{Box-Pack}(i,F) \text{ equals true.}$$

Now, clearly, if $\sum_{F\in\mathscr{F}} val_F \geq k$, then Box-Pack$(k,R)$ equals true. Observe that in order to compute all val_F, we need to compute only polynomially many instances of Box-Pack, where all the input parameters are bounded by k and R. So, it remains to reduce Box-Pack(i,F) to an instance of 2DΔ-separated.

To that end, consider $F \in \mathscr{F}$. Let h and w be the smallest values such that F is enclosed by an axis-parallel sides rectangle T of height h and width w. With loss of generality, assume that T and F are translated so that the bottom-left corner of T at $(1,1)$. Define vector $c \in \{0,1\}^{h\times w}$ to be an indicator vector of points of C at which can be placed square, i.e.

$$c^F_{i,j} = \begin{cases} 1 & \text{if } 3\times 3 \text{ square with upper-left corner at } (i,j) \text{ is in } F \\ 0 & \text{otherwise} \end{cases}$$

Note that $hw \leq |F|^2$. Next, define opt_F as follows

$$opt_F \stackrel{\text{def}}{=} \max_{i \in \{1,\ldots,\min\{|F|,k\}\}} \text{2D}\Delta\text{-separated}(c^F, i, 3).$$

Then, we claim opt_F equals val_F. Now, it is easy to see that the claim is true. Value val_F represents the maximal number of squares that can be packed within F. Each such square corresponds to an entry of c^F which has value 1. So, we have 2DΔ-separated$(c^F, val_F, 3) = val_F$, and hence $opt_F \geq val_F$. On the other hand, by the construction of c^F, opt_F represents a number of squares (not necessarily maximum, though) that can be packed in F. So, we have $val_F \geq opt_F$, and therefore $opt_F = val_F$ as claimed.

This concludes our proof. □

6.9 Uniform Size Neuronal Spike Trains

We propose a generalization of the Δ-separated model. In this model, we assume that neuronal spike trains correspond to blocks of uniform size. We use b to denote the size of blocks. Formally, we define model $\mathcal{M}_{k,\Delta,b}$ to be the set of all vector in $\mathbb{R}^d$ such that non-zero entries are grouped in blocks of size b, there are exactly k blocks, and every two blocks are separated by at least Δ.

In order to apply the existing framework for recovering measured vectors that belong to $\mathcal{M}_{k,\Delta,b}$, we develop a method that solves the following projection problem. Given vector $c \in \mathbb{R}^d$, the goal is to output k indices $p_1, \ldots, p_k$ such that

- $1 \leq p_i \leq d - b + 1$, for every $i = 1 \ldots k$;
- $\left|p_i - p_j\right| \geq \Delta + b - 1$, for all $i \neq j$; and,
- the sum
$$\sum_{i=1}^{k} \sum_{j=0}^{b-1} c_{p_i + j}$$
is maximized.

We refer to that problem by (Δ, b)-separated. Now, it is easy to show the following claim.

Theorem 6.32

Given vector $c \in \mathbb{N}_+^d$, sparsity k, and block size b, define vector $c^b \in \mathbb{N}_+^{d-b+1}$ as

$$c_i^b \stackrel{\text{def}}{=} \sum_{j=i}^{i+b-1} c_j.$$

Let S be a set of indices corresponding to an optimal solution for $(\Delta+b-1)$-separated problem for cost vector c^b and sparsity k. Then, S is an optimal solution to (Δ, b)-separated problem for cost vector c and sparsity k.

Proof. Clearly, S is feasible choice of indices for (Δ, b)-separated problem. Towards a contradiction, assume that there exists another set of indices S' that achieves larger value than S.

But then, S' is a feasible choice of indices for $(\Delta + b - 1)$-separated problem. Next, notice that S, and also S', achieve the same value for both (Δ, b)- and $(\Delta + b - 1)$-separated problem. However, as S' achieves higher value than S, this contradicts our assumption that S is an optimal choice of indices for $(\Delta + b - 1)$-separated problem. □

Now we can solve (Δ, b)-separated problem in nearly-linear time.

Corollary 6.33. *A solution to (Δ, b)-separated problem can be computed in nearly-linear time.*

Proof. We use the reduction from Theorem 6.32 to reduce (Δ, b)-separated problem to $(\Delta + b - 1)$-separated one. The reduction can be applied in linear time. Then, the claim follows by Theorem 6.29. □

6.9.1 Sample Complexity

Next we analyze the sample complexity of $\mathscr{M}_{k,\Delta,b}$ model. To that end, we count the number of support in that model. Now, all the supports are captured by $(\alpha_0, \alpha_1, \ldots, \alpha_k)$, where $\alpha_i \geq 0$ and $\sum_{i=0}^{k} \alpha_i = d - kb - (k-1)\Delta$. The idea is that k blocks split the vector in $k+1$ regions. All but the first and last region correspond to coordinates between two neighboring blocks. Value α_0 and α_k correspond to coordinates before the first block and after the last block, respectively. Value α_i, for $1 \leq i < k$ correspond to the slack distance between block i and $i+1$.

