Algorithms for Environments with Uncertainty

Gregory Schwartzman
Algorithms for Environments with Uncertainty

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Gregory Schwartzman

Submitted to the Senate of the Technion — Israel Institute of Technology
SIVAN 5777 Haifa June 2017
This research was carried out under the supervision of Prof. Keren Censor-Hillel, in the Faculty of Computer Science.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author’s doctoral research period, the most up-to-date versions of which being:


**Acknowledgements**

I would like to thank Professor Keren Censor-Hillel for her support and guidance. I would also like to thank my parents for the moral and culinary support.

The generous financial help of the Technion is gratefully acknowledged.
## Contents

### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Abstract

Abstract

### Abbreviations and Notations

Abbreviations and Notations

### 1 Introduction

1.1 Models of computation

1.1.1 Distributed Graph Algorithms

1.1.2 Semi-Streaming Graph Algorithms

1.2 Results of this thesis

1.2.1 Approximating weighted maximum matching in the semi-streaming model

1.2.2 Approximating weighted vertex cover in the distributed setting

1.2.3 Approximating weighted maximum matching and weighted maximum independent set in the distributed setting

1.2.4 Distributed property testing

1.2.5 Derandomizing local distributed algorithms

### 2 Related work

2.1 Maximum matching in the semi-streaming model

2.2 Distributed minimum vertex cover approximation

2.3 Distributed maximum independent set and maximum matching approximation

2.4 Property testing

2.4.1 Historical overview

2.4.2 Distributed property testing

2.5 Distributed derandomization

2.5.1 Distributed computation of MIS.

2.5.2 Distributed constructions of spanners.

2.5.3 Algorithms in the congested clique.

2.5.4 Derandomization in the parallel setting.

### 3 Preliminaries

3.1 The Local-ratio technique

3.1.1 A Simple Local-Ratio Approximation Algorithm for MWM

3.2 Additional background on property testing
3.3 Derandomization tools ......................................................... 28
  3.3.1 Lenzen’s routing algorithm ............................................. 29
3.4 Mathematical background .................................................. 30

4 Approximating weighted maximum matching in the semi-streaming model 31
  4.1 Approximating Maximum Weight Matching ............................... 31
    4.1.1 Extending the Local-Ratio Technique ................................. 31
  4.2 A Semi-Streaming Algorithm .............................................. 33
    4.2.1 Algorithm $\text{MWM-seq}$ ............................................ 34
    4.2.2 Implementing Algorithm $\text{MWM-seq}$ in the Semi-Streaming Model 39

5 Approximating weighted vertex cover in the distributed setting 43
  5.1 A local ratio template for approximating MWVC .......................... 43
  5.2 A fast distributed implementation ......................................... 44
  5.3 Adaptation to the CONGEST model ........................................ 48
  5.4 Discussion of [KMW10] ....................................................... 49
  5.5 A tighter bound on the number of communication rounds .................. 50

6 Approximating weighted maximum matching and weighted maximum independent set in the distributed setting 51
  6.1 MaxIS approximation .......................................................... 51
    6.1.1 Sequential MaxIS approximation via local ratio ...................... 52
    6.1.2 Distributed MaxIS approximation via local ratio .................... 54
    6.1.3 Deterministic coloring-based approximation algorithm ............ 56
  6.2 Distributed 2-approximation for maximum weighted matching .............. 58
  6.3 Faster Approximations of Maximum Matching ............................... 60
    6.3.1 A fast $(2 + \varepsilon)$-approximation of maximum weighted matching . 60
    6.3.2 A fast $(1 + \varepsilon)$-approximation of maximum cardinality matching in LOCAL 63
    6.3.3 A fast $(1 + \varepsilon)$-approximation of maximum cardinality matching in CONGEST 67
  6.4 Alternative Fast Method for $2 + \varepsilon$ Approximation of Unweighted Maximum Matching 75
    6.4.1 The Algorithm for Bipartite Unweighted Graphs ....................... 75
    6.4.2 The Algorithm for General Unweighted Graphs ........................ 76

7 Distributed property testing .................................................. 77
  7.1 Distributed emulation of sequential tests in the dense model ............ 77
    7.1.1 Applications: $k$-colorability and perfect graphs .................. 80
  7.2 Distributed test for triangle-freeness .................................... 81
  7.3 Distributed bipartiteness test for bounded degree graphs .................. 84
  7.4 Distributed test for cycle-freeness ....................................... 88
  7.5 Lower bounds for testing bipartiteness and cycle-freeness ................. 92

8 Derandomizing local distributed algorithms .................................. 95
  8.1 Deterministic MIS ............................................................ 96
    8.1.1 An $O(\log \Delta)$ round randomized MIS algorithm in the congested clique 96
8.1.2 Derandomizing the modified MIS algorithm ................................. 97
8.1.3 An $O(\log \Delta)$ deterministic MIS algorithm for $\Delta = O(n^{1/3})$ ........ 105
8.1.4 An $O(D \log^2 n)$ deterministic MIS algorithm for CONGEST .............. 106
8.2 Deterministic spanner construction .................................................. 106

9 Conclusions ................................................................. 115

Hebrew Abstract .......................................................... i
List of Figures

6.1 Augmenting Paths in a Bipartite Graph ........................................ 68
Abstract

In this research we study computation in environments with uncertainty, specifically, the distributed and streaming environments. We adapt the local-ratio technique to the distributed and streaming environments. In doing so we achieve state of the art approximation algorithms for weighted vertex cover, weighted maximum matching and weighted maximum independent set in the distributed setting. Many of our results in the distributed setting achieve optimal running time. In the semi-streaming model we present a deterministic $(2 + \epsilon)$-approximation algorithm for maximum weight matching. This improves upon the previous best known approximation ratio of $(3.5 + \epsilon)$.

We also show how to apply property testing and derandomization techniques in the distributed settings. We initiate a thorough study of distributed property testing – producing algorithms for property testing problems in the CONGEST model. In particular, we present a distributed emulation technique for many sequential tests, and present distributed tests for triangle-freeness, cycle-freeness, and bipartiteness. In addition, we show a logarithmic lower bound for testing bipartiteness and cycle-freeness. We also present tools for derandomizing solutions to local problems in the Congested Clique and CONGEST model. Our techniques give a deterministic maximal independent set (MIS) algorithm in the CONGEST model running in $O(D \log^2 n)$ rounds, where $D$ is the diameter of the graph.
Abbreviations and Notations

$G(V,E)$ : a graph on the vertex set $V$ and edge set $E$

$n$ : the number of nodes in a graph

$m$ : the number of edges in a graph

$\Delta$ : the maximum degree in a graph

$d(v)$ : the degree of the vertex $v$

$D$ : the diameter of a graph

$k$-spanner : a spanner with stretch $k$

MIS : maximal independent set

MaxIS : maximum independent set

LOCAL : a distributed computational model with unbounded message size

CONGEST : a distributed computational model with bounded message size (usually $O(\log n)$)
Chapter 1

Introduction

In this research we study computation in environments with uncertainty, specifically, the distributed and streaming environments. We achieve state of the art results for central problems in the distributed and streaming environments for matching and covering problems. We also introduce techniques such as derandomization and property testing to the distributed setting.

We address a cornerstone family of problems in distributed computing, the so-called \textit{local problems}. Intuitively, as opposed to \textit{global problems}, local problems admit solutions that do not require communication over the entire network graph. Specifically, we are interested in approximating packing and covering problems. While finding an exact solution is a global problem, finding an approximate solution is a local problem, which admits to very fast distributed algorithms.

One of the main tools used in this research is the Local-Ratio technique, pioneered by Bar-Yehuda and Even [BYE85]. We apply this technique in the distributed and streaming environments and achieve significant improvements for approximating various covering and packing problems. The main advantage of this technique over other techniques in the distributed setting is its very local nature (despite having many non-local applications). This property turns out to be helpful in the semi-streaming environment, implying that locality plays a role in the semi-streaming environment.

One fundamental characteristic of distributed algorithms for local problems is whether they are deterministic or randomized. Currently, there exists a curious gap between the known complexities of randomized and deterministic solutions for local problems. Interestingly, the main indistinguishability-based technique used for obtaining the relatively few lower bounds that are known seems unsuitable for separating these cases. We explore this gap in the distributed setting by introducing derandomization techniques from the sequential setting to the distributed environment.

1.1 Models of computation

1.1.1 Distributed Graph Algorithms

In the classical case of distributed graph algorithms, we wish to perform some task given a communication graph. Each node is allowed to communicate locally with its neighbors and is
assumed to have unlimited computational power. We wish to solve the task while minimizing the communication between nodes. Here also there are a few models that define how to measure the amount of communication used. The two main synchronous models are LOCAL and CONGEST. CONGEST tries to minimize communication rounds while limiting the message size (\(O(\log n)\) bits for example), while the LOCAL model does not limit the message size. Of course we could also analyze the total communication bits sent and not only the communication rounds. Another model which we consider is the Congested Clique model. This model is almost identical to the CONGEST model, and differs in the fact that it allows all-to-all communication.

For global problems, such as computing a minimum spanning tree (MST) and more, separations between LOCAL and CONGEST are known [Elk04, PR99, DSHK+11]. Intriguingly, such separations are in general not known for local problems and are a central open question. Hence, it is crucial to study the complexity of local problems under bandwidth restrictions. Surprisingly, some (but not all) of the algorithms for local problems already use only small messages. This includes, for example, the classic MIS algorithms of Luby [Lub86] and Cole and Vishkin [CV86], as well as the deterministic coloring algorithm of Barenboim [Bar15].

1.1.2 Semi-Streaming Graph Algorithms

In the semi-streaming model of computation, the input graph is given as a stream of edges. At each iteration, the algorithm receives an edge from the stream and processes it. As in sequential algorithms, we wish to compute some function of the input graph. But, we assume the number of edges in the graph is too large to fit in memory, so we limit the algorithm to use only \(O(n \text{ polylog } n)\) space. Moreover, if processing an incoming edge takes long, we have to keep a queue of later incoming edges, which may result in exceeding the space limitations; thus, short processing time for an incoming edge is of high importance, while pre-processing and post-processing times are of less importance.

This model is motivated by the recent rise of big-data. We aim to develop algorithms for huge graphs where any amount of data substantially larger than \(O(n \text{ polylog } n)\) cannot fit within memory. Thus, all classical graph algorithms would benefit from a semi-streaming version. Approximation algorithms play a vital role here, as problems for which there exists an exact algorithm in the classical setting may become hard in the semi-streaming setting. When considering approximation algorithms, we usually aim to achieve the best approximation factor possible within the constraints of the model.

There are various variations to this model which differ by two main parameters: space and number of passes. The streaming model demands memory consumption to not exceed \(O(n)\) (no polylogarithmic factors), hence the name ”semi-streaming” when we do allow polylogarithmic factors. Some algorithms try to operate within \(O(n^{1+\epsilon})\) space, while others even allow \(O(n^c)\), \(c \in (1, 2)\) space. The second parameter is the number of passes. This is the number of times the algorithms receives the input stream. When allowing multiple passes we usually limit it to some small constant. This parameter plays a key role in the performance of streaming algorithms, and often there is a trade off between the space required by an algorithm and the number of passes we allow.
1.2 Results of this thesis

1.2.1 Approximating weighted maximum matching in the semi-streaming model

In [PS17] we present a simple \((2 + \epsilon)\)-approximation algorithm for the maximum weight matching (MWM) problem in the semi-streaming model. Our algorithm is deterministic, single-pass, requires only \(O(1)\) processing time per incoming edge, and uses \(O(n \log^2 n)\) space for any constant \(\epsilon > 0\). This improves upon the previously best known approximation algorithm of Crouch and Stubbs [CS14, GMZ16], which achieves an approximation ratio of \((3.5 + \epsilon)\) and takes \(O(\log n)\) time to process an edge.

The MWM problem is a classical problem in graph theory. Its first efficient solution is due to Edmonds [Edm65b], which was later improved by Micali and Vazirani [MV80]. The MWM problem was one of the first to be considered in the semi-streaming model when it was first presented [FKM+05], and probably the most studied problem in this model since.

In the first algorithms for the MWM problem, a matching is maintained at all times, and is being updated according to the incoming edges. More recent algorithms sort the edges into weight classes, keep a subset of each class, and then find a matching in the union of these subsets.

Like previous algorithms, our algorithm maintains a set of edges from which the final matching is constructed; however, we do not maintain a matching at all times, but only construct it in a post-processing stage. Our main technical contribution is the adaptation of the local-ratio technique for maximization problems [BYE85, BBF+01] to the semi-streaming model, in a novel, yet simple, manner. Our work presents a significantly better approximation ratio for the MWM problem, along with a new approximation technique for optimization problems in the semi-streaming model.

For the maximum unweighted matching problem, a simple greedy algorithm yields a 2-approximation. This was observed in the very first paper on the semi-streaming model [FKM+05], and not improved since. Any future improvement of our result by more than an \(\epsilon\)-factor will also solve this long-standing problem.

Our Contribution

When developing an algorithmic framework for a new model, it is natural to first address the most fundamental algorithmic problems. Finding a large matching in a graph is indeed a fundamental problem, which has been extensively studied in the model of semi-streaming graph algorithms. Our algorithm uses an extension of a well studied approximation framework, namely the local-ratio technique, while previous algorithm used clever ideas which were specifically crafted for the problem and model.

A simple local-ratio algorithm for the MWM problem in the sequential model of computation goes roughly as follows: repeatedly select an edge with positive weight; reduce its weight from the edge itself and from all its neighboring edges; push the edge into a stack and continue to the next edge, as long as there is an edge with positive weight; finally, unwind the stack and add the edges greedily to the matching. This procedure results in a 2-approximation for the MWM problem. It can be extended to a \((2\alpha)\)-approximation, for \(\alpha > 1\), if at each step we reduce the weight of the processed edge multiplied by \(\alpha\) from its neighbors.
The challenge in translating this technique to the semi-streaming model is twofold. First, we have to reduce edge weights from edges that are yet to arrive. This is solved by saving, for each node, the total amount that should be reduced from each arriving edge containing this node, and reducing weight retroactively from incoming stream edges. The idea of reducing weight retroactively appeared in [BBF+01].

The second, more substantial challenge, is limiting the size of the stack, so it can comply with the \( O(n \text{polylog } n) \) space bound. It is not hard to come up with an execution of the above algorithm where all edges are eventually stored in the stack, which may take up to \( \Omega(n^2 \text{polylog } n) \) space. To overcome this problem, we remove edges from within the stack, during the execution of the algorithm. The traditional local-ratio technique was not designed to work under space limitations, and thus does not guarantee any approximation ratio if edges are removed from the stack. The crux of our approach is a variation of the local-ratio technique, which provides conditions under which an edge may be removed from the stack while incurring only a small loss in the approximation ratio.

Specifically, we show that if an edge in the stack is significantly lighter than its neighboring edge, and this neighboring edge is added to the stack, then removing the light edge has only a small effect on the total weight of the solution. In order to use this conclusion, we must first assure a steady increase in the edge weights around each node. This again requires some sophistication beyond the classical local-ratio approach for the problem.

We achieve this by increasing the weight an edge reduces from its neighborhood to be \( \alpha \)-times its weight. This results in another deterioration in the approximation ratio, but has the benefit of forcing the weights of edges in the stack to exhibit an exponential growth pattern. This, in turn, creates the conditions for our modified local-ratio theorem to show its strength, allowing us to keep the size of the stack within the model’s limits. Choosing parameters that carefully manage the tradeoff between the space and the approximation ratio, we achieve a \((2 + \epsilon)\)-approximation using \( O(n \log^2 n) \) space.

Finally, we note that the basic structure of the local-ratio technique, namely processing the edges one by one in an arbitrary order and then performing some postprocessing, suits very naturally to the streaming environment. Combined with the machinery we develop here in order to follow the semi-streaming space constraints, we believe this technique can be applied to additional problems in the semi-streaming model and in similar computational models.

### 1.2.2 Approximating weighted vertex cover in the distributed setting

In [BCS16] we present a simple deterministic distributed \((2 + \epsilon)\)-approximation algorithm for minimum weight vertex cover (MWVC), which completes in \( O(\log \Delta/\epsilon \log \log \Delta) \) rounds, where \( \Delta \) is the maximum degree in the graph, for any \( \epsilon > 0 \) which is at most \( O(1) \), and in particular \( o(1) \). If \( \Delta \leq 16 \) then our algorithm simply requires \( O(1/\epsilon) \) rounds. Our algorithm adapts the local ratio technique [BYE85] to the distributed setting in a novel simple manner. Roughly speaking, in the simplest form of this technique, one repeatedly reduces the same amount of weight from both endpoints of an arbitrary edge, while not going below zero for any vertex. Terminating this process at the time in which for every edge there is at least one endpoint with no remaining weight, gives that the set of vertices with no remaining weight is a 2-approximation.
for MWVC. This can be extended to produce a \((2 + \epsilon)\)-approximation if instead the process terminates at the time in which for every edge there is at least one endpoint with a remaining weight of at most an \(\epsilon'\) fraction of its initial weight, where \(\epsilon' = \epsilon/(\epsilon + 2)\). The idea of ignoring objects with very small weights was already previously used in [BBF'01].

The challenge in translating this framework to the distributed setting is that the weights we can reduce from endpoints of neighboring edges must depend on each other. This is because we need to make sure that no weight goes below zero. However, as common to computing in this setting, we cannot afford long chains of dependencies, as these directly translate to a large number of communication rounds. Our key method is to divide the weight of a vertex into two parts, a vault from which it initiates requests for weight reductions with its neighbors, and a bank from which it reduces weight in response to requests from its neighbors. Carefully balancing these two reciprocal weight reductions at each vertex gives the claimed \((2 + \epsilon)\) approximation factor and \(O(\log \Delta/\epsilon \log \log \Delta)\) time complexity.

In fact, in our distributed algorithm, each vertex \(v\) with degree \(d(v)\) completes in \(O(1/\epsilon)\) rounds if \(d(v) \leq 16\), and in \(O(\log d(v)/\epsilon \log \log d(v))\) rounds otherwise (and requires no knowledge of \(n\) or \(\Delta\)). The algorithm also works in anonymous networks, i.e., no IDs are required. Moreover, the vertices are not required to start at the same round: as long as each vertex starts no later than after the first message has been sent to it, then each vertex completes within \(O(\log d(v)/\epsilon \log \log d(v))\) rounds after it starts (or in \(O(1/\epsilon)\) rounds if \(d(v) \leq 16\)). Finally, provided that the weights of all vertices as well as the ratio between the maximal and minimal weights fit in \(O(\log n)\) bits, our algorithm can be modified to work in the CONGEST model.

For any constant \(\epsilon\), our algorithm provides a constant approximation in \(O(\log \Delta/\epsilon \log \log \Delta)\) rounds. Apart from improving upon the previous best known complexity for distributed \((2 + \epsilon)\)-approximation algorithm for minimum weight vertex cover and providing a new way of adapting the sequential local ratio technique to the distributed setting, our algorithm has the consequence of contradicting the lower bound of [KMW10]. The latter states that a constant approximation algorithm requires \(\Omega(\log \Delta)\) rounds. Its refutation implies that the current lower bound is \(\Omega(\log \Delta/\log \log \Delta)\) from [KMW04], which means that our algorithm is tight. Following the publication of this result, the error was corrected in [KMW16].

### 1.2.3 Approximating weighted maximum matching and weighted maximum independent set in the distributed setting

In [BYCHGS] we present a simple distributed \(\Delta\)-approximation algorithm for maximum weight independent set (MaxIS) in the CONGEST model which completes in \(O(\text{MIS}(G) \cdot \log W)\) rounds, where \(\Delta\) is the maximum degree, \(\text{MIS}(G)\) is the number of rounds needed to compute a maximal independent set (MIS) on \(G\), and \(W\) is the maximum weight of a node. Plugging in the best known algorithm for MIS gives a randomized solution in \(O(\log n \log W)\) rounds, where \(n\) is the number of nodes. We also present a deterministic \(O(\Delta + \log^* n)\)-round algorithm based on coloring.

We then show how to use our MaxIS approximation algorithms to compute a 2-approximation for maximum weight matching without incurring any additional round penalty in the CONGEST model. We use a known reduction for simulating algorithms on the line graph while incurring
congestion, but we show our algorithm is part of a broad family of local aggregation algorithms for which we describe a mechanism that allows the simulation to run in the CONGEST model without an additional overhead.

**Δ-approximation algorithms for maximum weight independent set.** We present a simple distributed Δ-approximation algorithm for maximum weight independent set (MaxIS), where Δ is the maximum degree, which completes in \(O(\text{MIS}(G) \cdot \log W)\) rounds, where \(\text{MIS}(G)\) is the number of rounds needed to compute a maximal independent set (MIS) on \(G\), and \(W\) is the maximum weight of a node. As standard, we assume that \(W\) is at most polynomial in \(n\), so that the weight of each edge can be described in one message. Our algorithm adapts the local ratio technique [BYE85] for maximization problems [BBF01] to the distributed setting in a novel, yet simple, manner. Local ratio and MIS were previously used in [PRS12]. Moreover, [PS08] used the primal-dual schema and b-matching in a similar manner. This algorithms can be seen as a local ratio algorithm using the equivalence between the two paradigms [BYR05].

A simple local ratio algorithm for the problem goes as follows: one repeatedly picks a node \(v\) and reduces its weight from every \(u \in N(v)\). Every neighbor \(u \in N(v)\) whose weight becomes less than or equal to zero is removed from the graph, while \(v\) is added to a stack. We repeat this process with the induced graph until no nodes remain. We then begin popping nodes from the stack, adding them to the independent set if they have no neighbors in the set. This yields a Δ-approximation.

The challenge in translating this framework to the distributed setting is that if we allow all nodes to perform weight reductions simultaneously, then the above does not hold. For example, consider a star graph where the weight of the center is larger than the weight of any of its neighbors but smaller than their sum. After a single iteration the weights of all the nodes become negative, and no node gets selected. However, we show that if we first compute an independent set and then go on to perform weight reductions we achieve a Δ-approximation factor, while allowing using the power of parallelism. At each iteration we find an MIS, and the nodes chosen to the MIS perform weight reductions. This process is repeated until no nodes with positive weight remain. Nodes are then added to the independent set in reverse order of removal while maintaining the independence constraints. To analyze the running time, our main technique is to group the nodes into \(\log W\) layers based on their weight. At each iteration, all of the nodes from the topmost layer move to lower layers.

This results in a round complexity of \(O(\text{MIS}(G) \cdot \log W)\) in the CONGEST model. Our algorithm is deterministic apart from using a black-box algorithm to find an MIS at each iteration. Whether our algorithm is randomized or deterministic depends on the MIS algorithm it uses as a black-box.

We also present a deterministic coloring-based algorithm running in \(O(\Delta + \log^* n)\) rounds. Here we first color the graph using \(\Delta + 1\) colors, and then use each color group as an independent set to perform weight reductions as in the previous algorithm.

**2-approximation algorithms for maximum weighted matching.** We use a known reduction to simulate algorithms on the line graph [Kuh05], our MaxIS Δ-approximation algorithm gives a 2-approximation for maximum weight matching. Simulating an execution on the line
graph in a naive fashion results in a $O(\Delta)$ multiplicative overhead in the CONGEST model. We show our algorithm is part of a broad family of local aggregation algorithms for which we describe a mechanism which allows the simulation to run in the CONGEST model without added overhead.