Let $\mathbb{M}_{k,\Delta,b}$ be the family of support patterns of $\mathscr{M}_{k,\Delta,b}$. Then, we have:

$$\left|\mathbb{M}_{k,\Delta,b}\right| = \binom{d - kb - (k-1)\Delta + k}{k}.$$

Now, from the result that there are RIP matrices with $O\left(k + \log\left|\mathbb{M}_{k,\Delta,b}\right|\right)$ rows, [BCDH10], the following claim follows.

Theorem 6.34

The sample complexity of $\mathscr{M}_{k,\Delta,b}$ is $O(k\log(d/k - (b+\Delta) + (\Delta-1)/k))$. In particular, if $(\Delta-1)/k \in O(1)$, then the sample complexity is $O(k\log(d/k - (b+\Delta))$.

7 Conclusion

In this thesis, we proposed a general strategy for designing efficient algorithms for large-scale tasks. Furthermore, we demonstrated that this strategy is effective on a number of fundamental problems. The techniques we developed drew upon tools from graph theory, combinatorics, discrete probability, and convex optimization. Randomization played a special role in the design of our methods that enable us to also develop simple algorithms.

Our approaches proved to be useful in tackling some of the central problems in computer science. Nevertheless, there are still numerous open questions in the context of large-scale computation. We believe that our work will open new directions to pursue for the future work in this area. In Section 7.1, we discuss some of these natural directions.

Here, we summarize the contributions presented in the previous chapters.

- In Chapter 2, we studied matchings in the context of massively parallel computation. Our work has introduced two techniques that, for any constant $\epsilon > 0$, enabled us to design an algorithm for constructing $(1+\epsilon)$-approximate maximum matchings. When the memory per machine is $\Theta(n)$, our algorithm provides almost exponential improvement in the round complexity, compared to the prior work.

- Our second result on graph algorithms in models of parallel computation concerns maximal independent sets. When the memory per machines is $\Theta(n)$, we showed in Chapter 3 that there is an algorithm that solves this problem in $O(\log\log n)$ rounds of computation. This constitutes an exponential improvement in the round complexity, compared to the algorithms known previously.

- In the context of network routing, in Chapter 4, we designed a stateless randomized routing scheme that in k-connected graphs delivers messages even if $k-1$ links have failed. Furthermore, each router uses only local information to perform routing decisions. Notably, our scheme employs randomness only when the routing encounters a failed link. This is the first such stateless scheme that can tolerate up to $k-1$ link failures.

- In Chapter 5, we studied the problem of maximizing submodular functions subject to cardinality constraint k in the streaming setting. We considered a regime in which up to m elements can be removed from the stream. In this context, we designed an algorithm that stores only $\tilde{O}(k+m)$ elements from the stream and provides a constant-factor approximation to this problem.

- In Chapter 6, we designed both randomized and deterministic nearly-linear time algorithms for projecting onto the set of separated sparse vectors. The core technique in our randomized algorithm is Lagrangian relaxation. There are separated sparsity instances for which the Lagrangian relaxation alone does not provide an optimal solution. One of our key insights here was that it is still possible to obtain an optimal solution if the original input instance is only slightly perturbed. We also proposed a generalization of the separated sparsity model.

7.1 Future Directions

In the rest of this chapter, we briefly discuss some directions for future research that we find interesting.

7.1.1 Parallel Computation of Approximately Maximum Matchings

Our work on computing approximate maximum matchings in the MPC model described in Chapter 2 was inspired by the question: can we design graph algorithms that require rounds of computation in the MPC fewer than in the PRAM model when the memory per machine is $\Theta(n)$? Our result provided an affirmative answer to this question in the context of one of the most well-studied problems in this domain: the (approximate) maximum matching problem. It would be very interesting to further explore the power of the MPC model and, in this context, study the round complexity of computing maximum matchings when the memory per machine is truly sublinear, i.e., $S \in \Theta(n^{1-\delta})$, for some constant $\delta \in (0,1)$. In this regime, our algorithm requires $O(\log n)$ round complexity, matching those of the prior work [Lub86, ABI86, IS86, II86, LMSV11, AG15].

While investing this direction, it is worth keeping in mind that our approach is highly efficient in reducing the maximum degree of a graph, as long as that degree is at least $(n/S)\operatorname{poly}\log n$. For instance, when $S = n$, in only $O(1)$ MPC-rounds our algorithm reduces the maximum degree from n to $\sqrt{n}$. Whereas, to reduce the maximum degree from $\log n$ to $O(1)$ the method requires $O(\log\log n)$ MPC-rounds. Hence, it might be constructive to first design a more efficient algorithm for the case when the maximum degree is $\Theta(\operatorname{poly}\log n)$.