Our deterministic coloring-based algorithm has a favorable running time compared to the algorithm presented in [EMR15] with parameters that result in a 2-approximation. Our randomized algorithm improves upon the $(2 + \epsilon)$-approximation factor of [LPSP15]. Using the MIS algorithm of [Gha16] we get a running time of $O((\log \Delta + 2^{O(\sqrt{\log\log n})}) \cdot \log W)$, with high probability, for the LOCAL model, and using Luby’s classical MIS algorithm [Lub86], we get an $O(\log n \cdot \log W)$ algorithm for the CONGEST model. For constant values of $W$, this is $O(\log n)$ rounds.

Optimal $O(\log \Delta / \log \log \Delta)$-round approximations for maximum weighted matching.

We then proceed to obtain faster approximation algorithms for maximum matching under either of the following relaxations.

First, we provide an optimal $O(\log \Delta / \log \log \Delta)$-round algorithm for a $(2 + \epsilon)$-approximation of maximum weighted matching, for any constant $\epsilon > 0$. Second, we address the unweighted case, and show an $O(\log \Delta / \log \log \Delta)$-round algorithm for a $(1 + \epsilon)$-approximation of maximum cardinality matching, for any constant $\epsilon > 0$. These are the first constant-approximation algorithms that attain the $\Omega(\log \Delta / \log \log \Delta)$ lower bound of Kuhn et al. [KMW06], which holds for any constant approximation. Moreover, they can be viewed as an improvement on the $O(\log n)$-round algorithms of Lotker et al. [LPSP15] for the same problems.

One of the key ingredients in both of the above fast algorithms is an improvement of the nearly-maximal independent set algorithm of Ghaffari [Gha16]. A nearly-maximal independent set is an independent set for which each node in the graph is in the set or has a neighbor in the set with probability at least $1 - \delta$ for a small $\delta$. Ghaffari gives a maximal independent set algorithm in $O(\log \Delta) + 2^{O(\sqrt{\log\log n})}$ rounds, in which the central building block is finding a nearly-maximal independent set in $O(\log \Delta)$ rounds. Here, we build upon his technique and extend it in a manner inspired by the recent vertex-cover approximation algorithm of Bar-Yehuda et al. [BCS16], which allows us to obtain a nearly-maximal independent set in $O(\log \Delta / \log \log \Delta)$ rounds.

While this does not allow us to improve upon Ghaffari’s maximal independent set algorithm, it helps us in obtaining our fast maximum matching approximation algorithms. For the $(2 + \epsilon)$-approximation, we run our new near-maximal independent set algorithm on the line graph, and obtain an approximation for the unweighted case in the CONGEST model by arguing that it is a local aggregation algorithm. Then, we extend this approximation ratio to the weighted case using techniques developed in [LPSR09, LPSP15].

For the unweighted $(1 + \epsilon)$-approximation, our goal is to use the general framework of Hopcroft and Karp [HK73], in which we repeatedly search for short non-intersecting augmenting paths and augment the matching with them, hence improving its size. However, in our setting, this does not work as is and poses significant challenges. One key challenge is that, to have the desired approximation factor, we need a much stronger near-maximality guarantee. It does not suffice to have a low probability for each short augmenting path to remain; we need to show that each
node has a low probability of having a remaining augmenting path. To overcome the obstacles, first we show how to find a nearly-maximal matching in low-rank hypergraphs and how to modify the algorithm for obtaining the \((1 + \epsilon)\)-approximation guarantee in the LOCAL model.

Making the algorithm suitable for the CONGEST model is even more demanding, in part because here we cannot explicitly work with the structure of the intersections between short augmenting paths; instead, we need to have a new variant of the near-maximal independent set algorithm that works on the fly. At a high level, we first address bipartite graphs, and show how to find a nearly-maximal independent set of short augmenting paths in them. Since the augmenting paths are not known explicitly, an interesting aspect here will be a variant of the dynamic probability adjustments in the algorithm of [Gha16]. Now, various nodes of a path might decide differently regarding whether to raise or lower its probability. However, we will prove that still the net effect provides a sufficient move in the right direction. We complete by generalizing this to all graphs, using the method of Lotker et al. [LPSP15], which essentially transforms the problem into randomly chosen bipartite subgraphs of it.

### 1.2.4 Distributed property testing

The performance of many distributed algorithms naturally depends on properties of the underlying network graph. Therefore, an inherent goal is to check whether the graph, or some given subgraph, has certain properties. However, in some cases this is known to be hard, such as in the CONGEST model [Pel00]. In this model, computation proceeds in synchronous rounds, in each of which every vertex can send an \(O(\log n)\)-bit message to each of its neighbors. Lower bounds for the number of rounds of type \(\tilde{\Omega}(\sqrt{n} + D)\) are known for verifying many global graph properties, where \(n\) is the number of vertices in the network and \(D\) is its diameter (see, e.g. Das-Sarma et al. [SHK+12]).

To overcome such difficulties, we adopt the relaxation used in graph property testing, as first defined in [GGR98, GR02], to the distributed setting. That is, rather than aiming for an exact answer to the question of whether the graph \(G\) satisfies a certain property \(P\), we settle for distinguishing the case of satisfying \(P\) from the case of being \(\epsilon\)-far from it, for an appropriate measure of being far.

Apart from its theoretical interest, this relaxation is motivated by the common scenario of having distributed algorithms for some tasks that perform better given a certain property of the network topology, or given that the graph almost satisfies that property. For example, Hirvonen et al. [HRSS14] show an algorithm for finding a large cut in triangle-free graphs (with additional constraints), and for finding an \((1 - \epsilon)\)-approximation if at most an \(\epsilon\) fraction of all edges are part of a triangle. Similarly, Pettie and Su [PS15] provide fast algorithms for coloring triangle-free graphs.

We construct fast distributed algorithms for testing various graph properties. An important byproduct of this study is a toolbox that we believe will be useful in other settings as well.

---

1Here \(\tilde{\Omega}\) hides factors that are polylogarithmic in \(n\).
Our contributions

We provide a rigorous study of property testing methods in the realm of distributed computing under the CONGEST model. We construct 1-sided error distributed $\epsilon$-tests, in which if the graph satisfies the property then all vertices output accept, and if it is $\epsilon$-far from satisfying the property then at least one vertex outputs reject with probability at least $2/3$. Using the standard amplification method of invoking such a test $O(\log n)$ times and having a vertex output reject if there is at least one invocation in which it should output reject, gives rejection with higher probability at the price of a multiplicative $O(\log n)$ factor for the number of rounds.

The definition of a graph being $\epsilon$-far from satisfying a property is roughly one of the following: (1) Changing any $\epsilon n^2$ entries in the adjacency matrix does not give a graph that satisfies the property (dense model), or (2) changing any $\epsilon \cdot \max\{n, m\}$ entries in the adjacency matrix does not give a graph that satisfies the property, where $m$ is the number of edges (general model). A particular case here is when the degrees are bounded by some constant $d$, and any resulting graph must comply with this restriction as well (sparse model).

In a sequential $\epsilon$-test, access to the input is provided by queries, whose type depends on the model. In the dense model these are asking whether two vertices $v, u$ are neighbors, and in the general and sparse models these can be either asking what the degree of a vertex $v$ is, or asking what the $i$-th neighbor of $v$ is (the ordering of neighbors is arbitrary). While a sequential $\epsilon$-test can touch only a small handful of vertices with its queries, in a distributed test the lack of ability to communicate over large distances is offset by having all $n$ vertices operating in parallel.

Our first contribution is a general scheme for a near-complete emulation in the distributed context of $\epsilon$-tests originating from the dense graph model (Section 7.1). This makes use of the fact that in the dense model all (sequential) testing algorithms can be made non-adaptive, which roughly means that queries do not depend on responses to previous queries. In fact, such tests can be made to have a very simple structure, allowing the vertices in the distributed model to “band together” for an emulation of the test. There is only one additional technical condition (which we define below), since in the distributed model we cannot handle properties whose counter-examples can be “split” to disjoint graphs. For example, the distributed model cannot hope to handle the property of the graph having no disjoint union of two triangles, a property for which there exists a test in the dense model.

**Theorem 7.1** Any $\epsilon$-test in the dense graph model for a non-disjointed property that makes $q$ queries can be converted to a distributed $\epsilon$-test that takes $O(q^2)$ communication rounds.

We next move away from the dense graph model to the sparse and general models, that are sometimes considered to be more realistic. In the general model, there exists no test for the property of containing no triangle that makes a number of queries independent of the number of graph vertices [AKKR08]. Here the distributed model can do better, because the reason for this deficiency is addressed by having all vertices operate concurrently. In Section 7.2 we adapt the interim lemmas used in the best testing algorithm constructed in [AKKR08], and construct a distributed algorithm whose number of rounds is independent of $n$.

**Theorem 7.2** There exists a distributed $\epsilon$-test in the general graph model for the property of containing no triangles, that requires $O(\epsilon^{-2})$ rounds.
The sparse and general models inherently require adaptive property testing algorithms, since there is no other way to trace a path from a given vertex forward, or follow its neighborhood. Testing triangle freeness sequentially uses adaptivity only to a small degree. However, other problems in the sparse and general models, such as the one we explore next, have a high degree of adaptivity built into their sequential algorithms, and we need to take special care for emulating it in the distributed setting.

In the sparse model (degrees bounded by a constant \(d\)), we adapt ideas from the bipartiteness testing algorithm of [GR99], in which we search for odd-length cycles. Here again the performance of a distributed algorithm surpasses that of the test (a number of rounds polylogarithmic in \(n\) vs. a number of queries which is \(\Omega(\sqrt{n})\) – a lower bound that is given in [GR02]). The following is proved in Section 7.3.

**Theorem 7.3** There exists a distributed \(\epsilon\)-test in the bounded degree graph model for the property of being bipartite, that requires \(O(\text{poly}(\epsilon^{-1} \log(n\epsilon^{-1})))\) rounds.

In the course of proving Theorem 7.3 we develop a method that we consider to be of independent interest\(^2\). The algorithm works by performing \(2n\) random walks concurrently (two starting from each vertex). The parallel execution of random walks despite the congestion restriction is achieved by making sure that the walks have a uniform stationary distribution, and then showing that congestion is “close to average”, which for the uniform stationary distribution is constant.

In Section 7.4 we show a fast test for cycle-freeness. This makes use of a combinatorial lemma that we prove, about cycles that remain in the graph after removing edges independently with probability \(\epsilon/2\). The following summarizes our result for testing cycle-freeness.

**Theorem 7.4** There exists a distributed \(\epsilon\)-test in the general graph model for the property of being cycle-free, that requires \(O(\log n/\epsilon)\) rounds.

We also prove lower bounds for testing bipartiteness and cycle-freeness (matching the upper bound for the latter). Roughly speaking, these are obtained by using the probabilistic method with alterations to construct graphs which are far from being bipartite or cycle-free, but all of their cycles are of length that is at least logarithmic. This technique bears some similarity to the classic result by Erdős [Erd59], which showed the existence of graphs with large girth and large chromatic number. The following are given in Section 7.5.

**Theorem 7.5** Any distributed \(1/100\)-test for the property of being bipartite requires \(\Omega(\log n)\) rounds of communication.

**Theorem 7.6** Any distributed \(1/100\)-test for the property of being cycle-free requires \(\Omega(\log n)\) rounds of communication.

### 1.2.5 Derandomizing local distributed algorithms

In [CPS16] we investigate the complexity of local problems in the domain that lies in the intersection of the randomized-deterministic gaps and the natural assumption of restricted

---

\(^2\)This technique was recently independently and concurrently devised in [GKS16] for a different use.
bandwidth. Specifically, we show how to derandomize distributed algorithms for local problems in the CONGEST model and in its relaxation known as the congested clique model, which further allows all-to-all communication (regardless of the input graph for which the problem needs to be solved), thereby abstracting away the effect of distances and focusing on the bandwidth restrictions. The essence of our derandomization technique lies in the design of a sample space that is easy to search by communicating only little information to a leader.

Our Contribution

Maximal Independent Set (MIS): We begin by derandomizing the MIS algorithm of Ghaffari [Gha16], which runs in $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds, w.h.p. In a nutshell, this algorithm works in constant-round phases, in which nodes choose to mark themselves with probabilities that evolve depending on the previous probabilities of neighbors. A marked node that does not have any marked neighbors joins the MIS and all of its neighbors remove themselves from the graph. The analysis shows that after $O(\log \Delta)$ phases the graph consists of a convenient decomposition into small clusters for which the problem can be solved fast. This is called the shattering phenomena, and it appears in several distributed algorithms for local problems (see, e.g., [BEPS12]).

We first show that a tighter analysis for the congested clique model of Ghaffari’s MIS algorithm can improve its running time from $O(\log \Delta + \log^* n)$ (which follows from combining [Gha16] with the new connectivity result of [GP16]) to $O(\log \Delta)$ rounds.

**Theorem 8.2** There is a randomized algorithm that computes MIS in the congested clique model within $O(\log \Delta)$ rounds with high probability.

For the derandomization, we use the method of conditional expectations (see e.g., [MU05, Chapter 6.3]). In our context, this shows the existence of an assignment to the random choices made by the nodes that attains the desired property of removing a sufficiently large part of the graph in each iteration, where removal is due to a node already having an output (whether the vertex is in the MIS or not). As in many uses of this method, we need to reduce the number of random choices that are made in order to be able to efficiently compute the above assignment.

However, we need to overcome several obstacles. First, we need to reduce the search space of a good assignment to the random choices of the nodes, by showing that pairwise independence (see, e.g., [MU05, Chapter 13]) is sufficient for the algorithm to work. Unfortunately, this does not hold directly in the original algorithm.

The first key ingredient is a slight modification of the constants used by Ghaffari’s algorithm. Ghaffari’s analysis is based on a definition of golden nodes, which are nodes that have a constant probability of being removed in the given phase. We show that this removal-probability guarantee holds also with pairwise independence upon our slight adaptation of the constants used by the algorithm.

Second, the shattering effect that occurs after $O(\log \Delta)$ rounds of Ghaffari’s algorithm with full independence, no longer holds under pairwise independence. Instead, we take advantage of the fact that in the congested clique model, once the remaining graph has a linear number of
edges then the problem can be solved locally in constant many rounds using Lenzen’s routing algorithm [Len13]. Thus, we modify the algorithm so that after $O(\log \Delta)$ rounds, the remaining graph (containing all undecided nodes) contains $O(n)$ edges. The crux in obtaining this is that during the first $O(\log \Delta)$ phases, we favor the removal of old nodes, which, roughly speaking, are nodes that had many rounds in which they had a good probability of being removed. This prioritized (or biased) removal strategy allows us to employ an amortized (or accounting) argument to claim that every node that survives $O(\log \Delta)$ rounds, can blame a distinct set of $\Delta$ nodes for not being removed earlier. Hence, the total number of remaining nodes is bounded by $O(n/\Delta)$, implying a remaining number of edges of $O(n)$.

To simulate the $O(\log \Delta)$ randomized rounds of Ghaffari’s algorithm, we enjoy the small search space (due to pairwise independence) and employ the method of conditional expectations on a random seed of length $O(\log n)$. Note that once we start conditioning on random variables in the seed, the random choices are no longer pairwise independent as they are in the unconditioned setting. However, we do not use the pairwise independence in the conditioning process. That is, the pairwise independence is important in showing that the unconditional expectation is large, and from that point on the conditioning does not reduce this value.

As typical in MIS algorithms, the probability of a node being removed stems from the random choices made in its 2-neighborhood. With a logarithmic bandwidth, collecting this information is too costly. Instead, we use a pessimistic estimator to bound the conditional probabilities rather than compute them.

Finally, to make the decision of the partial assignment and inform the nodes, we leverage the power of the congested clique by having a leader that collects the relevant information for coordinating the decision regarding the partial assignment. In fact, the algorithm works in the more restricted Broadcast Congested Clique model, in which a node must send the same $O(\log n)$-bit message to all other nodes in any single round. Carefully placing all the pieces of the toolbox we develop, gives the following.

**Theorem 8.3** There is a deterministic MIS algorithm for the broadcast congested clique model that completes in $O(\log \Delta \log n)$ rounds.

If the maximal degree satisfies $\Delta = O(n^{1/3})$ then we can improve the running time in the congested clique model. In fact, under this assumption we can obtain the same complexity for $(\Delta + 1)$-coloring.

**Theorem 8.4** If $\Delta = O(n^{1/3})$ then there is a deterministic MIS algorithm (and a $(\Delta + 1)$-coloring algorithm) for the congested clique model that completes in $O(\log \Delta)$ rounds.

Our techniques immediately extend to the CONGEST model. In that setting we show that MIS can be computed in $O(D \cdot \log^2 n)$ rounds where $D$ is the diameter of the graph. Here, we simulate $O(\log n)$ rounds of Ghaffari’s algorithm rather than $O(\log \Delta)$ rounds as before. Each such randomized round is simulated by using $O(D \cdot \log n)$ deterministic rounds in which the nodes compute a $O(\log n)$ seed. Computing each bit of the seed, requires aggregation of the statistics to a leader which can be done in $O(D)$ rounds, and since the seed is of length $O(\log n)$, we have the following:
Theorem 8.5 There is a deterministic MIS algorithm for the CONGEST model that completes in $O(D \log^2 n)$ rounds.

The significance of the latter is that it is the first deterministic MIS algorithm in CONGEST to have only a polylogarithmic gap compared to its randomized counterpart when $D$ is polylogarithmic. Notice that this logarithmic complexity is the best that is known even in the LOCAL model. In [PS96] it is shown that an MIS can be computed deterministically in $O(2^{\sqrt{\log n}})$ rounds via network decomposition, which is super-polylogarithmic in $n$. Moreover, the algorithm requires large messages and hence is suitable for the LOCAL model but not for CONGEST. Focusing on deterministic algorithms in CONGEST, the only known non-trivial solution is to use any $(\Delta + 1)$-coloring algorithm running in $O(\Delta + \log^* n)$ rounds (for example [BEK14, Bar15]) to obtain the same complexity for deterministic MIS in CONGEST (notice that there are faster coloring algorithms, but the reduction has to pay for the number of colors anyhow). Our $O(D \log^2 n)$-round MIS algorithm is therefore unique in its parameters.

Multiplicative Spanners: We further exemplify our techniques in order to derandomize the Baswana-Sen algorithm for constructing a $(2k - 1)$-spanner. Their algorithm runs in $O(k^2)$ rounds and produces a $(2k - 1)$-spanner with $O(kn^{1+1/k})$ edges. In a nutshell, the algorithm starts with a clustering defined by all singletons and proceeds with $k$ iterations, in each of which the clusters get sampled with probability $n^{1/k}$ and each node joins a neighboring sampled cluster or adds edges to unsampled clusters.

We need to make several technical modifications of our tools for this to work. The key technical difficulty is that we cannot have a single target function. This arises from the very nature of spanners, in that a small-stretch spanner always exists, but the delicate part is to balance between the stretch and the number of edges. This means that a single function which takes care of having a good stretch alone will simply result in taking all the edges into the spanner, as this gives the smallest stretch. We overcome this challenge by defining two types of bad events which the algorithm tries to avoid simultaneously. One is that too many clusters get sampled, and the other is that too many nodes add too many edges to the spanner in this iteration. The careful balance between the two promises that we can indeed get the desired stretch and almost the same bound on the number of edges.

Additional changes we handle are that when we reduce the independence, we cannot go all the way down to pairwise independence and we need settle for $d$-wise independence, where $d = \Theta(\log n)$. Further, we can improve the iterative procedure to handle chunks of $\log n$ random bits, and evaluate them in parallel by assigning a different leader to each possible assignment for them. A careful analysis gives a logarithmic overhead compared to the original Baswana-Sen algorithm, but we also save a factor of $k$ since the congested clique allows us to save the $k$ rounds needed in an iteration of Baswana-Sen for communicating with the center of the cluster. This gives the following.

Theorem 8.6 There is a deterministic $(2k - 1)$-spanner algorithm for the congested clique model that completes in $O(k \log n)$ rounds and produces a spanner with $O(kn^{1+1/k} \log n)$ edges.

As in the MIS algorithm, the above algorithm works also in the broadcast congested clique.
model, albeit here we lose the ability to parallelize over many leaders and thus we pay another logarithmic factor in the number of rounds, resulting in $O(k \log^3 n)$ rounds.
Chapter 2

Related work

2.1 Maximum matching in the semi-streaming model

The study of graph algorithms in the semi-streaming model was initiated by Feigenbaum et al. [FKM+05], in order to tackle the problem of processing massive graphs whose edge set cannot be stored in memory. The need for algorithms for such massive graphs is evident, as they become increasingly common: graphs representing social networks, graphs for metabolic interactions used in computational biology and even the communication graph of the Internet, are only a few examples.

Feigenbaum et al. were also the first to study the MWM problem in the semi-streaming model. They show a 6-approximation algorithm for MWM, which maintains a matching by adding an edge to it and removing the edge’s neighbors only if the edge’s weights is twice the weight of the removed edges. This idea was later improved by McGregor [McG05] to achieve an approximation ratio of 5.828, by changing the threshold for inserting an edge to the matching. McGregor also presents a \((2 + \epsilon)\)-approximation algorithm for the problem, but with \(O(\epsilon^3)\) passes on the input. By using the same ideas, while keeping deleted edges and reviving them later, Zelke [Zel12] achieves a 5.585-approximation algorithm.

A different approach was taken by Epstein et al. [ELMS11], who achieve a \((4.911 + \epsilon)\)-approximation algorithm. They use bucketing, i.e. separate the edges into \(O(\log n)\) weight classes, find a matching in each bucket, and then find the final matching in the union of these matching. Crouch and Stubbs [CS14] achieve an approximation ratio of \((4 + \epsilon)\) using related ideas, but their algorithm use weight classes which are unbounded from above, and thus are not disjoint. Recently, Grigorescu et al. [GMZ16] improved the analysis of the last algorithm, to show it achieves an approximation ratio of 3.5.

The bucketing technique takes a heavy toll on the approximation factor, and Crouch and Stubbs [CS14] prove this technique cannot give an approximation ratio better than 3.5. To circumvent this bound, we use a known approximation framework, the local ratio technique. To the best of our knowledge, this is the first use of this technique in the streaming environment.

The problem of MWM was also considered in other streaming models, such as the MapReduce model [CS14, LMSV11], the sliding-window model [CS14, CMS13] and the turnstile stream model (allowing deletions as well as insertions) [Kon15, AKLY16, BS15, CCE+16]. More general submodular-function matching problems in the semi-streaming model have been considered by
2.2 Distributed minimum vertex cover approximation

Minimum vertex cover is known to be one of Karp’s 21 NP-hard problems [Kar72]. For the unweighted case, a simple polynomial-time 2-approximation algorithm is obtained by taking the endpoints of a greedy maximal matching (see, e.g., [CLRS09, GJ79]). For the weighted case, the first polynomial-time 2-approximation algorithm was given in [NJ75] and observed by [Hoc82]. The first linear-time 2-approximation algorithm is due to [BE81] using the primal-dual framework, and [BYE85] gives a linear-time 2-approximation local-ratio algorithm. Conditioned on the unique games conjecture, minimum vertex cover does not have a \((2 - \epsilon)\) polynomial-time approximation algorithm [KR08].

In the distributed setting, an excellent summary of approximation algorithms is given in [˚AS10], which we overview in what follows. For the unweighted case, it is known how to find a 2-approximation in \(O(\log^4 n)\) rounds [HKP01] and in \(O(\Delta + \log^* n)\) rounds [PR01]. With no dependence on \(n\), [˚AFP+09] give a \(O(\Delta^2)\)-round 2-approximation algorithm, and [PS09] give an \(O(\Delta)\)-round 3-approximation algorithm. The maximal matching algorithm of [BEPS12] gives a 2-approximation for vertex cover in \(O(\log \Delta + (\log \log n)^4)\) rounds. This can be made into a \((2 + 1/poly\Delta)\)-approximation within \(O(\log \Delta)\) rounds [Pet16].

For the weighted case, [GKP08] give a randomized 2-approximation algorithm in \(O(\log n + \log W)\) rounds, where \(W\) is the maximal vertex weight. Later, [KY09] were the first to give a randomized 2-approximation algorithm in \(O(\log n)\) rounds, i.e., in a running time that is logarithmic in \(n\) and independent of the weights. In [PR01], a 2-approximation algorithm which requires \(O(\Delta + \log^* n)\) rounds, and in [KVF94], a \((2 + \epsilon)\)-approximation algorithm is given, requiring \(O(\log^{-1} \log n)\) rounds. With no dependence on \(n\), [KMW06] give a \((2 + \epsilon)\)-approximation algorithm in \(O(\epsilon^{-4} \log \Delta)\), [˚FP+09] give a 2-approximation algorithm in \(O(1)\) rounds for \(\Delta \leq 3\), and [˚AS10] give a 2-approximation algorithm in \(O(\Delta + \log^* W)\) rounds, where \(W\) is the maximal weight.

Distributed algorithms have also been devised for additional problems, which are related to vertex cover. The first distributed algorithm that uses the local ratio technique is due to [PRS12], who give an approximation for a cellular coverage problem (which is a maximization problem). In [GKPS08], a bi-criteria \((2 + \epsilon)\)-approximation algorithm for the capacitated vertex cover problem is given, in which each vertex has a limited capacity for the number of edges it can cover and the solution violates the capacity constraints by a factor of at most \((4 + \epsilon)\). In [BBF+01] a similar weight reduction technique is presented.

2.3 Distributed maximum independent set and maximum matching approximation

The maximum independent set problem is known to be NP-hard, as it is complementary to the maximum clique problem, which is one of Karp’s 21 NP-hard problems [Kar72]. In the sequential setting, an excellent summary of the known results is given by [Ban15], which we overview in
what follows. For general graphs, the best known algorithm achieves a \( O(n \log^2 \log n / \log^3 n) \)-approximation factor \([Fei04]\). Assuming \( NP \not\subseteq ZPP \), \([Hås96]\) shows that no \((n^{1-\epsilon})\)-approximation exists for every constant \( \epsilon > 0 \).

When the degree is bounded by \( \Delta \), a simple \((\Delta + 2)/3\)-approximation is achieved by greedily adding the node with minimal degree to the independent set and removing its neighbors \([HR97]\). The best known approximation factor is \( O(\Delta \log \log \Delta / \log \Delta) \) \([AK98, \text{Hal98, Hal02, Hal00, KMS98}]\). Conditioned on the Unique Games Conjecture, there exist a \( \Omega(\Delta/\log^2 \Delta) \)-approximation bound \([AKS09]\), where \( \Delta \) is constant or some mildly increasing function of \( n \). Assuming \( P \neq NP \), a bound of \( \Omega(\Delta/\log^4 \Delta) \) is given in \([Cha13]\).

As for the distributed case, \([LW08, \text{CHW08}]\) give a lower bound of \( O(\log^* n) \) rounds for any deterministic algorithm approximating MaxIS. While \([CHW08]\) provide both randomized and deterministic approximation algorithms for planar graphs. In \([BHKK16]\) a randomized \( O(n^\epsilon) \)-approximation running in \( O(1/\epsilon) \) rounds in the \textsc{Local} model is presented with a matching lower bound for the unweighted case.

Maximum matching is a classical optimization problem, for which the first polynomial time algorithm was given by Edmonds \([Edm65a, \text{Edm65b}]\) for both the weighted and unweighted case. In the distributed setting, the first algorithm for computing an approximate maximum matching was given in \([WW04]\), where a 5-approximation factor is achieved w.h.p for general graphs, in \( O(\log^2 n) \) rounds. In \([LPSR09]\) a randomized \((4 + \epsilon)\)-approximation for the weighted case is given, running in \( O(\log n) \) rounds for constant \( \epsilon > 0 \). This was later improved in \([LPSP15]\) to achieve a \((2 + \epsilon)\)-approximation in \( O(\log \epsilon^{-1} \log n) \) rounds. In \([EMR15]\) a deterministic \((1 + \epsilon)\)-approximation is given, in \( \Delta^{O(1/\epsilon)} + O(1/(\epsilon^2)) \cdot \log^* n \) rounds for the unweighted case, and \( \log(\min\{1/w_{\min}, n/\epsilon\})^{O(1/\epsilon)} \cdot (\Delta^{O(1/\epsilon)} + \log^* n) \) rounds for the weighted case, where the edge weights are in \([w_{\min}, 1]\). Due to \([KMW06]\), every algorithm achieving a constant approximation to the maximum matching problem requires \( \Omega(\min\{\log \Delta / \log \log \Delta, \sqrt{\log n / \log \log n}\}) \) rounds.

In \([PS08]\), a similar technique of weight grouping is used in the primal-dual framework for scheduling. In \([PS08]\) there is also a similar use of an MIS computation and a distributed stack.

### 2.4 Property testing

#### 2.4.1 Historical overview

The first papers to consider the question of property testing were \([BLR93]\) and \([RS96]\). The original motivations for defining property testing were its connection to some Computerized Learning models, and the ability to leverage some properties to construct Probabilistically Checkable Proofs (PCPs – this is related to property testing through the areas of Locally Testable Codes and Locally Decodable Codes, LTCs and LDCs). Other motivations since then have entered the fray, and foremost among them are sublinear-time algorithms, and other big-data considerations. Since virtually no property can be decidable without reading the entire input, property testing introduces a notion of the allowable approximation to the original problem. In general, the algorithm has to distinguish inputs satisfying the property, from inputs that are \( \epsilon\)-far from it. For more information on the general scheme of “classical” property testing, consult the surveys \([Ron08, Fis04, GR10]\)
The older of the graph testing models discussed here is the dense model, as defined in the seminal work of Goldreich, Goldwasser and Ron [GGR98]. The dense graph model has historically kick-started combinatorial property testing in earnest, but it has some shortcomings. Its main one is the distance function, which makes sense only if we consider graphs having many edges (hence the name “dense model”) – any graph with $o(n^2)$ edges is indistinguishable in this model from an empty graph.

The stricter and at times more plausible distance function is one which is relative to the actual number of edges, rather than the maximum $(\frac{n^2}{2})$. The general model was defined in [AKKR08], while the sparse model was defined already in [GR02]. The main difference between the sparse and the general graph models is that in the former there is also a guaranteed upper bound $d$ on the degrees of the vertices, which is given to the algorithm in advance (the query complexity may then depend on $d$, either explicitly, or more commonly implicitly by considering $d$ to be a constant).

2.4.2 Distributed property testing

The only previous work that directly relates to our distributed setting is due to Brakerski and Patt-Shamir [BP11]. They show a tolerant property testing algorithm for finding large (linear in size) near-cliques in the graph. An $\epsilon$-near clique is a set of vertices for which all but an $\epsilon$-fraction of the pairs of vertices have an edge between them. The algorithm is tolerant, in the sense that it finds a linear near-clique if there exists a linear $\epsilon^3$-near clique. That is, the testing algorithm considers two thresholds of being close to having the property (in this case – containing a linear size clique). We are unaware of any other work on property testing in this distributed setting.

Testing in a different distributed setting was considered in Arfaoui et al. [AFIM14]. They study testing for cycle-freeness, in a setting where each vertex may collect information of its entire neighborhood up to some distance, and send a short string of bits to a central authority who then has to decide whether the graph is cycle-free or not.

Related to having information being sent to, or received by, a central authority, is the concept of proof-labelling schemes, introduced by Korman et al. [KKP10] (for extensions see, e.g., Baruch et al. [BFP15]). In this setting, each vertex is given some external label, and by exchanging labels the vertices need to decide whether a given property of the graph holds. This is different from our setting in which no information other than vertex IDs is available. Another setting that is related to proof-labelling schemes, but differs from our model, is the prover-verifier model of Foerster et al. [FLSW16].

Sequential property testing has the goal of computing without processing the entire input. The wider family of local computation algorithms (LCA) is known to have connections with distributed computing, as shown by Parnas and Ron [PR07] and later used by others. A recent study by Göös et al. [GHL+15] proves that under some conditions, the fact that a centralized algorithm can query distant vertices does not help with speeding up computation. However, they consider the LOCAL model, and their results apply to certain properties that are not influenced by distances.

Finding induced subgraphs is a crucial task and has been studied in several different distributed models (see, e.g., [KMRS15, DKO12, CKK+15, DLP12]). Notice that for finding subgraphs,
having many instances of the desired subgraph can help speed up the computation, as in [DLP12].
This is in contrast to algorithms that perform faster if there are no or only few instances, as explained above, which is why we test for, e.g., the property of being triangle-free, rather for the property of containing triangles. (Notice that these are not the same, and in fact every graph with \(3/\epsilon\) or more vertices is \(\epsilon\)-close to having a triangle.)

Parallelizing many random walks was addressed in [AAK+11], where the question of graph covering via random walks is discussed. It is shown there that for certain families of graphs there is a substantial speedup in the time it takes for \(k\) walks starting from the same vertex to cover the graph, as compared to a single walk. No edge congestion constraints are taken into account. In [SNPT13], it is shown how to perform, under congestion, a single random walk of length \(L\) in \(\tilde{O}(\sqrt{LD})\) rounds, and \(k\) random walks in \(\tilde{O}(\sqrt{kLD} + k)\) rounds, where \(D\) is the diameter of the graph. Our method has no dependence on the diameter, allowing us to perform a multitude of short walks much faster.

2.5 Distributed derandomization

2.5.1 Distributed computation of MIS.

The complexity of finding a maximal independent set is a central problem in distributed computing and hence has been extensively studied. The \(O(\log n)\)-round randomized algorithms date back to 1986, and were given by Luby [Lub86], Alon et al. [ABI86] and Israeli and Itai [II86]. A recent breakthrough by Ghaffari [Gha16] obtained a randomized algorithm in \(O(\log \Delta) + 2^{O(\sqrt{\log \log n})}\) rounds. The best deterministic algorithm is by Panconesi and Srinivasan [PS92], and completes in \(2^{O(\sqrt{\log n})}\) rounds. On the lower bound side, Linial [Lin92] gave an \(\Omega(\log^* n)\) lower bounds for 3-coloring the ring, which also applies to finding an MIS. Kuhn et al. [KMW16] gave lower bounds of \(\Omega(\sqrt{\log n / \log \log n})\) and \(\Omega(\sqrt{\log \Delta / \log \log \Delta})\) for finding an MIS.

Barenboim and Elkin [BE13] provide a thorough tour on coloring algorithms (though there are some additional recent results). An excellent survey on local problems is given by Suomela [Suo13].

2.5.2 Distributed constructions of spanners.

The construction of spanners in the distribute setting has been studied extensively both in the randomized and deterministic setting [DG06, DGP07, DGPV08, DGPV09, Pet10]. A randomized construction was given by Baswana and Sen in [BS07]. They show that their well-known centralized algorithm can be implemented in the distributed setting even when using small message size. In particular, they show that a \((2k - 1)\) spanner with an expected number of \(O(n^{1+1/k})\) edges can be constructed in \(O(k^2)\) rounds in the CONGEST model. Derandomization of similar randomized algorithms has been addressed mainly in the centralized setting [RTZ05]. We emphasize that we need entirely different techniques to derandomize the Baswana-Sen algorithm compared with the centralized derandomization of [RTZ05].

The existing deterministic distributed algorithms for spanner are not based on derandomization of the randomized construction. They mostly use messages of unbounded size and are mainly based on sparse partitions or network decomposition. The state of the art is due to Derbel et al [DGPV08]. They provide a tight algorithm for constructing \((2k - 1)\)-spanners with
optimal stretch, size and construction time of $k$ rounds. This was complemented by a matching lower bound, showing that any (even randomized) distributed algorithm requires $k$ rounds in expectation. Much less efficient deterministic algorithms are known for the CONGEST model. The best bounds for constructing a $(2k - 1)$-spanner are $O(n^{1-1/k})$ rounds due to [DMZ10].

### 2.5.3 Algorithms in the congested clique.

The congested clique model was first addressed in Lotker et al. [LPPSP03], who raised the question of whether the global problem of constructing a minimum spanning tree (MST) can be solved faster on a communication graph with diameter 1. Since then, the model gained much attention, with results about its computational power given by Drucker et al. [DKO14], faster MST algorithms by Hegeman et al. [HPP+15] and Ghaffari and Parter [GP16], distance computation by Nanongkai [Nan14, HKN16] and Holzer and Pinsker [HP15], subgraph detection by Dolev et al. [DLP12], algebraic computations by Censor-Hillel et al. [CKK+15], and routing and load balancing by Lenzen [Len13], Lenzen and Wattenhofer [LW11], and Patt-Shamir and Teplitsky [PT11]. Local problems were addressed by Hegeman et al [HPS14] who study ruling sets. Connections to the MapReduce model is given by Hegeman and Pemmaraju [HP14].

### 2.5.4 Derandomization in the parallel setting.

Derandomization of local algorithms has attracted much attention in the *parallel* setting [ABI86, II86, MNN94, PSZ89, KW84, GS89, BR91, Han96, SK07, CGH13]. Luby [Lub93] showed that his MIS algorithm (and more) can be derandomized in the PRAM model using $O(m)$ machines and $O(\log^2 n \log \log n)$ time. In fact, this much simpler algorithm can also be executed on the congested clique model, resulting in an $O(\log^4 n)$ running time. Similar variants of derandomization for MIS, maximal matching and $(\Delta + 1)$-coloring were presented in [ABI86, II86]. Berger and Rompel [BR91] developed a general framework for removing randomness from RNC algorithms when polylogarithmic independence is sufficient. The parallel setting bears some similarity to the all-to-all communication model but the barriers in these two models are different mainly because the complexity measure in the parallel setting is the computation time while in our setting local computation is for free. This raises the possibility of obtaining much better results in the congested clique model compared to what is known in the parallel setting.
Chapter 3

Preliminaries

3.1 The Local-ratio technique

The local ratio technique is a common element in many of the works presented in this document. Thus, we briefly describe the technique.

The local ratio technique was first introduced in [BYE85] where the first linear time 2-approximation algorithm for vertex cover was presented. Initially the technique was only applicable for minimization problems, but was later also extended for maximization problems.

Let us state the local ratio technique for maximization problems (it can be stated equivalently for minimization problems). Let \( w \) be a profit vector, and let \( C \) be a set of feasibility constraints on vectors \( x \in \mathbb{R}^m \). We say that \( x \) is feasible if it satisfies all of the constraints in \( C \). We are interested in maximizing \( w \cdot x \) within the feasibility constraints \( C \). We present the local-ratio theorem for maximization problems [BBF*01].

**Theorem 3.1.** Let \( C \) be a set of feasibility constraints on vectors in \( \mathbb{R}^m \). Let \( w, w_s, w_r \in \mathbb{R}^m \) be vectors such that \( w = w_s + w_r \). Let \( x \in \mathbb{R}^n \) be a feasible solution (with respect to \( C \)) that is a \( p \)-approximation with respect to \( w_s \) and with respect to \( w_r \). Then \( x \) is a \( p \)-approximation with respect to \( w \) as well.

In order to convey the spirit of the technique, we show how the local ratio technique can be used to achieve a 2-approximate solution for maximum weighted matching. First, let us define the problem.

Let \( G = (V, E, w) \) be a simple graph with non-negative edge weights, \( w : E \to \mathbb{R}_{>0} \). Denote \( n = |V| \) and \( m = |E| \); the neighboring edges of \( e \) are \( N(e) = \{e' | e \cap e' = 1\} \), and \( N^+(e) = N(e) \cup \{e\} \).

**Maximum Weighted Matching** A matching in \( G \) is a set \( M \subseteq E \) of edges such that no two edges share a node. A maximum weight matching (MWM) in \( G \) is a matching \( M \) of maximum weight: for every matching \( M' \) in \( G \), we have \( \sum_{e \in M} w(e) \geq \sum_{e \in M'} w(e) \).

We represent edge weights and matchings using vectors indexed by the edges. The weight function is identified with a vector \( w \), where \( w[e] \) is the weight of the edge \( e \). A matching \( M \) is identified with its indicator vector \( x \), defined as \( x[e] = 1 \) if \( e \in M \), and \( x[e] = 0 \) otherwise.
The weight of a matching \( x \) is the value of the inner product of \( x \) and \( w \), denoted \( xw \). A set of feasibility constraints \( C \) on \( x \) is induced by the graph in a straightforward manner.

**Approximation Algorithms** A feasible matching \( x \) is said to be a \( p \)-approximation of a MWM in \( G \), for a constant \( p \geq 1 \), if every matching \( x^* \) satisfies \( x^*w \leq p \cdot xw \). An algorithm returning a \( p \)-approximation on every input graph is said to be a \( p \)-approximation algorithm for the MWM problem, and \( p \) is called the approximation ratio of the algorithm. Note that if \( p' > p \) than a \( p \)-approximation algorithm is also a \( p' \)-approximation algorithm.

### 3.1.1 A Simple Local-Ratio Approximation Algorithm for MWM

The basic building blocks in local-ratio algorithms are the weight reduction steps. A weight reduction step on \( G = (V,E,w) \) is done by defining two edge-weight functions on \((V,E)\), the subtracted graph with weight vector \( w_s \) and the residual graph with weight vector \( w_r \), such that \( w = w_s + w_r \). We apply weight reduction steps iteratively, while assuring that any \( p \)-approximate solution to \( w_s \) can be easily extended into a \( p \)-approximate solution to \( w_r \).

For the specific problem of MWM, a weight reduction step is done by picking an arbitrary edge \( e \in E \) of positive weight and reducing the weight of \( e \) from every \( e' \in N^+(e) \). This splits the weight vector \( w \) into two vectors, \( w_s \) and \( w_r \), as follows: \( w_r[e'] = w[e] \) for every \( e' \in N^+(e) \), \( w_r[e'] = 0 \) for every other edge, and \( w_s = w - w_r \). Any 2-approximate solution for the subtracted graph can be easily extended into a 2-approximate solution for the residual graph by making sure that at least some \( e' \in N^+(e) \) is in the solution: if this is not the case, we can add \( e \) to the solution without violating the constraints. As \( w_s[e] = 0 \), adding \( e \) to the solution does not change the solution’s value. Thus, we get a 2-approximate solution for both \( w_s \) and \( w_r \).

This simple technique is realized by Algorithm 1. First, it applies weight reduction steps iteratively using edges of positive reduced weight, splitting a weight function \( w_i \) into \( w_{i+1} \) (reduced) and \( w_{i+1} \) (residual) and keeping the edge in a stack. When no edges with positive reduced weights remain, the algorithm unwinds the stack and adds the edges greedily to the matching. When we unwind the stack we maintain a set of interim solutions \( \{x_i\} \). We use the local-ratio theorem to guarantee that every \( x_i \) is a 2-approximate solution for \( w_i \). Finally returning \( x_1 \) which is a 2-approximate solution for the original problem.

### 3.2 Additional background on property testing

While the introduction provided rough descriptions of the different property testing models, here we provide more formal definitions. The dense model for property testing is defined as follows.

**Definition 3.2.1** (dense graph model [GGR98]). The dense graph model considers as objects graphs that are given by their adjacency matrix. Hence it is defined by the following features.

- **Distance**: Two graphs with \( n \) vertices each are considered to be \( \epsilon \)-close if one can be obtained from the other by deleting and inserting at most \( \epsilon n^2 \) edges (this is, up to a constant factor, the same as the normalized Hamming distance).
Algorithm 1: MWM-simple \((V, E, w)\). A simple 2-approximation algorithm for MWM

\begin{algorithm}
\begin{algorithmic}[1]
\State \(S \leftarrow \text{empty stack}\)
\State \(w_1 \leftarrow w; i \leftarrow 1\)
\For {\(e_i \in E \text{ s.t } w_i(e_i) \geq 0\)}
\State \(S\text{.push}(e_i)\)
\State \(w_{i+1} \leftarrow w_i\) \hfill \text{/* Implicit: } w_{i+1} \leftarrow \vec{0}\)
\For {\(e' \in N^+(e_i)\)}
\State \(w_{i+1}[e'] \leftarrow w_i[e'] - w_i[e_i]\) \hfill \text{/* Implicit: } w_{i+1}[e'] \leftarrow w_i[e_i] \quad \text{for every } e' \in N^+(e_i)\)
\EndFor
\State \(i \leftarrow i + 1\)
\EndFor
\State \(k \leftarrow |S|\)
\State \(x_{k+1} \leftarrow \vec{0}\)
\For {\(i \leftarrow k \text{ down to } 1\)}
\State \(x_i \leftarrow x_{i+1}\)
\State \(e_i \leftarrow S\text{.pop}()\)
\If {\(\forall e \in N(e_i) : x_i[e] = 0\)}
\State \(x_i[e_i] \leftarrow 1\)
\EndIf
\EndFor
\State \text{return } x_1\)
\end{algorithmic}
\end{algorithm}

- **Querying scheme:** A single query of the algorithm consists of asking whether two vertices \(u, v \in V\) form a graph edge in \(E\) or not.

- **Allowable properties:** All properties have to be invariant under permutations of the input that pertain to graph isomorphisms (a prerequisite for them being graph properties).

The number of vertices \(n\) is given to the algorithm in advance.

As discussed earlier, the sparse and general models for property testing relate the distance function to the actual number of edges in the graph. They are formally defined as follows.

**Definition 3.2.2** (sparse [GR02] and general [AKKR08] graph models). These two models consider as objects graphs given by their adjacency lists. They are defined by the following features.

- **Distance:** Two graphs with \(n\) vertices and \(m\) edges (e.g. as defined by the denser of the two) are considered to be \(\epsilon\)-close if one can be obtained from the other by deleting and inserting at most \(\epsilon \max\{n, m\}\) edges\(^1\).

- **Querying scheme:** A single query consists of either asking what is the degree of a vertex \(v\), or asking what is the \(i\)’th neighbor of \(v\) (the ordering of neighbors is arbitrary).

- **Allowable properties:** All properties have to be invariant under graph isomorphisms (which here translate to a relabeling that affects both the vertex order and the neighbor ids obtained in neighbor queries), and reordering of the individual neighbor lists (as these orderings are considered arbitrary).

\(^1\)Sometimes in the sparse graph model the allowed number of changes is \(\epsilon dn\), as relates to the maximum possible number of edges; when \(d\) is held constant the difference is not essential.
We mainly refer to the distance functions of these models, and less so to the querying scheme, since the latter will be replaced by the processing scheme provided by the distributed computation model. Note that most property testing models get one bit in response to a query, e.g., “yes/no” in response to “is uv an edge” in the dense graph model. However, the sparse and general models may receive $\log n$ bits of information for one query, e.g., an id of a neighbor of a vertex. Also, the degree of a vertex, which can be given as an answer to a query in the general model, takes $\log n$ bits. Since the distributed CONGEST model allows passing a vertex id or a vertex degree along an edge in $O(1)$ rounds, we can equally relate to all three graph models.

Another important point is the difference between 1-sided and 2-sided testing algorithms, and the difference between non-adaptive and adaptive algorithms.

Definition 3.2.3 (types of algorithms). A property testing algorithm is said to have 1-sided error if there is no possibility of error on accepting satisfying inputs. That is, an input that satisfies the property will be accepted with probability 1, while an input $\epsilon$-far from the property will be rejected with a probability that is high enough (traditionally this means a probability of at least $2/3$). A 2-sided error algorithm is also allowed to reject satisfying inputs, as long as the probability for a correct answer is high enough (traditionally at least $2/3$).