Our approach does not compute a maximal matching, despite that for any constant $\epsilon > 0$ it provides a $(1+\epsilon)$-approximate maximum matching. It is still an open question whether it is possible to compute maximal matching in $o(\log n)$ MPC-rounds when the memory per machine is $\Theta(n)$. We believe that obtaining a round complexity similar to our result, e.g., $O(\operatorname{poly}\log\log n)$, for computing maximal matchings would lead to new techniques applicable in the context of parallel computation.

7.1.2 Network Routing under Link Failures

In the context of k-connected graphs, we designed a stateless routing scheme that with respect to k has the optimal robustness against link failures. To achieve this, the scheme performs routing along arborescences and, when a failure occurs, carefully switches between them.

This approach does not necessarily route a message along a shortest path from its source to its destination, even when there are no failed links in the network. It would be interesting to

extend our routing scheme to make it obtain short routing paths in the case when there are only a few failed links, while at the same time retaining high robustness against failures.

7.1.3 Fast Recovery for Separated Sparsity Signals

Our work illustrated how to apply Lagrangian relaxation to solve the separated sparsity projection. In our proofs, we rely on the fact that the separated sparsity projection has an LP formulation, i.e., it can be cast as an Integer LP with the constraint-matrix being totally unimodular (TUM).

It would be very exciting to extend this connection and describe a class of problems that exhibits similar property, i.e., for which Lagrangian relaxation of one or more equality constraints leads to the optimal solution. For instance, consider any problem that has a sparsity constraint and can be cast as LP with TUM constraint-matrix. Is it the case that, after perturbing the input instance slightly, the Lagrangian relaxation of the sparsity constraint solves the original problem?

Bibliography

[ABB+17] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. *CoRR*, abs/1711.03076, 2017.

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.

[AG15] Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13–15, 2015*, pages 202–211, 2015.

[AK17] Sepehr Assadi and Sanjeev Khanna. Randomized composable coresets for matching and vertex cover. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24–26, 2017*, pages 3–12, 2017.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

[ANOY14] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th ACM Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31–June 3, 2014*, pages 574–583, 2014.

[Ass17] Sepehr Assadi. Simple round compression for parallel vertex cover. *CoRR*, abs/1709.04599, September 2017.

[AST94] Alok Aggarwal, Baruch Schieber, and Takeshi Tokuyama. Finding a minimum-weightk-link path in graphs with the concave monge property and applications. *Discrete & Computational Geometry*, 12(3):263–280, 1994.

[AT18] Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. *arXiv preprint arXiv:1802.06440*, 2018.

[AZ06] A. Atlas and A. Zinin. U-turn Alternates for IP/LDP Fast-Reroute. *IETF Internet draft version 03*, February 2006.

[AZ08] A. Atlas and A. Zinin. Basic Specification for IP Fast Reroute: Loop-Free Alternates. *IETF, RFC 5286*, 2008.

[BBC14] Bubacarr Bah, Luca Baldassarre, and Volkan Cevher. Model-based sketching and recovery with expanders. In *SODA*, pages 1529–1543, 2014.

[BCDH10] Richard G. Baraniuk, Volkan Cevher, Marco F. Duarte, and Chinmay Hegde. Model-based compressive sensing. *IEEE Transactions on Information Theory*, 56(4):1982–2001, 2010.

[BCH17] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization, IPCO 2017, Waterloo, ON, Canada, June 26–28, 2017*, pages 86–98, 2017.

[BEPS12] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *Foundations of Computer Science (FOCS) 2012*, pages 321–330. IEEE, 2012.

[Ber09] Roberto Beraldi. Biased random walks in uniform wireless networks. *Mobile Computing, IEEE Transactions on*, 8(4):500–513, 2009.

[BFARR15] Florent Becker, Antonio Fernandez Anta, Ivan Rapaport, and Eric Reémila. Brief announcement: A hierarchy of congested clique models, from broadcast to unicast. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, PODC '15, pages 167–169. ACM, 2015.

[BFS12] Guy E Blelloch, Jeremy T Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 308–317. ACM, 2012.

[BHI15] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4–6, 2015*, pages 785–804, 2015.

[BHKP08] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Fast Edge Splitting and Edmonds' Arborescence Construction for Unweighted Graphs. In *Proc. SODA*, pages 455–464, 2008.

[BHN16] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18–21, 2016*, pages 398–411, 2016.

[BHN17] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$

worst case update time. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16–19*, pages 470–489, 2017.

[BHP12] Andrew Berns, James Hegeman, and Sriram V Pemmaraju. Super-fast distributed algorithms for metric facility location. In *International Colloquium on Automata, Languages, and Programming*, pages 428–439. Springer, 2012.

[BHW00] Costas Busch, Maurice Herlihy, and Roger Wattenhofer. Randomized greedy hot-potato routing. In *SODA*, pages 458–466, 2000.

[BJMO12] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Structured sparsity through convex optimization. *Statistical Science*, 27(4):450–468, 11 2012.

[BKS13] Paul Beame, Paraschos Koutris, and Dan 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.