A property testing algorithm is said to be non-adaptive if it decides all its queries in advance (i.e. based only on its internal coin tosses and before receiving the results of any query), while only its accept/reject output may depend on the actual input. An adaptive algorithm may make each query in turn based on the results of its previous queries (and, as before, possible internal coin tosses).

In the following we address both adaptive and non-adaptive algorithms. However, we restrict ourselves to 1-sided error algorithms, since the notion of 2-sided error is not a good match for our distributed computation model.

3.3 Derandomization tools

Our derandomization approach consists of first reducing the independence between the coin flips of the nodes. Then, we find some target function we wish to maintain during each iteration of the derandomized algorithm. Finally, we find a pessimistic estimator for the target function and apply the method of conditional expectations to get a deterministic algorithm. Below we elaborate upon the above ingredients.

$d$-wise independent random variables. In the algorithms we derandomize, a node $v \in V$ flips coins with probability $p$ of being heads. As we show, it is enough to assume only $d$-wise independence between the coin flips of nodes. We show how to use a randomness seed of only $t = d \max \{ \log n, \log 1/p \}$ bits to generate a coin flip for each $v \in V$, such that the coin flips are $d$-wise independent.

We first need the notion of $d$-wise independent hash functions as presented in [Vad12].

Definition 3.3.1 ([Vad12, Definition 3.31]). For $N, M, d \in \mathbb{N}$ such that $d \leq N$, a family of functions $\mathcal{H} = \{ h : [N] \rightarrow [M] \}$ is $d$-wise independent if for all distinct $x_1, x_2, ..., x_d \in [N]$, the
random variables $H(x_1),...,H(x_d)$ are independent and uniformly distributed in $[M]$ when $H$ is chosen randomly from $H$.

In [Vad12] an explicit construction of $H$ is presented, with parameters as stated in the next Lemma.

**Lemma 3.3.2 ([Vad12, Corollary 3.34]).** For every $\gamma, \beta, d \in \mathbb{N}$, there is a family of $d$-wise independent functions $H_{\gamma,\beta} = \{h : \{0,1\}^\gamma \to \{0,1\}^\beta\}$ such that choosing a random function from $H_{\gamma,\beta}$ takes $d$-max $\{\gamma, \beta\}$ random bits, and evaluating a function from $H_{\gamma,\beta}$ takes time $\text{poly}(\gamma, \beta, d)$.

Let us now consider some node $v \in V$ which needs to flip a coin with probability $p$ that is $d$-wise independent with respect to the coin flips of other nodes. Using Lemma 3.3.2 with parameters $\gamma = \lfloor \log n \rfloor$ and $\beta = \lfloor \log 1/p \rfloor$, we can construct $H$ such that every function $h \in H$ maps the ID of a node to the result of its coin flip. Using only $t$ random bits we can flip $d$-wise independent biased coins with probability $p$ for all nodes in $v$.

We define $Y$ to be a vector of $t$ random coins. Note we can also look at $Y$ as a vector of length $t/\log n$ where each entry takes values in $[\log n]$. We use the latter when dealing with $Y$. From $Y$ each node $v$ can generate its random coin toss by accessing the corresponding $h \in H$ and checking whether $h(ID(v)) = 0$. From Definition 3.3.1 it holds that $Pr[h(ID(v)) = 0] = 1/p$, as needed.

**The method of conditional expectations.** Next, we consider the method of conditional expectations. Let $\phi : A^k \to \mathbb{R}$, and let $X = (X_1,...,X_k)$ be a vector of random variables taking values in $A$. If $E[\phi(X)] \geq \alpha$ then there is an assignment of values $Z = (z_1,...,z_\ell)$ such that $\phi(Z) \geq \alpha$. We describe how to find the vector $Z$. We first note that from the law of total expectation it holds that $E[\phi(X)] = \sum_{a \in A} E[\phi(X) \mid X_1 = a] Pr[X_1 = a]$, and therefore for at least some $a \in A$ it holds that $E[\phi(X) \mid X_1 = a] \geq \alpha$. We set this value to be $z_1$. We then repeat this process for the rest of the values in $X$, which results in the vector $Z$. In order for this method to work we need it to be possible to compute the conditional expectation of $\phi(X)$. We now wish to use the method of conditional expectations after reducing the number of random bits used by the algorithm. Let us denote by $\tilde{\phi}$ the original vector of random bits used by the algorithm. Taking $Y$ as before to be the seed vector for $\tilde{\phi}$, we have that $\tilde{\phi}$ is a function of $Y$. We need to be able to compute $E[\phi(\tilde{\phi}(Y)) \mid y[1] = a_1,\ldots,y[i] = a_i]$ for all possible values of $i$ and $a_j, j \leq i$.

Computing the conditional expectations for $\phi$ might be expensive. For this reason we use a pessimistic estimator. A pessimistic estimator of $\phi$ is a function $\phi' : A^k \to \mathbb{R}$ such that for all values of $i$ and $a_j, j \leq i$ it holds that $E[\phi(\tilde{\phi}(Y)) \mid y_1 = b_1,\ldots,y_i = b_i] \geq E[\phi'(\tilde{\phi}(Y)) \mid y_1 = b_1,\ldots,y_i = b_i]$. If $\phi'$ is a pessimistic estimator of $\phi$, then we can use the method of conditional expectations on $\phi'$ and obtain $z_1,\ldots,z_n$, such that $\phi(z_1,\ldots,z_n) \geq \phi'(z_1,\ldots,z_n) \geq \alpha$.

**3.3.1 Lenzen’s routing algorithm.**

One important building block for algorithms for the congested clique model is Lenzen’s routing algorithm [Len13]. This procedure guarantees that if there is a component of an algorithm in which each node needs to send at most $O(n \log n)$ bits and receive at most $O(n \log n)$ bits then
$O(1)$ rounds are sufficient. This corresponds to sending and receiving $O(n)$ pieces of information of size $O(\log n)$ for each node. Intuitively, this is easy when each piece of information of a node has a distinct destination, by a direct message. The power of Lenzen’s routing is that the source-destination partition does not have to be uniform, and that, in fact, it can be not predetermined.

### 3.4 Mathematical background

An important role in our analyses is played by the Multiplicative Chernoff Bound (see, e.g., [MU05]), which we state it here for completeness.

**Fact 3.4.1.** Suppose that $X_1, ..., X_n$ are independent random variables taking values in \{0, 1\}. Let $X$ denote their sum and let $\mu = E[X]$ denote its expected value. Then, for any $\delta > 0$,

\[
Pr[X < (1 - \delta)\mu] < \left(\frac{e^{-\delta}}{(1 - \delta)(1 - \delta)}\right)^\mu,
\]

\[
Pr[X > (1 + \delta)\mu] < \left(\frac{e^{\delta}}{(1 + \delta)(1 + \delta)}\right)^\mu.
\]

Some convenient variations of the bounds above are:

\[
Pr[X \geq (1 + \delta)\mu] < e^{-\delta \mu / 3}, \quad \delta \geq 1
\]

\[
Pr[X \geq (1 + \delta)\mu] < e^{-\delta^2 \mu / 3}, \quad \delta \in (0, 1)
\]

\[
Pr[X \leq (1 - \delta)\mu] < e^{-\delta^2 \mu / 2}, \quad \delta \in (0, 1).
\]
Chapter 4

Approximating weighted maximum matching in the semi-streaming model

In this chapter we present a simple $(2 + \epsilon)$-approximation algorithm for the maximum weight matching (MWM) problem in the semi-streaming model. Our algorithm is deterministic, single-pass, requires only $O(1)$ processing time per incoming edge, and uses $O(n \log^2 n)$ space for any constant $\epsilon > 0$. This improves upon the previously best known approximation algorithm of Crouch and Stubbs [CS14, GMZ16], which achieves an approximation ratio of $(3.5 + \epsilon)$ and takes $O(\log n)$ time to process an edge.

4.1 Approximating Maximum Weight Matching

In this section we extend local-ratio theorem for maximization problems [BBF+01], and use it in order to adjust the sequential local-ratio algorithm (Algorithm 1, presented in Section 3.1) to the semi-streaming model, incurring only a small loss in the approximation ratio.

4.1.1 Extending the Local-Ratio Technique

We now extend the approximation techniques used in Algorithm 1. This allows us to present another sequential approximation algorithm for MWM in the following section, with a worse approximation ratio of $2 + \epsilon$ for $\epsilon > 0$. However, from the new algorithm we derive the desired approximation algorithm for the semi-streaming model, with no further increase in the approximation ratio.

If instead of reducing exactly $w[e]$ from the neighboring edges of $e$, we reduce $w[e]$ or $\alpha w[e]$ from each such edge, for some $\alpha \geq 1$, we get a $(2\alpha)$-approximation, as formalized in the next claim.
Lemma 4.1.1. Let $w, w_s$ and $w_r$ be weight functions and $e \in E$ an edge, such that

$$w_r[e'] = \begin{cases} 
  w[e] & e' = e; \\
  \alpha w[e] \text{ or } w[e] & e' \in N(e); \\
  0 & \text{otherwise},
\end{cases}$$

(4.1)

and $w_s = w - w_r$; the choice between $w[e]$ and $\alpha w[e]$ can be arbitrary.

Let $x \in \{0,1\}^m$ be a matching. If $x[e'] \neq 0$ for some $e' \in N^+(e)$, then $x$ is a $(2\alpha)$-approximate solution for $w_r$.

Proof. Let $x^*$ be any matching. The definition of $w_r$ guarantees that $x^*$ contains at most two edges of non-zero weight in $w_r$, each of weight at most $\alpha w[e]$, so $x^* w_r \leq 2\alpha w[e]$. On the other hand, $x[e'] \neq 0$ for some $e' \in N^+(e)$, so $w[e] \leq xw_r$. The claim follows.

Next, we note that if the optimal solution for the subtracted graph is greater than the optimal solution for the residual graph by some multiplicative factor $p \geq 1$, then it is also a $(1 + 1/p)$-approximation for the original graph. For large values of $p$, an approximate solution for the subtracted graph gives roughly the same approximation ratio for the original graph, which allows us to ignore the residual graph. We formalize this in the next lemma.

Lemma 4.1.2. Let $w, w_s$ and $w_r$ be weight functions satisfying $w = w_s + w_r$ and $w_s[e] \leq w[e]$ for all $e \in E$. Let $x_s$ be a $\beta$-approximate solution for $w_s$.

If $x_s w_s$ is at least $p$ times larger than any matching in $w_r$, then $x_s$ is a $(\beta + 1/p)$-approximate solution for $w$.

Proof. Let $x^*, x^*_s$ and $x^*_r$ be matchings of maximum weights in $w, w_s$ and $w_r$ respectively.

The assumptions imply $x^* w_s \leq \beta x_s w_s$ and $p x^*_r w_r \leq x_s w_s$, so

$$x^* w = x^* w_s + x^* w_r \leq x^*_s w_s + x^*_r w_r$$

$$\leq \beta x_s w_s + (1/p)x_s w_s$$

$$= (\beta + 1/p)x_s w_s \leq (\beta + 1/p)x_s w,$$

where the last inequality follows from the fact that $w_s[e] \leq w[e]$ for all $e \in E$.

Let $w_1$ be a weight vector for the MWM problem, and consider an iterative splitting of $w_i$ into $w_{i+1}$ and $w_{i+1}$ for $k$ times. The last lemma allowed us to ignore the residual graph once; we now extend it to allow the iterative omission of the residual graph.

Denote $\alpha = \sqrt{1 + \epsilon/2}$, $\gamma = n^2/\ln(\alpha)$, and $\beta_i = 2\alpha(1 + 1/\gamma)^{k+1-i}$ for all $i$.

Lemma 4.1.3. Let $G = (V, E, w_1)$ a graph, and $w_2, \ldots, w_k+1$ and $w_2, \ldots, w_{k+1}$ sequences of reduced and residual weight functions for $(V, E)$, respectively.

Assume that we generate a sequence of solutions $x_{k+1}, \ldots, x_i$, such that $x_{k+1}$ is an optimal solution for $w_{k+1}$, and that for $1 \leq i \leq k$, if $x_{i+1}$ is a $\beta_{i+1}$-approximate solution for $w_{i+1}$ then $x_i$ has the following properties:

1. $x_i$ is a $\beta_{i+1}$-approximate solution for $w_{i+1}$.
2. At least one of the following holds:

(a) \( x_i \) is a \( \beta_{i+1} \)-approximate solution for \( w_{i+1} \); or
(b) \( x_i w_{i+1} \geq (\gamma/\beta_{i+1})x^*w_{i+1} \) for every solution \( x^* \).

Then, \( x_1 \) is a \( \beta_1 \)-approximate solution for \( w_1 \).

Proof. We prove, by induction on \( i \) ranging from \( k+1 \) down to 1, that \( x_i \) is a \( \beta_i \)-approximate solution for \( w_i \).

The base, \( i = k + 1 \), is trivial by the assumption on \( x_{k+1} \).

Assume the claim is true for \( x_{i+1} \), then condition 1 holds for \( x_i \). If condition 2(a) holds, then by condition 1 and the local-ratio theorem (Theorem 3.1), \( x_i \) is a \( \beta_{i+1} \)-approximate solution for \( w_i \). Because \( \beta_i > \beta_{i+1} \), \( x_i \) is also a \( \beta_i \)-approximate solution for \( w_i \). If condition 2(b) holds, then from condition 1 and Lemma 4.1.2 we deduce that \( x_i \) is a \( (\beta_{i+1} + \beta_{i+1}/\gamma) \)-approximate solution for \( w_i \). The definition of \( \beta_i \) yields:

\[
\beta_{i+1} + \beta_{i+1}/\gamma = (1 + 1/\gamma) \cdot 2\alpha(1 + 1/\gamma)^{k+1-(i+1)} = 2\alpha(1 + 1/\gamma)^{k+1-i} = \beta_i.
\]

Specifically, \( x_1 \) is a \( \beta_1 \)-approximate solution for \( w_1 \), and the proof is complete.

\[\square\]

4.2 A Semi-Streaming Algorithm

We present a \( (2 + \epsilon) \)-approximation algorithm for the MWM problem, using our extensions to the local-ratio technique. This algorithm could be used in the semi-streaming model if no space constraints applied. We give a detailed analyses of this algorithm, and then present a lightweight variant of it which obeys the space constraints.

The new algorithm is similar to Algorithm 1: it performs a series of weight reduction steps generating a series of reduced weight functions \( \{w_i\} \), and then constructs a series of approximate solutions \( \{x_i\} \). To prove the desired approximation ratio is achieved, we use Lemma 4.1.3 as a substitute for the local-ratio theorem.

We start by presenting the challenges posed by the semi-streaming model, and the ways in which the new algorithm deals with them.

Retroactive weight reduction The sequential algorithm constructs \( w_{i+1} \) from \( w_i \) using an edge \( e_i \), by reducing \( w_i(e_i) \) form the weight of every \( e' \in N^+(e_i) \). This cannot be done directly in the semi-streaming model, as some edges of \( N^+(e_i) \) may not have arrived yet. Instead, the algorithm keeps a variable \( \phi_i(v) = \sum_{j=1}^i w_j[e_j] \) for every node \( v \in V \). When a new edge \( e = (u, v) \) arrives, its reduced weight is computed retroactively by reducing \( \phi_{i-1}(u) \) and \( \phi_{i-1}(v) \) from its original weight.

Removing edges from the stack In the sequential algorithm, the stack may grow to hold all of the graph edges. Lemma 4.1.3 presents conditions under which an approximate solution for \( w_{i+1} \) is also an approximate solution for \( w_i \). When these conditions hold, we may remove the edge \( e_i \) from the stack, which we use in order to make sure the stack’s size does not exceed

33
$O(n \text{ polylog } n)$ edges. The new algorithm does not discard edges from the stack, but only replaces them with the ⊥ symbol; this is only done for the sake of analysis.

**Assuring edge-weight growth** In order to make sure edges are removed from the stack, we force a small but consistent growth in the edge weights around each node. Roughly speaking, the edge weights grow exponentially by a multiplicative $\alpha$ factor; after a logarithmic number of new edges considered, the weights grow large enough to allow the algorithm to neglect the older edges and remove them from the stack.

### 4.2.1 Algorithm MWM-seq

Algorithm MWM-seq (Algorithm 2) has two phases: in the first phase, it iterates over the edges and pushes chosen edges into a stack. In the second phase, the edges are popped out of the stack and added greedily to the matching.

The algorithm begins with an edge-weight function $w_1$, given as input. For each node $u$, the algorithm explicitly maintains a non-negative weight function $\phi_i(u)$, which is used to filter edges (Line 6): an edge $e = (u, u')$ processed at iteration $i$ is *light* if $w_1[e] \leq \alpha(\phi_{i-1}(u) + \phi_{i-1}(u'))$, and *heavy* otherwise. Iterations are defined by the heavy edges. When the first heavy edge in iteration $i-1$ arrives, it is denoted $e_i$, and iteration $i$ begins. Thus, all heavy edges are eventually denoted with sub-indexes ($e_i$), while the light edges are left un-tagged ($e$).

When an edge $e = (u, v)$ is processed in iteration $i$, the algorithm performs all weight reduction steps on $e$ retroactively using $\phi_{i-1}(v)$ and $\phi_{i-1}(u)$, to set the value of $w_i[e]$. It decides between reducing $\phi_{i-1}(v) + \phi_{i-1}(u)$ or $\alpha(\phi_{i-1}(v) + \phi_{i-1}(u))$ from the weight of $e$, in a way that guarantees an exponential growth of $\phi$, implying a bound on the size of the stack.

For every node $u$, we hold a queue $E_i(u)$. This is a list of heavy edges containing the node $u$ currently present in the stack. Upon seeing a heavy edge, $e_i = (u, v)$, we perform a weight reduction step: $\phi_{i-1}(u)$ and $\phi_{i-1}(v)$ are increased by $w_i[e_i]$, and $e_i$ is pushed into the stack. We also enqueue $e_i$ in $E_i(u), E_i(v)$. If the size of $E_i(u)$ exceeds a certain bound, we dequeue an edge from $E_i(u)$, and remove it from the stack. For the sake of analysis, we insert ⊥ in its place in the stack.

In the second phase, the algorithm unwinds the stack, adding edges greedily to the matching while ignoring the ⊥ symbol. The usage of the ⊥ symbol is replaced by deletion of the relevant edge in the semi-streaming algorithm, presented in the next subsection.

We start the analyses of Algorithm MWM-seq by proving that the node-weight functions $\phi_i(u)$ grow exponentially. In the algorithm, the variable $c_i(u)$ counts the heavy edges containing $u$ that arrived until iteration $i$. Its value is not used in the algorithm itself; we only use it in the proof, to bound from below the growth $\phi(u)$. In various places in the proof we consider the expression $c_i(u) - c_j(u)$. This is the amount of heavy edges added to $u$ from iteration $j$ until iteration $i$. We will eventually show that the reduced weights of heavy edges exhibit a growth pattern exponential in $c_i(u) - c_j(u)$. 

---

34
Algorithm 2: MWM-seq \((V,E,w)\). A sequential approximation algorithm for MWM

1. \(S \leftarrow \text{empty stack}\)
2. \(w_1 \leftarrow w; \phi_0 \leftarrow \vec{0}; c_0 \leftarrow \vec{0}\)
   
   /* \(c_i\) is only used for the proof */
3. \(\forall v \in V : E_0(v) \leftarrow \text{empty queue}\)
4. \(i \leftarrow 1\)

5. foreach \(e = (u,u') \in E\) do
6.   if \(w_1[e] \leq \alpha(\phi_{i-1}(u) + \phi_{i-1}(u'))\) then
7.     continue
8.     /* Implicit: \(w_{j+1}[e] \leftarrow \alpha w_j[e]\) for every \(e_j \in N(e)\) */
9.     \(e_i \leftarrow e\)
10. \(S\).push\((e_i)\)
11. \(w_i[e_i] \leftarrow w_1[e_i] - (\phi_{i-1}(u) + \phi_{i-1}(u'))\)
12.   /* Implicit: \(w_{j+1}[e_i] \leftarrow w_j[e_j]\) for every \(e_j \in N^+(e_i)\) */
13. \(\phi_i \leftarrow \phi_{i-1}; E_i \leftarrow E_{i-1}; c_i \leftarrow c_{i-1}\)
14. foreach \(v \in e\) do
15.   \(c_i(v) \leftarrow c_i(v) + 1\)
16.   \(E_i(v)\).enqueue\((e_i)\)
17.   \(\phi_i(v) \leftarrow \phi_{i-1}(v) + w_i[e_i]\)
18.   if \((\alpha - 1)\alpha|E_i(v)| - 2 > 2\alpha\gamma\) then
19.     \(e_j \leftarrow E_i(v)\).dequeue()
20.     \(S[j] \leftarrow \perp\)
21.   \(i \leftarrow i + 1\)
22. \(k \leftarrow |S|\)
23. \(x_{k+1} \leftarrow \vec{0}\)
24. for \(i \leftarrow k\) down to 1 do
25.   \(x_i \leftarrow x_{i+1}\)
26.   \(e_i \leftarrow S\).pop\()\)
27.   if \(e_i = \perp\) then
28.     continue
29.   if \(\forall e \in N(e_i) : x_i[e] = 0\) then
30.     \(x_i[e_i] \leftarrow 1\)
31. return \(x_1\)
Lemma 4.2.1. For every \( u \in V \) and \( i \geq j \), it holds that \( \phi_i(u) \geq \alpha^{c_i(u) - c_j(u)} \phi_j(u) \).

Proof. We fix \( j \), and prove by induction on \( i \), where \( i \geq j \). The base case, \( i = j \), is trivial. For \( i > j \), we consider two cases: if \( u \notin e_i \) then \( c_i(u) = c_{i-1}(u) \), so \( \phi_i(u) = \phi_{i-1}(u) \geq \alpha^{c_{i-1}(u) - c_j(u)} = \alpha^{c_i(u) - c_j(u)} \) by the induction hypothesis.

Otherwise, \( e_i = (u, u') \) for some \( u' \in V \), so \( c_i(u) = c_{i-1}(u) + 1 \), and

\[
\phi_i(u) = \phi_{i-1}(u) + w_i[e_i] \\
\geq \phi_{i-1}(u) + (\alpha - 1)(\phi_{i-1}(u) + \phi_{i-1}(u')) \\
\geq \phi_{i-1}(u) + (\alpha - 1)\phi_{i-1}(u) \\
= \alpha\phi_{i-1}(u) \\
\geq \alpha \cdot \alpha^{c_{i-1}(u) - c_j(u)} \phi_j(u) \\
= \alpha^{c_i(u) - c_j(u)} \phi_j(u)
\]

The first equality is due to Line 15, the first inequality follows Lines 6 and 10, and the last two transitions use the induction hypothesis and the fact that \( c_i(u) = c_{i-1}(u) + 1 \). \qed

Consider the sequences of reduced and residual edge-weight functions, \( w_2, \ldots, w_{k+1} \) and \( w_2, \ldots, w_{k+1} \), induced by the algorithm. While \( w_i[e_i] \) is defined explicitly in the algorithm, the other values are only set after the first phase ends, and so does \( k \), the length of the sequences.

The weight functions are defined inductively as follows. We formally define \( w_1 = w \), where \( w \) is the function given as input. The edge \( e_i \) is used to split the weight function \( w_i \) into \( w_{i+1} \) and \( w_{i+1} \), the latter defined by

\[
w_{i+1}[e'] = \begin{cases} 
w_i[e_i] & e' = e_i; \\
w_i[e_i] & e' \in N(e_i) \text{ and } e' \text{ is heavy}; \\
\alpha w_i[e_i] & e' \in N(e_i) \text{ and } e' \text{ is light}; \\
0 & \text{otherwise.}
\end{cases}
\]

and the former by \( w_{i+1} = w_i - w_{i+1} \). The length \( k \) is the number of heavy edges encountered in the first phase. Note that \( w \) is non-negative, so \( w_i[e] \) is a non-increasing function of \( i \), for any fixed edge \( e \).