[BKS14] Paul Beame, Paraschos Koutris, and Dan Suciu. Skew in parallel query processing. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22–27, 2014*, pages 212–223, 2014.

[BLP92] WW Bein, LL Larmore, and JK Park. The d-edge shortest-path problem for a Monge graph. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1992.

[BMKK14] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD*, pages 671–680. ACM, 2014.

[BMSC17] Ilija Bogunovic, Slobodan Mitrović, Jonathan Scarlett, and Volkan Cevher. Robust submodular maximization: A non-uniform partitioning approach. In *Int. Conf. Mach. Learn. (ICML)*, 2017.

[BS13] Michael Borokhovich and Stefan Schmid. How (Not) to Shoot in Your Foot with SDN Local Fast Failover - A Load-Connectivity Tradeoff. In *OPODIS*, pages 68–82, 2013.

[BS15] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, ICALP 2015, Kyoto, Japan, July 6–10, 2015, Proceedings, Part I*, pages 167–179, 2015.

[BS16] Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10–12, 2016*, pages 692–711, 2016.

[BW90] Graham Brightwell and Peter Winkler. Maximum hitting time for random walks on graphs. *Random Struct. Algorithms*, 1(3):263–276, October 1990.

[CGM+16] Marco Chiesa, Andrei Gurtov, Aleksander Mądry, Slobodan Mitrović, Ilya Nikolaevskiy, Michael Shapira, and Scott Shenker. On the resiliency of randomized routing against multiple edge failures. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 55. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[CHKK+15] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 143–152. ACM, 2015.

[CHPS17] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. In *31 International Symposium on Distributed Computing*, 2017.

[CŁM+18] Artur Czumaj, Jakub Łącki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. In *STOC*, 2018.

[CNM+16] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrović, Aurojit Panda, Andrei Gurtov, Aleksander Mądry, Michael Schapira, and Scott Shenker. The quest for resilient (static) forwarding tables. In *International Conference on Computer Communications (INFOCOM), 2016 IEEE*. IEEE, 2016.

[CNM+17] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrović, Andrei Gurtov, Aleksander Mądry, Michael Schapira, and Scott Shenker. On the resiliency of static forwarding tables. *IEEE/ACM Transactions on Networking (TON)*, 25(2):1133–1146, 2017.

[CNP+14] Marco Chiesa, Ilya Nikolaevskiy, Aurojit Panda, Andrei Gurtov, Michael Schapira, and Scott Shenker. Exploring the limits of static failover routing. *CoRR*, abs/1409.0034, 2014.

[CRT06] Emmanuel J. Candes, Justin Romberg, and Terence Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[DB13] Marco F. Duarte and Richard G. Baraniuk. Spectral compressive sensing. *Applied and Computational Harmonic Analysis*, 35(1):111 – 129, 2013.

[DDJB10] Eva L. Dyer, Marco F. Duarte, Don H. Johnson, and Richard G. Baraniuk. Recovering spikes from noisy neuronal calcium signals via structured sparse approximation. In *LVA/ICA*, pages 604–611, 2010.

[DE11] Marco F. Duarte and Yonina C. Eldar. Structured compressed sensing: From theory to applications. *IEEE Transactions on Signal Processing*, 59(9):4053–4085, 2011.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Commununication of the ACM*, 51(1):107–113, January 2008.

[DKO14] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 367–376. ACM, 2014.

[DLP12] Danny Dolev, Christoph Lenzen, and Shir Peled. "Tri, Tri again": Finding triangles and small subgraphs in a distributed setting. In *Distributed Computing*, pages 195–209. Springer, 2012.

[Don06] David L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[DSRB13] Eva L. Dyer, Christopher Studer, Jacob T. Robinson, and Richard G. Baraniuk. A robust and efficient method to recover neural events from noisy and corrupted data. In *6th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 593–596, 2013.

[EAG11] Khalid El-Arini and Carlos Guestrin. Beyond keyword search: discovering relevant scientific literature. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 439–447. ACM, 2011.

[Edm65] Jack Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, pages 449–467, 1965.

[Edm72] Jack Edmonds. Edge-disjoint branchings. *Combinatorial Algorithms*, pages 91–96, 1972.

[EGR14] Theodore Elhourani, Abishek Gopalan, and Srinivasan Ramasubramanian. IP Fast Rerouting for Multi-Link Failures. In *Proc. IEEE INFOCOM*, pages 2148–2156, 2014.

[EM09] Yonina Eldar and Moshe Mishali. Robust recovery of signals from a structured union of subspaces. *IEEE Transactions on Information Theory*, 55(11):5302–5316, 2009.

[ERC07] Gábor Enyedi, Gábor Rétvári, and Tibor Cinkler. A Novel Loop-free IP Fast Reroute Algorithm. In *Proc. EUNICE*, pages 111–119. Springer-Verlag, 2007.