The next lemma focuses on a node \( u \) and the heavy edges adjacent to it. It asserts that for \( i > j \), the reduced weight at iteration \( j + 1 \) of a heavy edge \( e_i \) grows exponentially with respect to \( w_j[e_j] \).

Lemma 4.2.2. Let \( e_i, e_j \in E \) such that \( i > j \) and \( e_i \cap e_j = \{u\} \). Then \( w_{j+1}[e_i] > (\alpha - 1)\alpha^{c_i(u) - c_j(u) - 1} w_j[e_j] \).
Proof. The lemma follows by a simple computation. As \( w_i(e) \) is a non-increasing
\[
\begin{align*}
    w_{j+1}(e_i) & \geq w_i(e_i) \\
    & \geq (\alpha - 1)i_{j-1}(u) \quad \text{(Lines 6 and 10)} \\
    & \geq (\alpha - 1)i_{c_i-1}(u) - c_j(u) \phi_j(u) \quad \text{(Lemma 4.2.1)} \\
    & \geq (\alpha - 1)i_{c_i-1}(u) - c_j(u) w_j[e_j] \quad \text{(Line 15)} \\
    & = (\alpha - 1)i_{c_i(u)} - c_i(u) - 1 w_j[e_j] \quad (u \in e_i \text{ implies } c_i-1(u) = c_i(u) - 1)
\end{align*}
\]
as desired.

In the second loop of the algorithm, the edges are taken out of the stack and a solution is
greedily constructed. The algorithm’s approximation ratio is the approximation ratio of the
solution \( x_1 \) on the original weight function \( w_1 \). To bound this quantity, we prove by induction
that every \( x_i \) is a \( \beta_i \)-approximate solution for \( w_i \). We break our analysis into cases, for which we
need the next three lemmas.

Lemma 4.2.3. If \( x_{i+1} \) is a \( \beta_{i+1} \)-approximate solution for \( w_{i+1} \) and the condition in Line 25
holds for \( e_i \), then \( x_i \) is a \( \beta_i \)-approximate solution for \( w_i \).

Proof. Since the condition in Line 25 holds, we have \( x_i = x_{i+1} \). This immediately guarantees
that \( x_i \) is a feasible solution and that condition 1 of Lemma 4.1.3 holds. We show that condition
2(b) of Lemma 4.1.3 holds as well.

Let \( e_i = (u, u') \). Because the condition in Line 25 holds, we know that in some iteration \( i' \),
\( i' > i \), the condition in Line 16 held as well. That is, \( e_{i'} \) was enqueued into \( E_{i'}(v) \), the condition
\((\alpha - 1)\alpha_{E_{i'}(v)} - 2 \geq 2\alpha \gamma \) held, and \( e_i \) was then dequeued from \( E_{i'}(v) \). Every enqueue operation
to \( E_i(v) \) is accompanied by an increases of \( c_i(v) \) by 1, so when the condition in Line 16 was
checked, \( e_i \) and \( e_{i'} \) were the first and last elements in \( E_{i'}(v) \), and the size of \( E_{i'}(v) \) was exactly
\( c_{i'}(v) - c_i(v) + 1 \). Thus, \((\alpha - 1)\alpha_{E_{i'}(v)} - c_i(v) - 1 \geq 2\alpha \gamma \).

Using this inequality and Lemma 4.2.2, we have
\[
    w_{i+1}[e_{i'}] \geq (\alpha - 1)\alpha_{E_{i'}(v)} - c_i(v) - 1 w_i[e_i] \geq 2\alpha \gamma w_i[e_i].
\]
Hence, the single edge \( e_{i'} \) is a matching of weight at least \( 2\alpha \gamma w_i[e_i] \) in \( w_{i+1} \). As \( x_{i+1} \) is a
\( \beta_{i+1} \)-approximate solution for \( w_{i+1} \), we have \( \beta_{i+1} x_{i+1} w_{i+1} \geq 2\alpha \gamma w_i[e_i] \).

The definition of \( w_{i+1} \) guarantees it has the following structure:
\[
    w_{i+1}[e'] = \begin{cases} 
        w_i[e_i] & e' = e_i; \\
        \alpha w_i[e_i] \text{ or } w_i[e_i] & e' \in N(e_i); \\
        0 & \text{otherwise}.
    \end{cases} \quad (4.3)
\]
Thus, any solution \( x^* \) for \( w_{i+1} \) contains at most two edges, of weight at most \( \alpha w_i[e_i] \), i.e.
\( 2\alpha w_i[e_i] \geq x^* w_{i+1} \). The last two inequalities guarantee any solution \( x^* \) satisfies
\[
    (\beta_{i+1}/\gamma) x_{i+1} w_{i+1} \geq 2\alpha w_i[e_i] \geq x^* w_{i+1}
\]
so \( x_iw_{i+1} = x_{i+1}w_{i+1} \geq (\gamma/\beta) x^*w_{i+1} \), and condition 2(b) of Lemma 4.1.3 holds.

\[ \square \]

**Lemma 4.2.4.** If \( x_{i+1} \) is a \( \beta_{i+1} \)-approximation for \( w_{i+1} \) and the condition on line 25 does not hold for \( e_i \), then \( x_i \) is a \( \beta_i \)-approximation for \( w_i \).

**Proof.** If the condition on Line 27 holds, then \( x_i \) is derived from \( x_{i+1} \) by adding \( e_i \) to \( x_{i+1} \). The condition in this line, together with the assumption that \( x_{i+1} \) is a matching, guarantee that \( x_i \) is a matching. Since \( w_{i+1}[e_i] = x_i[e_i] \) and \( w_{i+1} = w_i - w_{i+1} \), we have \( w_{i+1}[e_i] = 0 \). Hence, \( x_iw_{i+1} = x_{i+1}w_{i+1} \), so \( x_i \) is also a \( \beta_{i+1} \)-approximate solution for \( w_{i+1} \) and condition 1 of Lemma 4.1.3 holds. By Lemma 4.1.1, \( x_i \) is a \( (2\alpha) \)-approximate solution for \( w_{i+1} \), and because \( 2\alpha \leq \beta_{i+1} \) it is also a \( \beta_{i+1} \)-approximate solution to \( w_{i+1} \) and condition 2(a) of Lemma 4.1.3 holds.

Finally, if the condition on line Line 27 does not hold, we set \( x_i = x_{i+1} \). Then \( x_i \) is a feasible matching satisfying condition 1 of Lemma 4.1.3. The condition in Line 27 does not hold, so \( x_{i+1}[e_i'] \neq 0 \) for some \( e_i' \in N^+[e_i] \), and Lemma 4.1.1 promises \( x_i \) is a \( (2\alpha) \)-approximation for \( w_{i+1} \). As before, \( 2\alpha \leq \beta_{i+1} \) proves condition 2(a) of Lemma 4.1.3 holds.

The next lemma asserts that when the first phase ends, none of the reduced edge weights is positive.

**Lemma 4.2.5.** At the end of the first phase, \( w_{k+1}[e] \leq 0 \) for all \( e \in E \).

**Proof.** Consider an edge \( e \). If \( e = e_i \) is heavy then \( w_{i+1}[e_i] = x_i[e_i] \) and \( w_{i+1} = w_i - w_{i+1} \) imply \( w_{i+1}[e_i] = 0 \). The monotonicity of \( w_i[e] \) completes the proof.

If \( e = (u, u') \) is a light edge considered in iteration \( i \), then
\[
\phi_{i-1}(u) = \sum_{\{e_j \mid e_j \in e_i, j \leq i-1\}} w_j[e_j],
\]
and a similar claim holds for \( u' \). On the other hand, \( w_{j+1} = w_j - w_{j+1} \) and \( w_{j+1}[e] = \alpha w_j[e_j] \) for all \( e_j \in N(e) \). Hence
\[
w_{j+1}[e] = w_j[e] - \alpha w_j[e_j],
\]
and a simple induction implies
\[
w_i[e] = w_1[e] - \alpha \sum_{\{e_j \mid e_j \in N(e), j \leq i-1\}} w_j[e_j].
\]

The last two equalities, together with the definition of \( N(e) \), imply \( w_i[e] = w_1[e] - \alpha(\phi_{i-1}(u) + \phi_{i-1}(u')) \). The inequality \( w_i[e] \leq \alpha(\phi_{i-1}(u) + \phi_{i-1}(u')) \) implies \( w_i[e] \leq 0 \) for all \( e \in E \), and the monotonicity of \( w_i[e] \) completes the proof.

\[ \square \]

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

**Theorem 4.1.** Algorithm \textit{MWM-seq} returns a \((2 + \epsilon)\)-approximation for the MWM problem.

**Proof.** By Lemma 4.2.5, the first loop ends when \( w_{k+1} \leq 0 \), so \( x_{k+1} = 0 \) is indeed an optimal solution for \( w_{k+1} \).

38
Algorithm 3: MWM-semi \((V, E, w)\). A Semi-Streaming approximation algorithm for MWM

1. \(S \leftarrow\) empty stack
2. \(\phi \leftarrow \vec{0}\)
3. \(\forall v \in V : E(v) \leftarrow\) empty queue
4. foreach \(e = (u, u') \in E\) do
   5. if \(w[e] \leq \alpha (\phi(u) + \phi(u'))\) then continue
   6. \(S.\)push\((e)\)
   7. \(w'[e] \leftarrow w[e] - (\phi(u) + \phi(u'))\)
   8. foreach \(v \in \{u, u'\}\) do
      9. \(E(v).\)enqueue\((e)\)
     10. \(\phi(v) \leftarrow \phi(v) + w'[e]\)
     11. if \((\alpha - 1)\alpha |E(v)|^{-2} > 2\alpha \gamma\) then
        12. \(e' \leftarrow E(v).\)dequeue\()\)
        13. remove \(e'\) from \(S\)
15. \(M \leftarrow \emptyset\)
16. while \(S \neq \emptyset\) do
   17. \(e \leftarrow S.\)pop\()\)
   18. if \(M \cap N(e) = \emptyset\) then
      19. \(M \leftarrow M \cup \{e\}\)
20. return \(M\)

Assume \(x_{i+1}\) is a \(\beta_{i+1}\)-approximate solution for \(w_{i+1}\). From Lemmas 4.2.3 and 4.2.4 we conclude that in all cases the conditions of Lemma 4.1.3 hold, so \(x_1\) is a \(\beta_1\)-approximate solution for \(w = w_1\).

Substitute \(\beta_1 = 2\alpha(1 + 1/\gamma)^k\), \(\alpha = \sqrt{1 + \epsilon/2}\) and \(\gamma = n^2/\ln(\alpha)\), and note \(k \leq m \leq n^2\), to get

\[
\beta_1 \leq 2\alpha(1 + 1/\gamma)n^2 = 2\alpha \left(1 + (\ln \alpha)/n^2\right)n^2 \leq 2\alpha e^{\ln \alpha} \leq 2\alpha^2 = 2 + \epsilon.
\]

The desired approximation ratio is achieved. \(\square\)

4.2.2 Implementing Algorithm MWM-seq in the Semi-Streaming Model

In the previous section we showed that Algorithm MWM-seq computes a \((2 + \epsilon)\)-approximation for MWM. In the semi-streaming model, we must obey space constraints in addition to maintaining a good approximation ratio. In the presentation of the sequential algorithm we ignored the space constrains: we did not remove edges from the stack, and we represented the temporary solutions as the vectors \(x_i\) of size \(\Theta(n^2)\).

In order to follow the space constraints, we replace any insertion of \(\bot\) into the stack by a removal of the relevant edge, and the vectors \(x_i\) by a single set containing the current matching. For the sake of completeness, we present Algorithm MWM-semi (Algorithm 3), an implementation of Algorithm MWM-seq in the semi-streaming model. The correctness of Algorithm MWM-semi is derived directly from the correctness of Algorithm MWM-seq, so we only need to prove it obeys the space constraints.
After omitting notations and auxiliary variables from Algorithm MWM-seq, we are only left with three types of data structures in Algorithm MWM-semi: \( M \) is the matching constructed, \( S \) is the stack and \( E(v) \) is a queue of edges from \( S \) that contain node \( v \). Every edge \((u, v)\) that is added to \( S \) is also added to \( E(v) \) and \( E(u) \). When \((u, v)\) is removed from \( S \), it is also removed from \( E(u) \) and \( E(v) \), implying \(|S| \leq \sum_v |E(v)|\). The next lemma bounds the size of \( E(v) \) for every \( v \).

**Lemma 4.2.6.** During the execution of Algorithm MWM-semi, \(|E(v)| = O\left(\frac{\log n + \log(1/\epsilon)}{\epsilon}\right)\) for each \( v \in V \).

**Proof.** After each iteration of the loop in Lines 4–14, we have \((\alpha - 1)\alpha |E(v)| - 2 \leq 2\alpha \gamma\) for each \( v \in V \): this is true at the beginning; \( E(v) \) can grow only by 1 at each iteration; and whenever the inequality does not hold, an edge is removed from \( E(v) \).

From the above inequality, we derive an asymptotic bound for \(|E(v)|\).

\[
|E(v)| = O\left(\frac{\alpha \gamma}{\alpha - 1}\right)
= O\left(\frac{\log \gamma + \log \alpha - \log(\alpha - 1)}{\log \alpha}\right)
= O\left(\frac{\log \gamma - \log(\alpha - 1)}{\log \alpha}\right)
= O\left(\frac{\log \gamma - \log((\alpha^2 - 1)/(\alpha + 1))}{\log \alpha}\right)
= O\left(\frac{\log \gamma - \log(\alpha^2 - 1)}{\log \alpha}\right).
\]

Plugging in \( \alpha = \sqrt{1 + \epsilon/2} \) and \( \gamma = n^2 / \ln(\alpha) \) we get

\[
|E(v)| = O\left(\frac{\log \gamma - \log(\alpha^2 - 1)}{\log \alpha}\right)
= O\left(\frac{\log(n^2/\ln \alpha) + \log(1/\epsilon)}{\log(1 + \epsilon)}\right)
= O\left(\frac{\log n + \log(1/\epsilon)}{\log(1 + \epsilon)}\right)
= O\left(\frac{\log n + \log(1/\epsilon)}{\epsilon}\right).
\]

where the last step uses the inequality \( \ln(1 + x) \geq x/(1 + x) \) for \( x > -1 \).

From Lemma 4.2.6 we conclude that for a constant \( \epsilon \), Algorithm MWM-semi maintains at most \( O(n \log n) \) edges, each of size \( O(\log n) \). Thus, the total space used is \( O(n \log^2 n) \). Our algorithm requires \( O(1) \) time to process a new edge arriving from the stream, and finally we execute a post-processing step which requires \( O(n \log n) \) time. We arrive at the main theorem of this section:

**Theorem 4.2.** There exists an algorithm in the semi-streaming model computing a \((2 + \epsilon)\)-approximation for MWM, using \( O(\epsilon^{-1} n \log n \cdot (\log n + \log(1/\epsilon))) \) space and having an \( O(1) \) processing time.
In our analysis we assume that the weights of edges can be represented using $O(\log n)$ bits. If this is not the case, and the weights are bounded by some $W$, our algorithm requires $O(n(\log^2 n + \log W))$ space, as we keep a sum of weights for every node. For every $W$ that can be represented using a polylogarithmic number of bits, this is still $O(n \text{ polylog } n)$ space.
Chapter 5

Approximating weighted vertex cover in the distributed setting

In this chapter we present a simple deterministic distributed \((2 + \epsilon)\)-approximation algorithm for minimum weight vertex cover (MWVC), which completes in \(O(\log \Delta/\epsilon \log \log \Delta)\) rounds, where \(\Delta\) is the maximum degree in the graph, for any \(\epsilon > 0\) which is at most \(O(1)\), and in particular \(o(1)\). If \(\Delta \leq 16\) then our algorithm simply requires \(O(1/\epsilon)\) rounds.

For any constant \(\epsilon\), our algorithm provides a constant approximation in \(O(\log \Delta/\log \log \Delta)\) rounds. Apart from improving upon the previous best known complexity for distributed \((2 + \epsilon)\)-approximation algorithm for minimum weight vertex cover and providing a new way of adapting the sequential local ratio technique to the distributed setting, our algorithm has the consequence of contradicting the lower bound of \([KMW10]\). The latter states that a constant approximation algorithm requires \(\Omega(\log \Delta)\) rounds. Its refutation implies that the current lower bound is \(\Omega(\log \Delta/\log \log \Delta)\) from \([KMW04]\), which means that our algorithm is tight. Following the publication of this result, the error was corrected in \([KMW16]\).

In Section 5.4 we pinpoint the flaw in the lower bound of \([KMW10]\). This also includes refuting the second result of \([KMW10]\), which is a lower bound in terms of \(n\), of \(\Omega(\sqrt{\log n})\) rounds for a constant approximation algorithm. Roughly speaking, we claim that the statement of the main theorem is only correct for some smaller range of parameters than claimed, and hence, in particular, one cannot apply it for a number of rounds that is \(\Theta(\log \Delta)\) or \(\Theta(\sqrt{\log n})\). We emphasize that, as far as we are aware, this bug does not occur in the previous version of the lower bound \([KMW04]\), implying that the current lower bounds are \(\Omega(\sqrt{\log n}/\log \log n)\) in terms of \(n\), and \(\Omega(\log \Delta/\log \log \Delta)\) in terms of \(\Delta\).

5.1 A local ratio template for approximating MWVC

In this section we provide the template for using the local-ratio technique (presented in Section 3.1) for obtaining a \((2 + \epsilon)\)-approximation for MWVC. This template does not assume any specific computation model and only describes the paradigm and correctness. It can be proven either using the primal-dual framework \([BE81]\), or the local-ratio framework \([Bar00]\), which are known to be equivalent \([BR05]\). A similar idea, though in the primal-dual framework, was given in \([KVY94]\) which obtained a \((2 + \epsilon)\)-approximation as well, but with a larger number of rounds.
Our distributed implementation is more efficient and allows us to obtain a faster algorithm. Here we provide the template and proof for completeness. In Section 5.2 we provide a distributed implementation of the template and analyze its running time.

We assume a given weighted graph \( G = (V, w, E) \), where \( w : V \rightarrow \mathbb{R}^+ \) is an assignment of weights for the vertices. Let \( \delta : E \rightarrow \mathbb{R}^+ \) be a function that assigns weights to edges. We say that \( \delta \) is \( G \)-valid if for every \( v \in V \), \( \sum_{e \in E} \delta(e) \leq w(v) \), i.e., the sum of weights of edges that touch a vertex is at most the weight of that vertex in \( G \).

Fix any \( G \)-valid function \( \delta \). Define \( \tilde{\delta} : V \rightarrow \mathbb{R}^+ \) by \( \tilde{\delta}(v) = \sum_{e \in E} \delta(e) \), and let \( w' : V \rightarrow \mathbb{R}^+ \) be such that \( w'_\delta(v) = w(v) - \tilde{\delta}(v) \). Since \( \delta \) is \( G \)-valid, it holds that \( w'_\delta(v) \geq 0 \) for every \( v \in V \).

Let \( S_\delta = \{ v \in V \mid w'_\delta(v) \leq \epsilon'w(v) \} \), where \( \epsilon' = \epsilon/(2 + \epsilon) \). The following theorem states that if \( S_\delta \) is a vertex cover, then it is a \((2 + \epsilon)\)-approximation for MWVC.

**Theorem 5.1.** Fix \( \epsilon > 0 \) and let \( \delta \) be a \( G \)-valid function. Let \( OPT \) be the sum of weights of vertices in a minimum weight vertex cover \( S_{OPT} \) of \( G \). Then \( \sum_{v \in S_\delta} w(v) \leq (2 + \epsilon)OPT \). In particular, if \( S_\delta \) is a vertex cover then it is a \((2 + \epsilon)\)-approximation for MWVC for \( G \).

**Proof.** For every \( v \in V \) we have that \( w'_\delta(v) = w(v) - \tilde{\delta}(v) \), which implies that \( w(v) = w'_\delta(v) + \tilde{\delta}(v) \). For every \( v \in S_\delta \) it holds that \( w'_\delta(v) \leq \epsilon'w(v) \), and therefore \( w(v) \leq \epsilon'w(v) + \tilde{\delta}(v) \).

Put otherwise, for every \( v \in S_\delta \) we have \( w(v) \leq (1/(1 - \epsilon'))\tilde{\delta}(v) \). This gives:

\[
\sum_{v \in S_\delta} w(v) \leq \frac{1}{1 - \epsilon'} \sum_{v \in S_\delta} \tilde{\delta}(v)
\leq \frac{1}{1 - \epsilon'} \sum_{v \in S_\delta} \sum_{e \in E} \delta(e)
\leq \frac{1}{1 - \epsilon'} \sum_{v \in V} \sum_{e \in E} \delta(e)
\leq \frac{1}{1 - \epsilon'} \cdot 2 \sum_{e \in E} \delta(e).
\]

The above is at most \((2/(1 - \epsilon'))OPT \) because \( OPT \geq \sum_{e \in E} \delta(e) \). To see why \( OPT \geq \sum_{e \in E} \delta(e) \), associate each edge \( e \) with its endpoint \( v_e \) in \( S_{OPT} \) (choose an arbitrary endpoint if both are in \( S_{OPT} \)). The weight \( w(v) \) of each \( v \in S_{OPT} \) is at least \( \sum_{e: v_e = v} \delta(e) \), because it is at least \( \sum_{e: v_e = v} \delta(e) \). Hence, \( OPT = \sum_{v \in S_{OPT}} w(v) \geq \sum_{e \in E} \delta(e) = \sum_{e \in E} \delta(e) \). Hence the sum of weights in \( S_\delta \) is at most a factor \( 2/(1 - \epsilon') \) larger than \( OPT \). Since \( \epsilon' = \epsilon/(2 + \epsilon) \), we have that \( 2/(1 - \epsilon') = (2 - 2\epsilon' + 2\epsilon)/(1 - \epsilon') = 2(1 + \epsilon'/(1 - \epsilon')) = 2 + \epsilon \), which completes the proof. \( \Box \)

In the next section, we show how to implement efficiently in a distributed setting an algorithm that finds a function \( \delta \) that is \( G \)-valid, for which the set \( S_\delta \) is a vertex cover. This immediately gives a distributed \((2 + \epsilon)\)-approximation for MWVC.

### 5.2 A fast distributed implementation

Our goal in this section is to find a \( G \)-valid function \( \delta \) such that \( S_\delta \) is a vertex cover. Since every vertex knows whether it is in \( S_\delta \), this immediately gives a distributed \((2 + \epsilon)\)-approximation algorithm for MWVC. Our algorithm is deterministic and requires for every vertex \( v \) only
\(O(\log d(v) / \epsilon' \log \log d(v))\) rounds, where \(d(v)\) is the degree of \(v\) in \(G\), or \(O(1/\epsilon')\) if \(d(v) \leq 16\). Here \(\epsilon' = \epsilon/(2 + \epsilon)\) where \(\epsilon = O(1)\), which means that \(\epsilon' = \Theta(\epsilon)\). For clarity of presentation, in this section we describe an implementation for the LOCAL model. In Section 5.3 we show how this can be easily be adapted to the CONGEST model in which the message size is limited to \(O(\log n)\) bits, provided that the initial weights of the vertices and the ratio between the maximal and minimal weights can be expressed by \(O(\log n)\) bits.

In our algorithm, each vertex converges to agreeing with each of its neighbors on a function \(\delta\) that is \(G\)-valid, by iterating the process of decreasing the weight of neighbors by the same amount, until either its weight is below a small fraction of its original weight or it has no more neighbors in the graph induced by the vertices that remain so far. This would imply that the set of vertices whose weight decreased below the above threshold is a vertex cover, and by Theorem 5.1 its weight is a \((2 + \epsilon)\)-approximation to the weight of a minimal vertex cover.

**Overview of Algorithm 4** The algorithm consists of iterations, each of which has a constant number of communication rounds. Each vertex \(v\) splits its current weight \(w_i(v)\) into two amounts. The first amount is \(\text{vault}_i(v)\), which is equal to its threshold \(\epsilon' w_0(v)\), and the second amount is \(\text{bank}_i(v)\), which contains the rest of the weight \(w_i(v) - \text{vault}_i(v)\). Notice that \(\epsilon' < 1\) because \(\epsilon' = \epsilon/(2 + \epsilon)\) and therefore these amounts are well-defined.

In each iteration, vertex \(v\) sends a \(\text{request}_i(v, u)\) request to its neighbor \(u\), which is the amount in its \(\text{vault}_i(v)\) divided by the current number of neighbors of \(v\). This guarantees that any weight decrease that results for \(v\) from this part does not exceed its total remaining weight. The second amount is used to respond to \(\text{request}_i(u, v)\) requests from its neighbors. The vertex \(v\) processes these requests one by one in any arbitrary order, and responds with the amount \(\text{budget}_i(v, u)\) which is the largest amount by which \(v\) can currently decrease its weight, and no more than the request \(\text{request}_i(u, v)\). This amount is decreased from \(\text{bank}_i(v, u)\), and hence it is also guaranteed that decreasing this amount does not exceed the total remaining weight.

Once the weight of \(v\) reaches its threshold, \(v\) completes its algorithm, returning \(\text{InCover}\) after notifying its neighbors of this fact. If the weight is still above the threshold then \(v\) only removes its edges to neighbors that notified they are returning \(\text{InCover}\). This gives that each edge has at least one endpoint returning \(\text{InCover}\), and hence the set of all such vertices is a vertex cover, and by Theorem 5.1 its weight is a \((2 + \epsilon)\)-approximation to the weight of a minimal vertex cover. The analysis of the number of rounds is based on the observation that for each of the neighbors \(u\) of \(v\), either it decreases its weight by responding to \(\text{request}_i(v, u)\) with the entire amount and thereby contributes to decreasing the weight of \(v\) by this amount, or it does not have the required budget in its \(\text{bank}_i(u)\), in which case it contributes to decreasing the number of neighbors of \(v\) by 1.

We proceed by the full pseudocode, followed by an explicit analysis.

**Theorem 5.2.** For every \(\epsilon = O(1)\), Algorithm 4 is a deterministic distributed \((2 + \epsilon)\)-approximation algorithm for MWVC in which each vertex \(v\) with degree \(d(v)\) completes in \(O(1/\epsilon)\) rounds if \(d(v) \leq 16\), and in \(O(\log d(v) / \epsilon \log \log d(v))\) rounds otherwise.

We split the proof into two parts, showing correctness and complexity separately. We begin by showing correctness in the following lemma. Essentially, we show that the algorithm finds a
Algorithm 4: A distributed $(2 + \epsilon)$-approximation algorithm for MWVC, code for vertex $v$.

1. $w_0(v) = w(v)$
2. $d_0(v) = d(v)$
3. $N_0(v) = N(v)$
4. $i = 0$
5. $vault(v) = \epsilon'w_0(v)$

while true do

7. $bank_i(v) = w_i(v) - vault(v)$
8. $w_{i+1}(v) = w_i(v)$
9. $N_{i+1}(v) = N_i(v)$

foreach $u \in N_i(v)$ do

11. $request_i(v, u) = vault(v) / d_i(v)$
12. Send $request_i(v, u)$ to $u$
13. Let $budget_i(u, v)$ be the response from $u$
14. $w_{i+1}(v) = w_{i+1}(v) - budget_i(u, v)$
15. if $budget_i(u, v) < request_i(v, u)$ then
16.   $N_{i+1}(v) = N_{i+1}(v) \setminus \{u\}$

Let $u_1 \ldots u_{d_i(v)}$ be an order of $N_i(v)$

foreach $k = 1, \ldots, d_i(v)$ do

19. Let $request_i(u_k, v)$ be received from $u_k \in N_i(v)$
20. $budget_i(v, u_k) = \min\{request_i(u_k, v), bank_i(v) - \sum_{t=1}^{k-1} budget_i(v, u_t)\}$
21. Send $budget_i(v, u_k)$ to $u_k$
22. $bank_{i+1}(v) = bank_i(v) - \sum_{k=1}^{d_i(v)} budget_i(v, u_k)$
23. $w_{i+1}(v) = w_{i+1}(v) - \sum_{k=1}^{d_i(v)} budget_i(v, u_k)$
24. $d_{i+1}(v) = |N_{i+1}(v)|$
25. $i = i + 1$
26. if $w_i(v) \leq \epsilon'w_0(v)$ then
27.   Send $(v, cover)$ to all neighbors
28.   Return InCover

foreach $(u, cover)$ received from $u \in N_i(v)$ do

30. $N_i(v) = N_i(v) \setminus \{u\}$
31. $d_i(v) = d_i(v) - 1$
32. if $d_i(v) = 0$ then
33.   Return NotInCover
function \( \delta \) that is \( G \)-valid and for which \( S_4 \) is a vertex cover.

**Lemma 5.2.1.** Algorithm 4 is a deterministic distributed \((2 + \epsilon)\)-approximation algorithm for MWVC.

**Proof.** We first show that Algorithm 4 is a \((2 + \epsilon)\)-approximation algorithm for MWVC. That is, we claim that the set \( C = \{ v \in V \mid v \text{ outputs InCover} \} \) is a vertex cover, and that \( \sum_{v \in C} w(v) \leq (2 + \epsilon)OPT \), where \( OPT = \sum_{v \in S_{OPT}} w(v) \) for some optimal vertex cover \( S_{OPT} \). For this, we show that the sum of amounts deducted by neighbors can be used to define a \( G \)-valid function over the edges. This will be exactly the function according to which the vertices decide whether to output InCover or NotInCover.

For every \( e = \{v, u\} \in E \) and every \( i = 0, 1, \ldots, \) let \( \delta_i(e) = \text{budget}_i(u, v) + \text{budget}_i(v, u) \).

Let \( \delta(e) = \sum_{i=0,1,\ldots} \delta_i(e) \). We claim that \( \delta \) is \( G \)-valid, i.e., for every vertex \( v \) it holds that \( \sum_{e : v \in e} \delta(e) \leq w(v) \). Let \( j \) be the value of \( i \) when \( v \) returns, that is, \( v \) participates in iterations \( i = 0, \ldots, j - 1 \). For each iteration \( i = 0, \ldots, j - 1 \) it holds that

\[
\sum_{u_k \in N_i(v)} \text{budget}_i(u_k, v) \leq \sum_{u_k \in N_i(v)} \text{vault}(v)/d_i(v) = \text{vault}(v),
\]

where \( N_i(v) = \{u_1, \ldots, u_{d_i(v)}\} \) is the set of neighbors of \( v \) at the beginning of iteration \( i \). Further, since for \( u_k \in N_i(v) \), \( \text{budget}_i(u_k, v, u_k) = \min\{\text{request}_i(u_k, v), \text{bank}_i(v) - \sum_{t=1}^{k-1} \text{budget}_i(v, u_t)\} \), we have that

\[
\sum_{u_k \in N_i(v)} \text{budget}_i(v, u_k) \leq \text{bank}_i(v).
\]

Since \( \text{bank}_i(v) = w_i(v) - \text{vault}(v) \) it holds that \( \sum_{e = \{v, u_k\} : u_k \in N_i(v)} \delta_i(e) \leq w_i(v) \). Since \( w_{i+1}(v) = w_i(v) - \sum_{(u, v) \in E} (\text{budget}_i(u, v) + \text{budget}_i(v, u)) \), we have that \( w_{i+1}(v) = w_i(v) - \sum_{e : v \in e} \delta_i(e) \geq 0 \). This gives that \( w(v) = \sum_{j=0}^{j-1} (w_i(v) - w_{i+1}(v)) + w_j(v) = \sum_{j=0}^{j-1} \sum_{e : v \in e} \delta_i(e) + w_j(v) \geq 0 \), and hence \( w(v) - \sum_{e : v \in e} \delta(e) = w_j(v) \geq 0 \).

This proves that \( \delta \) is \( G \)-valid, which gives that for \( C = \{ v \in V \mid v \text{ outputs InCover} \} \) it holds that \( \sum_{v \in C} w(v) \leq (2 + \epsilon)OPT \), where \( OPT = \sum_{v \in S_{OPT}} w(v) \) for some optimal vertex cover \( S_{OPT} \), by Theorem 5.1. This is because a vertex \( v \) outputs InCover at the end of iteration \( i = j - 1 \) if and only if \( w_j(v) \leq \epsilon w_0(v) \). It remains to show that \( C \) is a vertex cover. To see why, consider an edge \( e = \{v, u\} \in E \). We claim that if \( u, v \) have both returned by the end of iteration \( i \), then at least one of them is in \( C \). This is because otherwise \( d_{i+1}(v), d_{i+1}(u) \geq 1 \), which implies that both have not returned yet. This completes the proof that \( C \) is indeed a \((2 + \epsilon)\)-approximation for MWVC.

It remains to bound the number of rounds. We do so in the following lemma, in which we show that in each iteration either enough weight is reduced or enough neighbors enter the vertex cover.

**Lemma 5.2.2.** In Algorithm 4, each vertex \( v \) with degree \( d(v) \) completes in \( O(1/\epsilon) \) rounds if \( d(v) \leq 16 \), and in \( O(\log d(v)/\epsilon \log \log d(v)) \) rounds otherwise.

**Proof.** Let \( K_\epsilon > 1 \) be a parameter to be chosen later. Let \( i \) be an iteration at the beginning of which a vertex \( v \in V \) has not yet returned. We claim that either \( d_{i+1}(v) \leq d_i(v)/K_\epsilon \) or
$w_{i+1}(v) \leq w_i(v) - \epsilon' w_0(v)/K_v$. To see why, suppose $d_{i+1}(v) > d_i(v)/K_v$. This means that for at least $\lceil d_i(v)/K_v \rceil$ vertices $u \in N_i(v)$, it holds that $budget_i(u, v) = request_i(v, u)$, and hence

$$w_{i+1}(v) \leq w_i(v) - \left\lceil \frac{d_i(v)}{K_v} \right\rceil \cdot \frac{\text{vault}(v)}{d_i(v)}$$

$$\leq w_i(v) - \frac{d_i(v)}{K_v} \cdot \frac{\epsilon' w_0(v)}{d_i(v)}$$

$$\leq w_i(v) - \epsilon' w_0(v)/K_v.$$

Next, we claim that $v$ returns after at most $K_v/\epsilon' + \log d(v)/\log K_v$ iterations of the algorithm. This is because at most $\log_{K_v} d(v) = \log d(v)/\log K_v$ of the iterations $i$ can be such that $d_{i+1}(v) \leq d_i(v)/K_v$ (since $v$ returns when $d_i(v) = 0$), and at most $K_v/\epsilon'$ iterations $i$ can be such that $w_{i+1}(v) \leq w_i(v) - \epsilon' w_0(v)/K_v$ (since $v$ returns when $w_i(v) \leq \epsilon' w_0(v)$).

Finally, we set $K_v$ as follows. If $d(v) \leq 16$ we set $K_v = d(v) + 1$. This guarantees $K_v > 1$ (an isolated vertex simply outputs NotInCover) and gives $O(1/\epsilon)$ rounds for $v$ to complete.

Otherwise, we set $K_v = \log d(v)/\log \log d(v)$. Since $d(v) > 16$, it holds that $K_v$ is well defined (as $\log \log d(v) > 1$) and that $K_v > 1$. It also holds that $\log K_v > 1$ which is used in what follows. This gives that vertex $v$ returns after at most $j$ iterations, where

$$j \leq \frac{K_v/\epsilon'}{\log d(v)/\log K_v} = \frac{\epsilon' \log \log d(v)}{\log K_v} + \log d(v)/\log K_v$$

$$= \frac{\log d(v)}{\epsilon' \log \log d(v)} + \frac{\log d(v)}{\log (\log d(v)/\log \log d(v))}$$

$$\leq O \left( \frac{\log d(v)}{\epsilon \log \log d(v)} \right),$$

where the last inequality follows because $\epsilon' = \epsilon/(2 + \epsilon)$ (and since $\epsilon$ is at most $O(1)$ and so $\epsilon' = \Theta(\epsilon)$) and $\log \log d(v)$ dominates $\log \log \log d(v)$, completing the proof.

Theorem 5.2 follows directly from Lemmas 5.2.1 and 5.2.2.

5.3 Adaptation to the CONGEST model

Our algorithm is described for the LOCAL model, but can be easily adapted to the CONGEST model in which the message size is limited to $O(\log n)$ bits, provided that the initial weights of the vertices and the ratio between the maximal and minimal weights can be expressed by $O(\log n)$ bits. In order to accommodate $O(\log n)$-bit messages, we slightly modify the messages that are sent as follows. First, in an initial round, each vertex $v$ sends $w_0(v)$ to all of its neighbors. Then, instead of sending $request_i(v, u)$ to neighbor $u$ in some iteration $i$, vertex $v$ only needs to send $d_i(v)$ to its neighbor $u$ and $u$ can locally compute $request_i(v, u) = \text{vault}(v)/d_i(v)$ since all vertices know the value of $\epsilon$ as part of their algorithm.

Second, we need to handle the messages of type $budget_i(v, u)$. In general, this amount can be
an arbitrary fraction which might not fit in $O(\log n)$ bits. However, we notice that we can avoid sending this explicit amount. To do this, we slightly modify $\text{vault}(v)$ to be $\epsilon w_0(v)/2$. Then, upon receiving a request$_i(u,v)$ message, if budget$_i(v,u) = \text{request}_i(u,v)$ then vertex $v$ replies with a predefined message accept, and otherwise, $v$ responds with the maximal integer $t$ such that $t\epsilon w_0(v)/2 \leq \text{budget}_i(v,u)$. The amount $t\epsilon w_0(v)/2$ can be locally computed by $u$, and $u$ can infer that $v$ returns InCover. This is because the remainder of weight in vertex $v$ will be another value of at most $\epsilon w_0(v)/2$ on top of the at most $\epsilon w_0(v)/2$ value which might remain in $\text{vault}(v)$, summing to no more than $\epsilon w_0(v)$, as needed.

### 5.4 Discussion of [KMW10]

The main result of [KMW10] is the following:

**Theorem 9 from [KMW10]** For every constant $\epsilon > 0$, there are graphs $G$, such that in $k$ communication rounds, every distributed algorithm for the minimum vertex cover problem on $G$ has approximation ratios at least

$$\Omega\left(n^{1/4 - \epsilon/k^2}\right) \text{ and } \Omega\left(\Delta^{1/\epsilon}\right),$$

where $n$ and $\Delta$ denote the number of nodes and the highest degree in $G$, respectively.

The argument in [KMW10] is that in order for the above approximation factors to be constant, the number of rounds, $k$, has to be $\Omega(\sqrt{\log n})$ and $\Omega(\log \Delta)$, respectively.

However, we argue that the above lower bounds only hold under the conditions that $k = O((\log n)^{1/3})$ and $k = O(\sqrt{\log \Delta})$, respectively. This means that they cannot be applied to $k = \Theta(\sqrt{\log n})$ or $k = \Theta(\log \Delta)$, and therefore do not imply the claimed bounds for constant approximation factors.

To justify our claim, we elaborate upon the proof of the theorem. Previous lemmas in the paper\footnote{We refer the reader to [KMW10] for exact details.} show that the approximation factor of any $k$-round algorithm is $\Omega(\delta)$, where $\delta$ satisfies the following two constraints\footnote{We use the notation $\delta$ as this is the notation in [KMW10]. Notice that it is unrelated to the function $\delta$ that we use in our framework in previous sections of this paper.}.

First, it holds that $n \leq 2^k/4k^2\delta^k$ and second, it holds that $\Delta = 2^k/2k^2\delta^{k+1}$.

The first constraint implies that

$$\delta \geq \frac{n^{1/4k^2}}{2(2k^3+4k)/4k^2} = \frac{n^{1/4k^2}}{n(2k^3+4k)/4k^2 \log n} = n^{1/4k^2 - (2k^3+4k)/4k^2 \log n}.$$ 

Hence, in order to deduce that $\delta = \Omega(n^{1/4 - \epsilon/k^2})$, it needs to hold that $(2k^3 + 4k)/4k^2 \log n \leq \epsilon/k^2$. However, for this to happen, it must be that $2k^3 + 4k \leq 4\epsilon \log n$, and in particular $k$ has to be within $O((\log n)^{1/3})$. 

\[\text{Technion - Computer Science Department - Ph.D. Thesis PHD-2017-08 - 2017}\]
The second constraint implies that

$$\delta = \frac{\Delta^{1/(k+1)}}{\Delta^{k/2}} = \frac{\Delta^{1/(k+1)}}{\Delta^{k/2} \log \Delta} = \Delta^{1/(k+1) - k/2 \log \Delta}.$$

Hence, in order to deduce that $\delta = \Omega(\Delta^{1/(k+1)})$, it needs to hold that $k/2 \log \Delta \leq \epsilon/(k+1)$. However, for this to happen, it must be that $k/(k+1) \leq 2\epsilon \log \Delta$, and in particular $k$ has to be within $O(\sqrt{\log \Delta})$.

We emphasize again that this last step in the proof of the lower bound is different in the previous version [KMW04], and hence we do not suggest that there is a flaw in [KMW04].

5.5 A tighter bound on the number of communication rounds

In the previous section, we set $K_v = \log d(v)/\log \log d(v)$ to get a bound of $O(\log d(v)/(\epsilon' \log \log d(v)))$ communication rounds. One may notice that plugging $K_v = \epsilon' \log d(v)/\log \log d(v)$ yields the much better bound of $O(\log d(v)/(\log \log d(v) - \log \epsilon'))$. However, for small values of $\epsilon'$ this expression becomes undefined or negative.

The reason for this is that for the proof to make sense, we need the condition $K_v > 1$ to hold. If we set $K_v = \epsilon' \log d(v)/log \log d(v)$, the condition does not hold for small values of $\epsilon'$. We can still get a better dependence on $\epsilon'$ with a slightly more careful analysis.

We note that if $\epsilon' \leq \log \log d(v)/\log d(v)$, then $K_v > 1$ cannot hold for $K_v = \epsilon' \log d(v)/\log \log d(v)$. Instead, for such values of $\epsilon'$, we set $K_v = 2$, which yields a bound of $O(1/\epsilon' + \log d(v))$. If $\epsilon' > \log \log d(v)/\log d(v)$ we still get the previous bound of $O(\log d(v)/(\log \log d(v) - \log \epsilon'))$ rounds, which for these values of $\epsilon$ is $O(\log d(v)/(\log \log d(v)))$. 

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-08 - 2017
Chapter 6

Approximating weighted maximum matching and weighted maximum independent set in the distributed setting

In this chapter we present a simple distributed \( \Delta \)-approximation algorithm for maximum weight independent set (MaxIS) in the CONGEST model which completes in \( O(\text{MIS}(G) \cdot \log W) \) rounds, where \( \Delta \) is the maximum degree, \( \text{MIS}(G) \) is the number of rounds needed to compute a maximal independent set (MIS) on \( G \), and \( W \) is the maximum weight of a node. Plugging in the best known algorithm for MIS gives a randomized solution in \( O(\log n \log W) \) rounds, where \( n \) is the number of nodes. We also present a deterministic \( O(\Delta + \log^* n) \)-round algorithm based on coloring.

We then show how to use our MaxIS approximation algorithms to compute a 2-approximation for maximum weight matching without incurring any additional round penalty in the CONGEST model. We use a known reduction for simulating algorithms on the line graph while incurring congestion, but we show our algorithm is part of a broad family of local aggregation algorithms for which we describe a mechanism that allows the simulation to run in the CONGEST model without an additional overhead.

Next, we show that for maximum weight matching, relaxing the approximation factor to \((2 + \varepsilon)\) allows us to devise a distributed algorithm requiring \( O\left(\frac{\log \Delta}{\log \log \Delta}\right) \) rounds for any constant \( \varepsilon > 0 \). For the unweighted case, we can even obtain a \((1 + \varepsilon)\)-approximation in this number of rounds. These algorithms are the first to achieve the provably optimal round complexity with respect to dependency on \( \Delta \).

6.1 MaxIS approximation

We begin, in Subsection 6.1.1, by showing the idea behind the use of local ratio (presented in Section 3.1) for approximating MaxIS. This is done by presenting a sequential meta-algorithm and analyzing its correctness. Then, in Subsection 6.1.2, we show how to implement this algorithm.
in the CONGEST model, and prove the claimed round complexity.

6.1.1 Sequential MaxIS approximation via local ratio

Here we provide a sequential $\Delta$-approximation meta-algorithm to be used as the base for our distributed algorithm. The correctness of the algorithm is proved using the local ratio technique for maximization problems [BBF+01].

We assume a given weighted graph $G = (V, w, E)$, where $w : V \to \mathbb{R}_+$ is an assignment of weights for the nodes and the degree of each node is bounded by $\Delta$. A simple $\Delta$-approximation local ratio algorithm exists for the problem [BBFR04]. We rely on the local ratio theorem for maximization problems Theorem 3.1 in our proof. We restate the theorem for convenience:

**Theorem 3.1** Let $C$ be a set of feasibility constraints on vectors in $\mathbb{R}^m$. Let $w, w_s, w_r \in \mathbb{R}^m$ be vectors such that $w = w_s + w_r$. Let $x \in \mathbb{R}^n$ be a feasible solution (with respect to $C$) that is a $p$-approximation with respect to $w_s$ and with respect to $w_r$. Then $x$ is a $p$-approximation with respect to $w$ as well.

In our case the vector $w$ is the weight vector representing the weight function of $G(V, E, w)$, $x$ is a binary vector indicating which nodes are chosen to the solution and the set of constraints $C$, is the set of independence constraints. We call the graph with weight vector $w_s$ the *subtracted graph* and the graph with weight vector $w_r$ the *residual graph*.

As standard practice with the local ratio technique, the splitting of the weight vector into $w_s, w_r$ is done such that any $p$-approximate solution to the subtracted graph can be easily transformed into an $p$-approximate solution to the residual graph, while keeping it an $p$-approximate solution for the subtracted graph. This allows us to apply weight reductions iteratively, solving each subproblem while maintaining the constraints. It is important to note that the theorem also holds if the weights in the subtracted graph take negative values.

For the specific problem of MaxIS, we note that picking some node $v \in V$ and reducing the weight of $v$ from every $u \in N(v)$ splits the weight vector $w$ into two vectors, $w_s$ and $w_r$. Where $w_r(v) = w(v)$ for every $u \in N(v)$ and zero for every other node, and $w_s = w - w_r$. Note that any $\Delta$-approximate solution for the subtracted graph can be easily turned into a $\Delta$-approximate solution for the residual graph. This is done by making sure that at least some $u \in N(v)$ is in the solution: If this is not the case, we can always add one $u \in N(v)$ to the solution without violating the independence constraints. This only increases the value of our solution, making it $\Delta$-approximate for both the residual and the subtracted graphs.