[FG17] Manuela Fischer and Mohsen Ghaffari. Deterministic distributed matching: Simpler, faster, better. *CoRR*, abs/1703.00900, 2017.

[FGP+12] Joan Feigenbaum, P. Brighten Godfrey, Aurojit Panda, Michael Schapira, Scott Shenker, and Ankit Singla. On the resilience of routing tables. In *Brief announcement PODC*, July 2012.

[FM85] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[FMN15] Simon Foucart, Michael F. Minner, and Tom Needham. Sparse disjointed recovery from noninflating measurements. *Applied and Computational Harmonic Analysis*, 39(3):558 – 567, 2015.

[FMS+10] Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Clifford Stein, and Zoya Svitkina. On distributing symmetric streaming computations. *ACM Transactions on Algorithms*, 6(4):66:1–66:19, 2010.

[FN18] Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy mis. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2152–2160. SIAM, 2018.

[FPT81] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information processing letters*, 12(3):133–137, 1981.

[FR13] Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Birkhäuser Basel, 2013.

[GB81] Eli M. Gafni and Dimitri P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 1981.

[GGK+18] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. *ACM Symposium on Principles of Distributed Computing*, 2018.

[Gha16] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proc. SODA*, 2016.

[Gha17] Mohsen Ghaffari. Distributed mis via all-to-all communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 141–149. ACM, 2017.

[GHJ+09] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, pages 51–62. ACM, 2009.

[GI10] Anna Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.

[GJN11] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361, August 2011.

[GKB03] Mesut Günes, Martin Kähmer, and Imed Bouazizi. Ant-routing-algorithm (ara) for mobile multi-hop ad-hoc networks-new features and results. In *The second mediterranean workshop on ad-hoc networks*, 2003.

[Goo16] Google CEO. Q3 2016 Earnings Call. https://abc.xyz/investor/pdf/2016_Q3_Earnings_Transcript.pdf, 2016. Accessed: 2018-01-12.

[GP16] Mohsen Ghaffari and Merav Parter. Mst in log-star rounds of congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, 2016.

[GSZ11] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the MapReduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer, 2011.

[HB11] Chinmay Hegde and Richard G. Baraniuk. Sampling and recovery of pulse streams. *IEEE Transactions on Signal Processing*, 59(4):1505–1517, 2011.

[HDC09] Chinmay Hegde, Marco F. Duarte, and Volkan Cevher. Compressive sensing recovery of spike trains using a structured sparsity model. In *SPARS'09-Signal Processing with Adaptive Sparse Structured Representations*, 2009.

[HIS14] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. Nearly linear-time model-based compressive sensing. In *International Colloquium on Automata, Languages, and Programming*, pages 588–599. Springer, 2014.

[HIS15a] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. Approximation algorithms for model-based compressive sensing. *IEEE Transactions on Information Theory*, 61(9), 2015.

[HIS15b] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. Fast algorithms for structured sparsity. *Bulletin of the European Association for Theoretical Computer Science*, 117:197–228, 2015.

[HIS15c] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. A nearly-linear time framework for graph-structured sparsity. In *ICML*, pages 928–937. JMLR Workshop and Conference Proceedings, 2015.

[HK15] Qingqing Huang and Sham M. Kakade. Super-resolution off the grid. In *Conference on Neural Information Processing Systems (NIPS)*, pages 2665–2673, 2015.

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 489–498. ACM, 2016.

[HKP99] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. A faster distributed algorithm for computing maximal matchings deterministically. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '99, pages 219–228, New York, NY, USA, 1999. ACM.

[HKP01] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001.

[HP14] James W Hegeman and Sriram V Pemmaraju. Lessons from the congested clique applied to MapReduce. In *the Proceedings of the International Colloquium on Structural Information and Communication Complexity*, pages 149–164. Springer, 2014.

[HPP+15] James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 91–100. ACM, 2015.

[HPS14] James W Hegeman, Sriram V Pemmaraju, and Vivek B Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 514–530. Springer, 2014.

[HTW15] Trevor Hastie, Robert Tibshirani, and Martin J. Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman and Hall/CRC, 2015.

[HZM11] Junzhou Huang, Tong Zhang, and Dimitris Metaxas. Learning with structured sparsity. *The Journal of Machine Learning Research*, 12:3371–3412, 2011.

[IBY+07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Operating Systems Review*, 41(3):59–72, March 2007.

[II86] Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986.

[IS86] Amos Israeli and Yossi Shiloach. An improved parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):57–60, 1986.

[JN18] Tomasz Jurdziński and Krzysztof Nowicki. Mst in O (1) rounds of congested clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2620–2632. SIAM, 2018.

[KG10] Andreas Krause and Ryan G Gomes. Budgeted nonparametric learning from data streams. In *ICML*, pages 391–398, 2010.

[KGGZ11] Kin-Wah Kwong, Lixin Gao, Roch Guérin, and Zhi-Li Zhang. On the Feasibility and Efficacy of Protection Routing in IP Networks. *IEEE/ACM Trans. Networking*, 19(5):1543–1556, October 2011.