The above solution is sequential by nature. Implementing it directly in the distributed setting will require $O(n)$ rounds. We notice that if two nodes are in different neighborhoods of the graph then this process can be performed by both of them simultaneously without affecting each other. This observation forms the base for our distributed implementation.

We expand this idea by taking any independent set $U \subseteq V$ and for every $v \in U$ reducing the weight of $v$ from every $u \in N(v)$ in parallel. Next, solve the problem for the subtracted graph. If for some $v \in U$, every $u \in N(v)$ is not in the solution for the subtracted graph, we add $v$ to the solution for the subtracted graph. This yields a $\Delta$-approximate solution for the problem.
For the sake of simplicity let \( V = [n] \). Let \( w_r \) be the weight vector of the residual graph after performing weight reductions as described above for some independent set \( U \subseteq V \). By definition \( w_r[v] = \sum_{u \in U \cap N(v)} w[u] \). The weight of the subtracted graph is given by \( w_s = w - w_r \). Let \( x \in \{0,1\}^n \) be some \( \Delta \)-approximate solution for the subtracted graph. The cost of the solution \( x \) is \( \sum_v w[v] x[v] \). Let \( x' \in \{0,1\}^n \) be defined as follows:

\[
x'[u] = \begin{cases} 
1 & u \in U \land \forall v \in N(u), x[v] = 0 \\
x[u] & \text{otherwise}
\end{cases}
\]

(6.1)

We prove the following lemma.

**Lemma 6.1.1.** \( x' \) is a \( \Delta \)-approximate solution for both the subtracted graph and the residual graph.

**Proof.** We note that \( w_r[u] = w[u] \) for every \( u \in U \). Thus, \( w_s[u] = 0 \) for every \( u \in U \). We do not incur any additional cost for the subtracted graph because \( x' \) is created by adding nodes from \( U \) to \( x \). Because \( x \) is \( \Delta \)-approximate for the subtracted graph, so is \( x' \).

For the residual graph, only nodes in \( \cup_{u \in U} N(u) \) have non zero weights. Let \( x^* \in \{0,1\}^n \) be an optimal solution for the residual graph. We can bound from above the weight of \( x^* \) by summing over the weights of \( N(u) \) for every \( u \in U \) where \( x^*[u] = 1 \), taking into account that for any neighborhood \( N(u) \), at most \(|N(v)| - 1\) nodes can be selected to a solution due to the independence constraints. We get the following upper bound for the weight \( x^* \):

\[
\sum_{v \in V} w_r[v] x^*[v] = \sum_{v \in V} \sum_{u \in U \cap N(v)} w_r[u] x^*[u] = \sum_{u \in U} \sum_{v \in N(u)} w_r[u] x^*[u] \\
\leq \sum_{u \in U} w_r[u] \cdot (|N(u)| - 1) = \sum_{u \in U} w_r[u] \cdot \deg(u) \leq \Delta \sum_{u \in U} w_r[u].
\]

On the other hand, \( x' \) is selected such that for each \( u \in U \) at least one \( v \in N(u) \) is in \( x' \) for any \( u \in U \). Thus, \( x' \cdot w_r = \sum_{u \in U} \sum_{v \in N(u)} w_r[u] \cdot x'[u] \geq \sum_{u \in U} w_r[u] \), which means that \( x' \) is at least a \( \Delta \)-approximation for \( x^* \) on the residual graph, and the proof is complete. \( \square \)

**Overview of Algorithm 5:** Using Lemma 6.1.1 we construct a meta-algorithm that at each iteration picks an independent set \( U \subseteq V \), reduces the weights of the elements in \( U \) from their neighborhood and calls itself recursively with the reduced weights. This implicitly splits the graph into the subtracted graph and the residual graph. A recursive call returns a \( \Delta \)-approximate solution for the subtracted graph which is turned into a \( \Delta \)-approximate solution for both graphs by adding all nodes in the independent set that do not have neighbors in the returned solution. According to the local ratio theorem the final solution is a \( \Delta \)-approximation. At this point we are only interested in the correctness of the algorithm, thus it does not matter how the set \( U \) is picked.

The recursive step of Algorithm 5 returns a \( \Delta \)-approximate solution for the subtracted graph which is then turned into a \( \Delta \)-approximate solution for the residual graph. Correctness follows from Lemma 6.1.1 combined with a simple inductive argument. In the next section we implement this algorithm in a distributed setting.
Algorithm 5: SeqLR(V, E, w) - Sequential LR algorithm for maximum independent set

1. if V = ∅ then
   2. Return ∅
3. foreach v ∈ V do
   4. if w(v) ≤ 0 then
      5. V = V \ {v}
      6. E = E \ {(v, u) | u ∈ V}
7. Let U ⊆ V be an independent set
8. Let ws = w
9. foreach u ∈ U do
   10. foreach v ∈ N(u) do
       11. ws(v) = w(v) − w(u)
12. R = SeqLR(V, E, ws)
13. U = U \ ∪v∈R N(v)
14. Return R ∪ U

6.1.2 Distributed MaxIS approximation via local ratio

In this subsection we implement Algorithm 5 in the distributed setting. We present an algorithm which iteratively finds independent sets and finishes after log W iterations. This yields a Δ-approximation in O(MIS(G) log W) rounds, where MIS(G) is the running time of a black-box MIS algorithm used. The algorithm that wraps the MIS procedure is deterministic, while the MIS procedure may be random. If the MIS procedure is random and finishes after T rounds w.h.p then our algorithm requires O(T log W) rounds w.h.p. This holds for the CONGEST model.

From now on we assume that all node weights are integers in [W]. The sequential meta algorithm can be implemented in a distributed manner, by having each node in the set perform weight reductions independently of other nodes. The key questions left open in the transition to the distributed setting is how to select our independent set at each iteration and how many rounds must we pay. Iteratively running the MIS procedure and performing weight reductions does not guarantee anything with regard to the number of nodes removed at each iteration or to the amount of weight reduced.

Overview of the distributed algorithm. The pseudocode is given in Algorithm 6. The algorithm works by dividing the nodes into layers according to their weights. The i-th layer is given by \( L_i = \{ v \mid 2^{i-1} < w(v) \leq 2^i \} \). During the algorithm each node keeps track of the weights (and layers) of neighboring nodes and updates are sent regarding weight changes and node removals. We divide the algorithm into two stages: the removal stage and the addition stage.

In the removal stage we find an independent set in the graph and perform weight reductions exactly as in the sequential meta algorithm. When finding the MIS, nodes in higher layers are prioritized over nodes in lower layers. A node cannot start running the MIS algorithm as long as it has a neighbor in a higher level. The most important thing to note here is that nodes in the topmost level never need to wait. A node who is selected to the MIS during the removal stage is
Algorithm 6: A distributed $\Delta$-approximation for weighted MaxIS, code for node $v$

// $w(v)$ is the initial weight of $v$, $w_v(v)$ changes during the run of the algorithm

1. $w_v(v) = w(v)$
2. $\ell_v(v) = \lceil \log w_v(v) \rceil$
3. status = waiting
4. while true do
5.     foreach reduce($x$) received from $u \in N(v)$ do
6.         $w_v(v) = w_v(v) - x$
7.         $N(v) = N(v) \setminus \{u\}$
8.         if $w_v(v) \leq 0$ then
9.             Send removed($v$) to all neighbors
10.            return NotInIS
11.     end
12.     foreach removed($u$) received from $N(v)$ do
13.         $N(v) = N(v) \setminus \{u\}$
14.         $\ell_v(v) = \lceil \log w_v(v) \rceil$
15.         Send weightUpdate($v, w_v(v)$) to all neighbors
16.     foreach weightUpdate($u, w'_v$) received from $N(v)$ do
17.         $w_v(u) = w'_v$
18.         $\ell_v(u) = \lceil \log w_v(u) \rceil$
19.     end
20.     if status = waiting then
21.         if $\forall u \in N(v), \ell_v(u) \leq \ell_v(v)$ then
22.             status($v$) = ready
23.             while $\exists u \in N(v), \ell_v(u) = \ell_v(v)$ and status($u$) $\neq$ ready do
24.                 Wait
25.             $v$ starts running MIS algorithm
26.         end
27.         if $v$ in MIS then
28.             Send reduce($w_v(v)$) to all neighbors
29.             $w_v(v) = 0$
30.             status = candidate
31.         else
32.             status = waiting
33.         end
34.     else if status = candidate then
35.         if $N(v) = \emptyset$ then
36.             Send addedToIS($v$) to all neighbors
37.             Return InIS
38.         end
39.     end
40.     endif
41. end
a candidate node. A node whose weight becomes zero or negative without being added to the MIS is said to be a removed node. Removed nodes output \texttt{NotInIS} and finish, while candidate nodes continue to the addition stage. Both candidate and removed nodes are deleted from the neighborhood of their neighbors.

In the addition stage, a candidate node \( v \) remains only with neighbors with higher weights. We say these nodes have precedence over the node \( v \). A node \( v \) may add itself to the solution only if it has no neighboring nodes which have precedence over it. After a node is added to the solution, all of its neighbors become removed. This corresponds line 13 in the sequential meta algorithm.

The correctness of the distributed algorithm follows directly from the correctness of the sequential meta algorithm. We are only left to bound the number of rounds. Let us consider the communication cost of the removal stage. We define the topmost layer to be \( L_{\text{top}} = L_j \) where \( j = \arg \max_i L_i \neq \emptyset \). Note that nodes in \( L_{\text{top}} \) never wait to run the MIS, and that after the MIS finishes for \( L_{\text{top}} \), the weight of every \( v \in L_{\text{top}} \) is reduced by at least a factor of two (proved in Lemma 6.1.2), emptying that layer. This can repeat at most \( \log W \) times.

We assume a black-box MIS algorithm that finishes after \( \text{MIS}(G) \) rounds with probability at least \( 1 - p \).

**Lemma 6.1.2.** With probability at least \( 1 - p \), \( L_{\text{top}} = \emptyset \) after \( \text{MIS}(G) \) rounds.

**Proof.** Let \( G' \) be the graph induced by \( L_{\text{top}} \). By the code, nodes in \( L_{\text{top}} \) need not wait to run an MIS algorithm and as long as a node is not in \( L_{\text{top}} \) it does not participate in an MIS algorithm. With probability at least \( 1 - p \) an MIS is selected for \( G' \) after \( \text{MIS}(G) \) rounds. All nodes selected to the MIS have their weights reduced to zero. Every other node \( v \) has at least one neighbor in the MIS, whose weight, by our definition of layers, is at least half of the weight of \( v \). Thus the weight of every node \( v \in L_{\text{top}} \) is halved, emptying the layer. \( \square \)

We can now prove the main theorem for this subsection.

**Theorem 6.1.** The distributed MaxIS approximation algorithm (Algorithm 6) finishes after at most \( \text{MIS}(G) \cdot \log W \) rounds with probability at least \( 1 - p \log W \) in the \textsc{CONGEST} model.

**Proof.** Applying a union bound over all layers, gives that all layers are empty after at most \( \text{MIS}(G) \cdot \log W \) iterations with probability at least \( 1 - p \log W \), by Lemma 6.1.2. We require \( p = o(1/\log W) \). This bounds the communication cost for the removal stage.

Denote by \( C_i \) the set of candidate nodes from level \( L_i \). These nodes are at level \( L_i \) when they are set to be candidate nodes. Nodes in \( C_i \) wait for neighbors with higher precedence to decide whether they enter the solution. We note that nodes in \( C_0 \) do not have any neighbors with higher precedence. After nodes in \( C_0 \) have decided, the nodes in \( C_1 \) do not have to wait and so on. Thus, all candidate nodes make a decision after at most \( \log W \) rounds. This bounds the communication cost for the addition stage. \( \square \)

### 6.1.3 Deterministic coloring-based approximation algorithm

In this subsection we present a simple coloring-based \( \Delta \)-approximation algorithm for MaxIS. The advantage of this approach is that we have no dependence on \( W \), yielding a deterministic algorithm running in \( O(\Delta + \log^* n) \) rounds in the \textsc{CONGEST} model.
In the algorithm (pseudocode in Algorithm 7), instead of partitioning the nodes based on weights, they are partitioned based on colors, where colors with larger index have priority. Nodes perform weight reductions if their color is a local maxima. As in the previous section we have two stages: removal and addition, and three types of node states: removed, candidate and precedent. After one iteration all nodes of the top color are either candidate or removed nodes. Thus after \( \Delta + 1 \) iterations all nodes are either candidate or removed nodes. Thus, the removal stage finishes in \( O(\Delta) \) rounds.

As in Algorithm 6, after the removal stage all candidate nodes only have nodes who have precedence over them as their neighbors. A node adds itself to the independent set if it has no neighbors with precedence over it, in which case all of its neighbors become removed. We again note that candidate nodes of the smallest color have no neighbors and are added to the solution. We conclude that the removal stages also finishes in \( O(\Delta) \) rounds.

Algorithm 7: Coloring-based distributed \( \Delta \)-approximation for weighted MaxIS, code for node \( v \)

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Run a ( \Delta + 1 ) coloring algorithm</td>
</tr>
<tr>
<td>2</td>
<td>Let ( c : v \to [\Delta + 1] ) be a coloring for the nodes</td>
</tr>
<tr>
<td>3</td>
<td>( w(v) = w_v )</td>
</tr>
<tr>
<td>4</td>
<td><strong>foreach</strong> ( w'(w) ) received from ( u \in N(v) ) <strong>do</strong></td>
</tr>
<tr>
<td>5</td>
<td>( w(v) = w(v) - w' )</td>
</tr>
<tr>
<td>6</td>
<td>( N(v) = N(v) \setminus {u} )</td>
</tr>
<tr>
<td>7</td>
<td><strong>if</strong> ( w(v) \leq 0 )** then**</td>
</tr>
<tr>
<td>8</td>
<td>Send <strong>removed</strong>(( v )) to all neighbors</td>
</tr>
<tr>
<td>9</td>
<td>return <strong>NotInIS</strong></td>
</tr>
<tr>
<td>10</td>
<td><strong>foreach</strong> <strong>removed</strong>(( u )) received from ( N(v) ) <strong>do</strong></td>
</tr>
<tr>
<td>11</td>
<td>( N(v) = N(v) \setminus {u} )</td>
</tr>
<tr>
<td>12</td>
<td><strong>if</strong> ( N(v) = \emptyset )** then**</td>
</tr>
<tr>
<td>13</td>
<td>Send <strong>addedToIS</strong>(( v )) to all neighbors</td>
</tr>
<tr>
<td>14</td>
<td>Return <strong>InIS</strong></td>
</tr>
<tr>
<td>15</td>
<td><strong>if</strong> <strong>addedToIS</strong>(( u )) received from ( N(v) )** then**</td>
</tr>
<tr>
<td>16</td>
<td>Send <strong>removed</strong>(( v )) to all neighbors</td>
</tr>
<tr>
<td>17</td>
<td>Return <strong>NotInIS</strong></td>
</tr>
<tr>
<td>18</td>
<td><strong>if</strong> ( \forall u \in N(v) \setminus {v} ) it holds that ( c(v) &gt; c(u) )** then**</td>
</tr>
<tr>
<td>19</td>
<td><strong>foreach</strong> ( u \in N(v) ) <strong>do</strong></td>
</tr>
<tr>
<td>20</td>
<td>Send <strong>reduce</strong>(( w(v) ))</td>
</tr>
<tr>
<td>21</td>
<td>( w(v) = 0 )</td>
</tr>
</tbody>
</table>

Algorithm 7 is a distributed implementation of Algorithm 5, where the independent set is selected via its color at each iteration. The correctness of the algorithm is derived from the correctness of Algorithm 5. The number of rounds of Algorithm 7 is \( O(\Delta + \log^* n) \) as it is known that deterministic distributed coloring can be achieved in less than \( O(\Delta + \log^* n) \) rounds [BEK14, Bar15].\(^1\) The \( \log^* n \) factor cannot be improved upon due to a lower bound by Linial [Lin87].

\(^1\)[FHK15] gives a faster coloring but is in \textsc{local} and in any case we need to pay for the number of colors as well.
6.2 Distributed 2-approximation for maximum weighted matching

From the results in the previous section we can now derive local 2-approximation algorithms for maximum matching. Let $G$ be a graph with weighted nodes, and let $L(G)$ be the line graph of $G$. Where $L(G)$ is created from $G$ by replacing each edge in $G$ with a node in $L(G)$. Two nodes are connected in $L(G)$ if their matching edges in $G$ share a node. It is well known that a maximum independent set in $L(G)$ corresponds to a maximum matching in $G$. An algorithm is executed on the line graph by assigning each edge in $G$ to have its computation simulated by one of its endpoints [Kuh05]. We show that running our local ratio based approximation algorithms on $L(G)$ yields a 2-approximate maximum matching in $G$. The main challenge is how to handle congestion, since nodes in $G$ may need to simulate many edges, thus may have to send many messages in a naive simulation.

Recall Algorithm 5, the sequential $\Delta$-approximation meta-algorithm. The approximation factor was proved in Lemma 6.1.1 to be $\Delta$. Specifically, the following equation provided an upper bound for the weight of an optimal solution $x^*$.

$$\sum_{u \in U} \sum_{v \in N(u)} w[u]x^*[u] \leq \sum_{u \in U} w[u] \cdot (|N(u)| - 1) = \sum_{u \in U} w[u] \cdot \deg(u) \leq \Delta \sum_{u \in U} w[u].$$

The above bound uses the fact that for any node $u \in U$, at most $|N(u)| - 1$ nodes in $N(v)$ can be selected for the solution due to independence constraints. But in $L(G)$ the largest independent set in the neighborhood of some node in $L(G)$ is at most 2, yielding the following upper bound:

$$\sum_{u \in U} \sum_{v \in N(u)} w[u]x^*[u] \leq 2 \sum_{u \in U} w[u].$$

We conclude that the algorithms presented in the previous sections provide 2-approximation for maximum matching when executed on $G(L)$.

As for the communication complexity, the line graph has at most $n\Delta$ nodes and degree bounded by $2\Delta - 2$. Thus, simulation of our algorithms on $L(G)$ in the LOCAL model does not incur any additional asymptotic cost. However, in a naive implementation in the CONGEST model we pay an $O(\Delta)$ multiplicative penalty due to congestion. This added congestion can be avoided with some careful modifications to our algorithms, as explained next.

For $e = (v, u)$ that is simulated by $v$, we call $v$ its primary node and $u$ its secondary node. We define a family of algorithms called local aggregation algorithms and show that these algorithms can be augmented to not incur any additional communication penalty when executed on the line graph relative to their performance on the original graph in the CONGEST model. We begin with some definitions.

**Definition 6.2.1.** We say that $f : \Sigma^n \rightarrow \Sigma$ is order invariant, if for any set of inputs $\{x_i\}_{i=1}^n$, and any permutation $\pi$, it holds that $f(x_1, ..., x_n) = f(x_{\pi(1)}, ..., x_{\pi(n)})$.

For the sake of simplicity, if $f$ is order invariant we write $f(x_1, ..., x_n)$ as $f(\{x_i\})$. We may also give a partial parameter set to our function, in which case we assume all remaining
inputs to the function are the empty character $\epsilon \in \Sigma$. Formally, for $X' = \{x_1, \ldots, x_k, \epsilon, \ldots, \epsilon\}$,

\[ f(X') = f(x_1, \ldots, x_k, \epsilon, \ldots, \epsilon). \]

Definition 6.2.2. We say that a function $f : \Sigma^n \to \Sigma$ is an aggregate function if it is order invariant and there exists a function $\phi : \Sigma^2 \to \Sigma$ such that for any set of inputs $X = \{x_1\}^k$, and any disjoint partition of the inputs into $X_1, X_2$ it holds that $f(X) = \phi(f(X_1), f(X_2))$. The function $\phi$ is called the joining function.

Observation 6.2.3. It is easy to see that Boolean "and" and "or" functions are aggregate functions.

Let $Alg$ be some algorithm for the CONGEST model. Let us denote by $D_{v,i}$ the local data stored by node $v$ during the run of the algorithm at round $i$. Let $D_{N(v),i} = \{D_{u,i} \mid u \in N(v)\}$ be the data of $v$’s immediate neighborhood.

Definition 6.2.4. We call $Alg$ a local aggregation algorithm if it only accesses $D_{N(v),i}$ using aggregate functions where $|\Sigma| = O(\log n)$ and $|D_{v,i}| = O(\log n)$ for every $v \in V, i \in [t]$.

For example, Luby’s MIS algorithm [Lub86] is a local aggregation algorithm. At each iteration a node $v$ asks its neighbors if there exists a neighbor whose random value is larger then that of $v$. It also asks if there exists a neighbor who entered the MIS. The data of a node at iteration $i$ is only its random number and its status. The data of the neighbors of $v$ is aggregated via the "or" function which is an aggregate function.

Theorem 6.2. If $Alg$ is a local aggregation algorithm running in the CONGEST model in $O(t)$ rounds, it can be executed on the line graph in $O(t)$ rounds.

Proof. $Alg$ is executed on the primary node, and we maintain the invariant that $D_{v,i}$ is always present in both the primary and secondary nodes. Every time $Alg$ needs to execute a function $f$, both the primary and secondary nodes already have the data of all of their neighbors. Each node calculates $f$ on the data of its neighbors, the secondary node sends this calculation to the primary which in turn executes the joining function yielding the desired result. Afterwards the new node data is sent to the secondary node.

No communication is needed to access the data of the neighbors, as a neighbor of $\epsilon$ must share a node with it, which contains its data. There is no congestion when sending the value of $f$ or the new data to the secondary node. Thus, the number of rounds is $O(t)$.

Theorem 6.3. Algorithm 6 is a local aggregation algorithm.

Proof. Let us explicitly define $D_{v,i}$ for every $v \in V$. Each node knows its weight, status and degree. Formally, $D_{v,i} = \{w_i(v), status_v, deg(v)\}$. The algorithm uses "and" and "or" Boolean functions, which by Observation 6.2.3, are aggregate functions. Each node also needs to update its weight at each iteration. The weight update function can be written as $f_w(w_v, \{w_u \mid u \in N(v)\}) = w_v - \sum w_u$, which is of course an aggregate function.

This exact same technique can be applied to Algorithm 7, giving the main result for this section.
Theorem 6.4. There exist a randomized 2-approximation algorithm for maximum weighted matching in the CONGEST model running in $O(\text{MIS}(G) \cdot \log W)$ rounds, and a deterministic 2-approximation algorithm for maximum weighted matching in the CONGEST model running in $O(\Delta + \log^* n)$ rounds.

6.3 Faster Approximations of Maximum Matching

In this section, we present $O\left(\frac{\log \Delta}{\log \log \Delta}\right)$-round algorithms for $(2 + \varepsilon)$-approximation of maximum weighted matching and $(1 + \varepsilon)$-approximation of maximum unweighted matching. As stated before, these are the first algorithms to obtain the provably optimal round complexity for matching approximation. Their complexity matches the seminal lower bound of Kuhn, Moscibroda, and Wattenhofer [KMW06] which shows that $\Omega\left(\frac{\log \Delta}{\log \log \Delta}\right)$ rounds are necessary, in fact for any constant approximation.

6.3.1 A fast $(2 + \varepsilon)$-approximation of maximum weighted matching

We first present a simple $O\left(\frac{\log \Delta}{\log \log \Delta}\right)$-round $(2 + \varepsilon)$-approximation for maximum unweighted matching. We then explain how this approximation extends to the weighted setting via known methods.