[KKS14] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5–7, 2014*, pages 734–751, 2014.

[KKT03] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.

[KMGG08] Andreas Krause, H Brendan McMahan, Carlos Guestrin, and Anupam Gupta. Robust submodular observation selection. *Journal of Machine Learning Research*, 9(Dec):2761–2801, 2008.

[KMVV15] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in MapReduce and streaming. *ACM Transactions on Parallel Computing*, 2(3):14, 2015.

[KMW06] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA 2006, pages 980–989, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[Kor16] Janne H Korhonen. Deterministic mst sparsification in the congested clique. *arXiv preprint arXiv:1605.02022*, 2016.

[KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17–19, 2010*, pages 938–948, 2010.

[Kur05] James F Kurose. *Computer networking: A top-down approach featuring the internet, 3/E*. Pearson Education India, 2005.

[LB11] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Assoc. for Comp. Ling.: Human Language Technologies-Volume 1*, 2011.

[LCR+07] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. In *SIGCOMM*, 2007.

[LDP07] Michael Lustig, David Donoho, and John M. Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine*, 58(6), 2007.

[Len13] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 42–50, 2013.

[Lin87] Nathan Linial. Distributive graph algorithms global solutions from local data. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 331–335. IEEE, 1987.

[LMSV11] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2011, San Jose, CA, USA, June 4–6, 2011*, pages 85–94, 2011.

[LPP15] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. *Journal of the ACM*, 62(5):38:1–38:17, November 2015.

[LPPSP03] Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in O(log log n) communication rounds. In *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pages 94–100. ACM, 2003.

[LPS+13] Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. Ensuring Connectivity via Data Plane Mechanisms. In *Proc. of NSDI*, pages 113–126, 2013.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[LWD16] Erik Lindgren, Shanshan Wu, and Alexandros G Dimakis. Leveraging sparsity for efficient submodular data summarization. In *Advances in Neural Information Processing Systems*, pages 3414–3422, 2016.

[LYSS11] Junda Liu, Baohua Yan, Scott Shenker, and Michael Schapira. Data-driven Network Connectivity. In *Proc. of HotNets*, pages 8:1–8:6, New York, NY, USA, 2011. ACM.

[MBNF+17] Slobodan Mitrović, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Volkan Cevher. Streaming robust submodular maximization: A partitioned thresholding approach. *NIPS*, 2017.

[McG05] Andrew 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.

[MJOB11] Julien Mairal, Rodolphe Jenatton, Guillaume Obozinski, and Francis Bach. Convex and network flow optimization for structured sparsity. *The Journal of Machine Learning Research*, 12:2681–2720, 2011.

[MKK17] Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In *International Conference on Machine Learning*, pages 2449–2458, 2017.

[MMS18] Aleksander Mądry, Slobodan Mitrović, and Ludwig Schmidt. A Fast Algorithm for Separated Sparsity via Perturbed Lagrangians. In *Artificial Intelligence and Statistics*, 2018.

[MP80] J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.

[MR17] Deep Medhi and Karthik Ramasamy. *Network routing: algorithms, protocols, and architectures*. Morgan Kaufmann, 2017.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. ACM Symposium on Theory of Computing*, 2014.

[Nee15] Tom Needham. Dictionary-sparse and disjointed recovery. In *International Conference on Sampling Theory and Applications (SampTA)*, pages 278–282, 2015.

[NFBEH+16] Ashkan Norouzi-Fard, Abbas Bazzi, Marwa El Halabi, Ilija Bogunovic, Ya-Ping Hsieh, and Volkan Cevher. An efficient streaming algorithm for the submodular cover problem. In *Adv. Neur. Inf. Proc. Sys. (NIPS)*, 2016.

[NLY+07] Srihari Nelakuditi, Sanghwan Lee, Yinzhe Yu, Zhi-Li Zhang, and Chen-Nee Chuah. Fast Local Rerouting for Handling Transient Link Failures. *IEEE/ACM Trans. Networking*, 15(2):359–372, April 2007.

[NRWY12] Sahand N. Negahban, Pradeep Ravikumar, Martin J. Wainwright, and Bin Yu. A unified framework for high-dimensional analysis of M-estimators with decomposable regularizers. *Statistical Science*, 27(4):538–557, 11 2012.

[NT09] Deanna Needell and Joel A. Tropp. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321, 2009.

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.

[NWF78] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[OR10] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5–8 June 2010*, pages 457–464, 2010.

[OSU16] James B Orlin, Andreas S Schulz, and Rajan Udwani. Robust monotone submodular function maximization. In *Int. Conf. on Integer Programming and Combinatorial Opt. (IPCO)*. Springer, 2016.

[PR07] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

[PSA05] Ping Pan, George Swallow, and Alia Atlas. Rfc 4090 fast reroute extensions to rsvp-te for lsp tunnels. *Internet Request for Comments*, page 42, 2005.

[PST11] Boaz Patt-Shamir and Marat Teplitsky. The round complexity of distributed sorting. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 249–256, 2011.

[RRN12] Nikhil S. Rao, Ben Recht, and Robert D. Nowak. Universal measurement bounds for structured sparse signal recovery. In *AISTATS*, volume 22 of *JMLR Proceedings*, pages 942–950, 2012.

[RVW16] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits: On lower bounds for modern parallel computation. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11–13, 2016*, pages 1–12, 2016.

[Sch02] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2002.

[SCK+03] Gero Schollmeier, Joachim Charzinski, Andreas Kirstadter, Christoph Reichert, Karl J. Schrodi, Yuri Glickman, and Chris Winkler. Improving the Resilience in IP Networks. In *Proc. HPSR*, 2003.

[SCR13] Brent Stephens, Alan L. Cox, and Scott Rixner. Plinko: Building Provably Resilient Forwarding Tables. In *Proc. of HotNets*, pages 26:1–26:7. ACM, 2013.

[Scr18] ScrapeHero. How Many Products Does Amazon Sell? https://www.scrapehero.com/many-products-amazon-carry-january-2018/, 2018. Accessed: 2018-01-16.

[Sta16] Statista. Number of active Amazon customer accounts worldwide. https://www.statista.com/statistics/476196/number-of-active-amazon-customer-accounts-quarter/, 2016. Accessed: 2018-01-16.

[SW06] FB Shepherd and PJ Winzer. Selective randomized load balancing and mesh networks with changing demands. *Journal of Optical Networking*, 5(5):320–339, 2006.

[TIWB14] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in neural information processing systems*, pages 1413–1421, 2014.

[Val82] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11(2):350–361, 1982.

[Wai14] Martin J. Wainwright. Structured regularizers for high-dimensional problems: Statistical and computational issues. *Annual Review of Statistics and Its Application*, 1(1):233–253, 2014.

[Whi12] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[WN07] Junling Wang and Srihari Nelakuditi. IP Fast Reroute with Failure Inferencing. In *Proc. of SIGCOMM Workshop on Internet Network Management*, INM, pages 268–273, New York, NY, USA, 2007. ACM.

[YL06] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2006.

[YLS+14] Baohua Yang, Junda Liu, Scott Shenker, Jun Li, and Kai Zheng. Keep Forwarding: Towards K-link Failure Resilient Routing. In *Proc. IEEE INFOCOM*, pages 1617–1625, 2014.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22*, 2010.

[ZNY+05] Zifei Zhong, Srihari Nelakuditi, Yinzhe Yu, Sanghwan Lee, Junling Wang, and Chen nee Chuah. Failure Inferencing based Fast Rerouting for Handling Transient Link and Node Failures. In *Proc. IEEE INFOCOM*, 2005.

[ZSM08] Rui Zhang-Shen and Nick McKeown. Designing a fault-tolerant network using valiant load-balancing. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*. IEEE, 2008.

[ZWB13] Baobao Zhang, Jianping Wu, and Jun Bi. RPFP: IP fast reroute with providing complete protection and without using tunnels. In *IWQoS*, pages 137–146, 2013.

Slobodan Mitrović

Rte de la Maladiere 4
1022 Chavannes-près-Renens
Switzerland

slobodan.mitrovic@epfl.ch
+41 78 78 37 001

EDUCATION

PhD in Theoretical Computer Science — February 2013 - present
École Polytechnique Fédérale de Lausanne, Switzerland

- Advisor: Prof. Rüdiger Urbanke; co-advisor: Prof. Aleksander Mądry
- Research interests: algorithmic approach to optimization; fast graph algorithms (particular distributed and streaming); randomized algorithm

MSc in Computer Science — September 2010 - February 2013
École Polytechnique Fédérale de Lausanne, Switzerland

- Thesis advisor: Prof. János Pach
- Thesis title: *Metric problems on graphs*
- Thesis defense date: February, 2013

BSc in Computer Science — October 2006 - July 2010
Faculty of Natural Sciences and Mathematics, Serbia

PUBLICATIONS

- **Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover**
  Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, Ronitt Rubinfeld
  ACM Symposium on Principles of Distributed Computing, **PODC 2018**
  https://arxiv.org/abs/1802.08237
- **Beyond** 1/2**-Approximation for Submodular Maximization on Massive Data Streams**
  Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrović, Amir Zandieh, Aidasadat Mousavifar, Ola Svensson
  The 35th International Conference on Machine Learning, **ICML 2018**
- **Round compression for parallel matching algorithms**
  Artur Czumaj, Jakub Łacki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, Piotr Sankowski
  The 50th ACM Symposium on Theory of Computing, **STOC 2018**
  Invited to a special issue of SIAM Journal on Computing
- **A Fast Algorithm for Separated Sparsity via Perturbed Lagrangians**
  Aleksander Mądry, Slobodan Mitrović, Ludwig Schmidt
  The 21st International Conference on Artificial Intelligence and Statistics, **AISTATS 2018**
- **Streaming Robust Submodular Maximization: A Partitioned Thresholding Approach**
  Slobodan Mitrović, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub Tarnawski, Volkan Cevher
  The 31th Conference on Neural Information Processing Systems, **NIPS 2017**

- **Robust submodular maximization: A non-uniform partitioning approach**
  Ilija Bogunovic, Slobodan Mitrović, Jonathan Scarlett, Volkan Cevher
  The 34th International Conference on Machine Learning, **ICML 2017**
- **On the Resiliency of Randomized Routing Against Multiple Edge Failures**
  Marco Chiesa, Andrei Gurtov, Aleksader Mądry, Slobodan Mitrović, Ilya Nikolaevskiy, Michael Schapira, Scott Shenker
  The 43rd International Colloquium on Automata, Languages, and Programming, **ICALP 2016**
- **On the Resiliency of Static Forwarding Tables**
  Marco Chiesa, Andrei Gurtov, Aleksander Mądry, Slobodan Mitrović, Ilya Nikolaevskiy, Michael Schapira, Scott Shenker
  IEEE/ACM Transactions on Networking (Volume: 25, Issue: 2), **ToN 2016**
- **The Quest for Resilient (Static) Forwarding Tables**
  Marco Chiesa, Andrei Gurtov, Aleksander Mądry, Slobodan Mitrović, Ilya Nikolaevskiy, Aurojit Panda, Michael Schapira, Scott Shenker
  The 35th IEEE International Conference on Computer Communications, **INFOCOM 2016**
- **Homometric sets in diameter-two and outerplanar graphs**
  Slobodan Mitrović
  The 17th IEEE Mediterranean Electrotechnical Conference, **MELECON 2014**
- **Homometric sets in trees**
  Radoslav Fulek, Slobodan Mitrović
  European Journal of Combinatorics 2014, 35:256-263

MANUSCRIPTS

- **Identifying Maximal Non-Redundant Integer Cone Generators**
  Slobodan Mitrović, Ruzica Piskac, Viktor Kuncak
  http://lara.epfl.ch/w/_media/projects:nicg.pdf

PRIZES, AWARDS, AND FELLOWSHIPS

- Doc.Mobility Fellowship granted by Swiss National Science Foundation, July 1, 2015 – May 31, 2016
- Excellence scholarship for Master studies at EPFL (2010/2011, 2011/2012)
- EFG Eurobank scholarship for the best students in Serbia (2009/2010)
- Serbian Ministry of Youth and Sport Scholarship for the best students in the final year of studies (2009/2010)
- 1st place at IEEE R8 student paper contest 2014
- Aleksandar Saša Popović Award for the best student of generation in informatics, Faculty of Natural Science, University of Novi Sad
- University award for excellent results in 2006/2007, 2007/2008, 2008/2009 and 2009/2010 year of study

EMPLOYMENT HISTORY

Software Development Engineer Intern July 2012 - September 2012
SMS Team, Facebook, Menlo Park, California

Software Development Engineer Intern March 2012 - June 2012
Semantic Search Team, Google, Zurich

- Improvements of web-references posterior probabilities

Software Development Engineer Intern July 2011 - September 2011
Google, Zurich
- Geocoding quality improvements to Google Maps

Half-time Software Development Engineer Intern July 2008 - September 2010
Wowd Inc., California (Department in Belgrade, Serbia)
- Research and development of scalable distributed web search engine

TEACHING ACTIVITIES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
- **Teaching assistant** for *CS-250 Algorithms* Fall 2016
- **Teaching assistant** for *CS-412 Discrete computational geometry* Spring 2015
- **Teaching assistant** for *CS-352 Theoretical Computer Science* Fall 2014
- **Teaching assistant** for *CS-251 Theory of Computation* Spring 2014
- **Teaching assistant** for *CS-352 Theoretical Computer Science* Fall 2013
- **Teaching assistant** for *CS-450 Advanced Algorithms* Fall 2012
- **Lecturer** for EPFL ACM ICPC teams September 2013 - present

SERBIA
- **Lecturer** in the training camps for Serbian teams in International Olympiad in Informatics September 2007 - present

OTHER ACCOMPLISHMENTS
- Finals of Challenge24 – BME International Programming Contest, 2010, 2011 and 2012
- Won 2nd in 2008 and 3rd place in 2010 at BubbleCup, a team contest on algorithmic problems solving organized by Microsoft
- Represented EPFL at ACM ICPC, Madrid, 2010 (bronze medal)
- Represented University of Novi Sad at International Student Informatics Competition ACM ICPC, Bucharest, Romania, 2006, 2008 and 2009
- Deputy-leader of Serbian team at BalkanOI 2008 in Macedonia, Ohrid
- Team-leader of Serbian team at CEOI 2009, Romania; BOI and JBOI 2009, Bulgaria, Shumen
- Author of the algorithmic tasks for TopCoder and High school competitions in Serbia since 2008