To get a $(2 + \varepsilon)$-approximation, the idea is to quickly find a large matching and remove its edges from the problem along with the other edges that are incident on them. Once no edge remains (or when only a few remain), we have reached our approximation goal. The key algorithmic component in our approach is an adaptation of the algorithm of Ghaffari [Gha16]. Ghaffari presented an MIS algorithm, which if executed on a graph $H$ with maximum degree $\Delta$ for $O(\log \Delta + 1/\delta)$ rounds, computes an independent set $\text{IS}$ of nodes of $H$, with the following probabilistic near-maximality guarantee: each node of $H$ is either in $\text{IS}$ or has a neighbor in it, with probability at least $1 - \delta$. We will be applying a similar method on the line graph of our original graph, hence choosing a nearly-maximal independent set of edges. However, this running time is not quite fast enough for our target complexity.

We first explain relatively simple changes in the algorithm and its analysis that improve the complexity to $O\left(\frac{\log \Delta}{\log \log \Delta}\right)$, for any constant $\varepsilon > 0$. We then explain how that leads to an $O\left(\frac{\log \Delta}{\log \log \Delta}\right)$-round $(2 + \varepsilon)$-approximation for maximum unweighted matching.

The Modified Nearly-Maximal Independent Set Algorithm In each iteration $t$, each node $v$ has a probability $p_t(v)$ for trying to join the independent set $\text{IS}$. Initially $p_0(v) = 1/K$, for a parameter $K$ to be fixed later. The total sum of the probabilities of neighbors of $v$ is called its effective-degree $d_t(v)$, i.e., $d_t(v) = \sum_{u \in N(v)} p_t(u)$. The probabilities change over time as follows:

$$p_{t+1}(v) = \begin{cases} p_t(v)/K, & \text{if } d_t(v) \geq 2 \\ \min\{Kp_t(v), 1/K\}, & \text{if } d_t(v) < 2. \end{cases}$$
The probabilities are used as follows: In each iteration, node $v$ gets marked with probability $p_t(v)$ and if no neighbor of $v$ is marked, $v$ joins IS and gets removed along with its neighbors.

**Theorem 6.5.** For each node $v$, the probability that by the end of round $\beta(\log \Delta/\log K + K^2 \log 1/\delta) = O(\frac{\log \Delta}{\log \log \Delta})$, for a large enough constant $\beta$, node $v$ is not in IS and does not have a neighbor in IS is at most $\delta$. Furthermore, this holds even if coin tosses outside $N^+_2(v)$ are determined adversarially.

Let us say that a node $u$ is low-degree if $d_t(u) < 2$, and high-degree otherwise. We define two types of golden rounds for a node $v$: (1) rounds in which $d_t(v) < 2$ and $p_t(v) = 1/K$, (2) rounds in which $d_t(v) \geq 1$ and at least $d_t(v)/(2K^2)$ of $d_t(v)$ is contributed by low-degree neighbors.

**Lemma 6.3.1.** By the end of round $\beta(\log \Delta/\log K + K^2 \log 1/\delta)$, either $v$ has joined IS, or has a neighbor in IS, or at least one of its golden round counts reached $\frac{1}{K^2}(\log \Delta/\log K + K^2 \log 1/\delta)$.

**Proof.** Let $T = \beta(\log \Delta/\log K + K^2 \log 1/\delta)$ for a sufficiently large constant $\beta$. We focus only on the first $T$ rounds. Let $g_1$ and $g_2$ respectively be the number of golden rounds of types 1 and 2 for $v$, during this period. We assume that by the end of round $T$, node $v$ is not removed and $g_1 \leq T/13$, and we conclude that $g_2 \geq T/13$.

Let $h$ be the number of rounds during which $d_t(v) \geq 2$. Notice that the changes in $p_t(v)$ are governed by the condition $d_t(v) \geq 2$ and the rounds with $d_t(v) \geq 2$ are exactly the ones in which $p_t(v)$ decreases by a $K$ factor. Since the number of $K$-factor increases in $p_t(v)$ can be at most equal to its number of $K$-factor decreases, there are at least $T-2h$ rounds in which $p_t(v) = 1/K$. Out of these rounds, at most $h$ rounds can have $d_t(v) \geq 2$. Hence, $g_1 \geq T-3h$. The assumption $g_1 \leq T/13$ gives that $h \geq 4T/13$. Let us now consider the changes in the effective-degree $d_t(v)$ of $v$ over time. If $d_t(v) \geq 1$ and this is not a golden round of type-2, then we have

$$d_{t+1}(v) \leq K \frac{1}{2K^2} d_t(v) + \frac{1}{K}(1 - \frac{1}{2K^2})d_t(v) < \frac{3}{2K} d_t(v).$$

There are $g_2$ golden rounds of type-2. Except for these, whenever $d_t(v) \geq 1$, the effective-degree $d_t(v)$ shrinks by at least a $\frac{3}{2K}$ factor. In these exception cases, it increases by at most a $K$ factor. Each of these exception rounds cancels the effect of no more than 2 shrinkage rounds, as $(\frac{3}{2K})^2 \cdot K \ll 1$. Thus, ignoring the total of at most $3g_2$ rounds lost due to type-2 golden rounds and their cancellation effects, every other round with $d_t(v) \geq 2$ pushes the effective-degree down by a $\frac{3}{2K}$ factor. This cannot happen more than $\log \frac{2\Delta}{2K}$ times as that would lead the effective degree to exit the $d_t(v) \geq 2$ region. Hence, the number of rounds in which $d_t(v) \geq 2$ is at most $\frac{\log \Delta}{\log \frac{2\Delta}{2K}} + 3g_2$. That is, $h \leq \frac{\log \Delta}{\log \frac{2\Delta}{2K}} + 3g_2$. Since $h \geq 4T/13$, and because $T = \beta(\log \Delta/\log K + K^2 \log 1/\delta)$ for a sufficiently large constant $\beta$, we get that $g_2 > T/13$. \qed

**Lemma 6.3.2.** In each type-1 golden round, with probability at least $\Theta(1/K)$, $v$ joins the IS. Moreover, in each type-2 golden round, with probability at least $\Theta(1/K^2)$, a neighbor of $v$ joins the IS. Hence, the probability that by the end of round $\beta(\log \Delta/\log K + K^2 \log 1/\delta)$, node $v$ has not joined the IS and does not have a neighbor in it is at most $\delta$. These statements hold even if the coin tosses outside $N^+_2(v)$ are determined adversarially.
Proof. In each golden type-1 round, we have \( d_t(v) < 2 \) and \( p_t(v) = 1/K \). The latter means that node \( v \) gets marked with probability \( 1/K \), and the former means that the probability that none of the neighbors of \( v \) is marked is at least \( \prod_{u \in N_t(v)} (1 - p_t(u)) \geq 4^{-\sum_{u \in N_t(v)} p_t(u)} = 4^{-d_t(v)} \geq 1/16 \). Hence, in each golden type-1 round, node \( v \) joins the IS with probability at least \( 1/(16K) \).

In each golden type-2 rounds, we have \( d_v(t) \geq 1 \) and at least \( d_v(t)/(2K^2) \) of \( d_v(v) \) is contributed by low-degree neighbors. Suppose we examine the set \( L_t(v) \) of low-degree neighbors of \( v \), one by one and check whether they are marked or not. We stop when we reach the first marked node. The probability that we find at least one marked node is at least \( 1 - \prod_{u \in L_t(v)} (1 - p_t(u)) \geq 1 - e^{-\sum_{u \in L_t(v)} p_t(u)} \geq 1 - e^{-d_t(v)/(2K^2)} \geq 1 - e^{-1/2K^2} \geq 1/4K^2 \), given that \( (2K^2) \geq 1 \). Now that we have found the first marked light neighbor \( u \), the probability that no neighbor \( w \) of \( u \) is marked is at least \( \prod_{w \in N_t(u)} (1 - p_t(w)) \geq 4^{-\sum_{w \in N_t(u)} p_t(w)} = 4^{-d_t(u)} \geq 1/16 \). Therefore, overall, the probability that node \( v \) gets removed in a type-2 golden round is at least \( \frac{1}{64K^2} \).

Now notice that these events are independent in different rounds. Hence, the probability that node \( v \) does not get removed after \( \Theta(K^2 \log 1/\delta) \) golden rounds is at most \( (1 - \frac{1}{64K^2})^{\Theta(K^2 \log 1/\delta)} \leq \delta \). Furthermore, in the above arguments, we only relied on the randomness in the nodes that are at most within 2 hops of \( v \). Hence, the guarantee is independent of the randomness outside the 2-neighborhood of \( v \).

Proof of Theorem 6.5. By Lemma 6.3.1, within the first \( \beta(\log \Delta/\log K + K^2 \log 1/\delta) \) round, each node \( v \) is either already removed (by joining or having a neighbor in the IS) or one of its golden round counts reaches at least \( \beta(\log \Delta/\log K + K^2 \log 1/\delta)/13 \). As Lemma 6.3.2 shows, in each golden round, node \( v \) gets removed with probability at least \( \Theta(1/K^2) \). Hence, given a large enough constant \( \beta \), the probability that node \( v \) remains through \( \beta(K^2 \log 1/\delta)/13 \) golden rounds is at most \( \delta \).

Theorem 6.6. There is a distributed algorithm in the CONGEST model that computes a \((2 + \varepsilon)\)-approximation of maximum unweighted matching in \( O(\frac{\log \Delta}{\log \log \Delta}) \) rounds, for any constant \( \varepsilon > 0 \), whp.

Proof. The algorithm executes the nearly-maximal independent set algorithm explained above on the line-graph. This finds a nearly-maximal set of independent edges, i.e., edges which do not share an endpoint, or in other words, a nearly-maximal matching. The fact that the algorithm can be run on the line-graph in the CONGEST model follows from 6.2, since it is easy to see that this is a local aggregation algorithm. The round complexity of \( O(\frac{\log \Delta}{\log \log \Delta}) \) follows from the \( O(\log \Delta/\log K + K^2 \log 1/\delta) \) bound of 6.5, by setting \( K = \Theta(\log^{0.1} \Delta) \) and \( \delta = 2^{-\log^{0.7} \Delta} \). Let us now examine the approximation factor. Each edge of the optimal matching has probability at most \( \delta \) of becoming unlucky and not being in our found matching and not having any adjacent edge in it either. These are the edges that remain after all iterations of the nearly-maximal independent set algorithm. Thus, we expect at most \( \delta \ll \varepsilon \) fraction of the edges of the optimal matching to become unlucky. The number also has an exponential concentration around this mean\(^2\). Ignoring these \( \varepsilon |OPT| \) unlucky edges of the optimal matching, among the rest of the

\(^2\)This concentration is due to the fact that the dependencies are local and each edge’s event of being unlucky depends on only at most \( \Delta \) other edges. However, one can obtain a better success probability. See 6.4.1 for an algorithm which provides a stronger concentration, giving a \((2 + \varepsilon)\)-approximation with probability \( 1 - e^{-\Omega(-|OPT|)} \).
edges, each edge of the found matching can be blamed for removing at most 2 edges of the optimal matching. So the found matching is a \((2 + \varepsilon)\)-approximation.

**Extension to the Weighted Case via Methods of Lotker et al.** Above, we explain an \(O\left(\frac{\log \Delta}{\log \log \Delta}\right)\)-round algorithm for \((2 + \varepsilon)\)-approximation of maximum unweighted matching. This can be extended to the weighted case via known methods, while preserving the asymptotic complexity, as follows: First, we sketch a method of Lotker et al.\[LPSR09\] which allows one to turn a \((2 + \varepsilon)\)-approximation for the unweighted case to an \(O(1)\)-approximation for the weighted case. Classify the weights of edges into powers of a large constant \(\beta\), i.e., by defining weight buckets of the form \([\beta^i, \beta^{i+1}]\). In each of these big-buckets, partition the weight range further into \(O(\log(1 + \varepsilon)\beta)\) small-buckets in powers of \(1 + \varepsilon\). Run the following procedure in all big-buckets in parallel: Starting from the edges of the highest weight small-bucket in this big-bucket, find a \((2 + \varepsilon)\)-approximation of the matching in that small-bucket using the unweighted matching algorithm, remove all their incident edges in that big-bucket, and move to the next biggest small-bucket. After \(O(\log_1(1 + \varepsilon)\beta)\) iterations of going through all the small-buckets, for each big-bucket, we have found a matching that is a \(2 + O(\varepsilon)\) approximation of the maximum weight matching among all the edges with weight in this big-bucket. However, altogether, this is not a matching as a node might have a “matching”-edge incident on it in each of the big-buckets. Keep each of these chosen edges only if it has the highest weight among the chosen edges incident on it. Lotker et al.\[LPSR09\] showed that this produces an \(O(1)\)-approximation of the maximum weight matching.

Now, this \(O(1)\)-approximation can be turned into a \((2 + \varepsilon)\)-approximation. Lotker et al.\[LPSP15, Section 4\] present a method that via \(O(1/\varepsilon)\) black-box usages of an \(O(1)\)-approximation Maximum Weight Matching algorithm \(\mathcal{A}\), produces a \((2 + \varepsilon)\)-approximation of the maximum weight matching. We here provide only a brief and intuitive sketch. The method is iterative, each iteration is as follows. Let \(M\) be the current matching. We look only at weighted augmenting paths of \(M\) with length at most 3. We define an auxiliary weight for each unmatched edge \(e\), which is equal to the overall weight-gain that would be obtained by adding \(e\) to the matching and instead erasing the matching \(M\) edges incident on endpoints of \(e\) (if there are any). Note that this auxiliary weight can be computed easily in \(O(1)\) rounds. Then, we use algorithm \(\mathcal{A}\) to find a matching which has an auxiliary weight at most an \(O(1)\) factor smaller than the maximum weight matching, according to the auxiliary weights. Then we augment \(M\) with all these found matching edges, erasing the previously matching edges incident on their endpoints. We are then ready for the next iteration. As Lotker et al. show, after \(O(1/\varepsilon)\) iterations, the matching at hand is a \((2 + \varepsilon)\)-approximation of the maximum weight matching.

### 6.3.2 A fast \((1 + \varepsilon)\)-approximation of maximum cardinality matching in LOCAL

Here, we present an \(O\left(\frac{\log \Delta}{\log \log \Delta}\right)\)-round algorithm in the LOCAL model for \((1 + \varepsilon)\)-approximation of maximum unweighted matching for any constant \(\varepsilon > 0\). In the next subsection, we explain how to extend a variant of this algorithm to the CONGEST model, in essentially the same round complexity, i.e., without incurring a loss in the asymptotic notation.

Our algorithm follows a general approach due to the classical work of Hopcroft and...
Karp[HK73], where one iteratively augments the matching with short augmenting paths, until the desired approximation (or exact bound) is achieved. In our case, the key algorithmic piece is to efficiently find a nearly-maximal set of disjoint short augmenting paths, in $O(\log \Delta/\log \log \Delta)$ rounds. Our base will again be Ghaffari’s MIS algorithm[Gha16]. However, here we need significant changes to the algorithm and its analysis.

**Augmenting Paths.** Consider an arbitrary matching $M$. A path $P = v_0, v_1, v_2, \ldots, v_p$ is called an *augmenting path* of length $p$ for $M$ if in $M$, we have the following two properties: (1) nodes $v_0$ and $v_p$ are unmatched, and (2) for every $2i + 1 \in [1, p - 1]$, node $v_{2i+1}$ is matched to node $v_{2i+2}$. In this case, let $M \oplus P = M \cup P \setminus (M \cap P)$. That is, $M \oplus P$ is the matching obtained by erasing the matching edges $\{v_1, v_2\}, \{v_3, v_4\}, \ldots, \{v_{p-2}, v_{p-1}\}$, and instead adding edges $\{v_0, v_1\}, \{v_2, v_3\}, \ldots, \{v_{p-1}, v_p\}$. Note that $M \oplus P$ is indeed a matching, and moreover, it has one more matching edge than $M$. The operation of replacing $M$ with $M \oplus P$ is called *augmenting* the matching $M$ with the path $P$.

For a fast distributed algorithm, we would like to be able to augment the matching $M$ with many augmenting paths simultaneously. For a given matching $M$, two augmenting paths $P_1$ and $P_2$ are called *dependent* if their node-sets intersect. Note that in that case, we can not augment $M$ simultaneously with both $P_1$ and $P_2$. However, in case we have two independent augmenting paths, we can augment $M$ with both of them simultaneously, and the result is the same as first performing the first augmentation, and then performing the second.

We now recall two well-known facts about augmenting paths, due to the classical work of Hopcroft and Karp[HK73]: (1) Matching $M$ is a $(1 + \varepsilon)$-approximation of the maximum matching if and only if it does not have an augmenting path of length at most $2\lceil 1/\varepsilon \rceil + 1$. (2) If the shortest augmenting path for $M$ has length $\ell$ and one augments $M$ with a maximal independent set of augmenting paths of length $\ell$, the shortest augmenting path of the resulting matching will have length at least $\ell + 1$.

**General Methodology.** Based on the above two facts, a natural and by now standard method for computing a $(1 + \varepsilon)$-approximation of maximum matching is as follows: for each $\ell = 1, \ldots, 2\lceil 1/\varepsilon \rceil + 1$, we find a maximal independent set of augmenting paths of length exactly $\ell$ and augment the matching with them. At the end, we have a $(1 + \varepsilon)$-approximation of maximum matching. This outline was followed by Fischer et al.[FGHP93] in the PRAM model and Lotker et al.[LPSP15] in the distributed model.

As clear from the above outline, the core algorithmic piece is to compute a maximal independent set of augmenting paths of length $\ell$. We consider an auxiliary graph with one node per each augmenting path of length $\ell$ and an edge between each two of them if they intersect. This auxiliary graph, which is usually called the *conflict graph*, can be constructed and simulated in $\ell = O(1/\varepsilon)$ rounds of communication on the base graph in the LOCAL model. Thus, the remaining question is how to find a maximal independent set on this graph. Lotker et al.[LPSP15] used a variant of Luby’s distributed MIS algorithm[Lub86] to compute this set in $O(\log n)$ rounds. However, aiming at the complexity of $O(\log \Delta/\log \log \Delta)$, we cannot afford to do that. Indeed, it remains open whether an MIS can be computed in $O(\log \Delta/\log \log \Delta)$ rounds. Thus, unless we resolve that question, we cannot compute a truly maximal independent set. Our remedy is to
resort to computing “nearly-maximal” sets, using ideas similar to the algorithm of the previous section. Here, the near-maximality should be according to an appropriate definition which allows us to preserve the approximation guarantee. However, there are crucial subtleties and challenges in this point, which require significant alterations in the algorithm, as discussed next.

**Intuitive Discussion of the Challenge.** To be able to follow the general method explained above and get its approximation guarantee, we need to ensure that no short augmenting path remains. However, the set of augmenting paths that we compute are not exactly maximal, which means some paths might remain. A natural solution would be that, after finding a “nearly-maximal” set of augmenting paths of a given length, we neutralize/deactivate the rest, say by removing one node of each of these remaining paths from the graph. However, to ensure that we do not lose in the approximation factor, we need to be sure that this removal does not damage the maximum matching size significantly. For instance, if we can say that each node is removed with a small probability $\delta \ll \epsilon$, in expectation this can remove at most a $2\delta$ fraction of the optimal matching edges, and thus $(1+\epsilon')(1+2\delta) \approx 1+\epsilon'+2\delta$. This is a good enough approximation, as we can choose the $\epsilon'$ and $\delta$ appropriately, e.g., about $\epsilon/10$. However, in our context, running the nearly-maximal independent set algorithm of the previous subsection among augmenting paths for $O(\log \Delta/ \log K + K^2 \log 1/\delta)$ rounds would only guarantee that the probability of each one augmenting path remaining is at most a small $\delta$. Since there can be up to $\Delta O(1/\epsilon)$ augmenting paths going through one node, and as we want the running time within $O(\log \Delta/ \log \log \Delta)$, we cannot afford to apply a union bound over all these paths and say that the probability of each one node having a remaining augmenting path is small. The fix relies on some small changes and a much tighter analysis of the nearly-maximal independent set algorithm for this special case, leveraging the fact we are dealing with paths of constant length at most $d = O(1/\epsilon)$. In fact, to present the fix in its general form, we turn to another (equivalent) formulation of finding nearly-maximal matchings—that is, a set of hyperedges where each node has at most one of its hyperedges in this set—in a hypergraph $H$ of rank $d = O(1/\epsilon)$. The connection is that we will think of augmenting paths as hyperedges of $H$ and nodes of $H$ will be the same as nodes of the original graph $G$, where a hyperedge in $H$ includes all nodes of the corresponding augmenting path in $G$.

**Nearly-Maximal Matching in Low-Rank Hypergraphs.** We want an algorithm for hypergraphs of rank $d$ that in $O(d^2 \log \Delta/ \log \log \Delta)$ rounds, deactivates each node with probability at most $\delta \ll \epsilon$, and finds a maximal matching in the hypergraph induced by active nodes. Note that this is stronger than guaranteeing that each edge is removed or has an adjacent edge in the matching with such a probability. The algorithm will be essentially the same as that of the previous subsection, where now each hyperedge $e$ has a probability $p_t(e)$ for each iteration $t$ and gets marked and joins the matching accordingly. We will however deactivate some nodes in the course of the algorithm. The more important new aspect is in the analysis.

**The Change in the Algorithm.** Call a hyperedge $e$ light iff $\sum_{e', e' \cap e \neq \emptyset} p_t(e') < 2$, and let $L_t$ be the set of light hyperedges of round $t$. Set $K = \log 0.1 \Delta$. Call a round $t$ good for a node $v$ if $\sum_{e, e' \in L_t, v \in e} p_t(e) \geq 1/(2dK^2)$. Note that in a round that is good for $v$, with probability at least
\(\Theta(\frac{1}{dK^2})\), one of these light hyperedges joins the matching and thus \(v\) gets removed. Deactivate node \(v\) if it has had more than \(\Theta(dK^2 \log 1/\delta)\) good rounds. Note that the probability that a node \(v\) survives through \(\Theta(dK^2 \log 1/\delta)\) good rounds and then gets deactivated is at most \(\delta\).

**Analysis.** A key property of the algorithm is the following deterministic guarantee, which proves the maximality of the found matching in the hypergraph induced by active nodes:

**Lemma 6.3.3.** After \(T = O(d^2 \frac{\log \Delta}{\log \log \Delta})\) rounds, there is no hyperedge with all its nodes active.

**Proof.** We consider an arbitrary hyperedge \(e = \{v_1, v_2, \ldots, v_d\}\) and prove that it cannot be the case that all of its nodes remain active for \(O(d^2 \frac{\log \Delta}{\log \log \Delta})\) rounds. We emphasize that this is a deterministic guarantee, and it holds for every hyperedge \(e\). We assume that hyperedge \(e\) is not removed (by removing itself due to an adjacent edge in the matching, or because of one of its nodes becoming deactivated) in the first \(T = O(d^2 \frac{\log \Delta}{\log \log \Delta})\) rounds and we show that this leads to a contradiction, assuming a large enough constant in the asymptotic notation definition of \(T\).

For each node \(v\), call a round heavy if \(\sum_{e,v \in e} p_t(e) \geq 1/d\). If round \(t\) is heavy but not good, then by definition of good rounds, at most \(1/(dK^3)\) weight in the summation comes from light hyperedges. This is at most a \(\frac{1/(2dK^2)}{1/d} = 1/2K^2\) fraction of the summation. Hence, in every heavy but not-good round, the summation shrinks by a factor of \(1/(2K^2) \cdot K + (1 - 1/(2K^2)) \cdot 1/K \leq 2/K\).

In each heavy and good round, the summation grows by at most a \(K\) factor, unless canceling at most 2 of the shrinkage rounds (as in the proof of Lemma 6.3.1). The number of good rounds is at most \(\Theta(dK^2 \log 1/\delta)\). Therefore, since \(\sum_{e,v \in e} p_t(e)\) starts with a value of at most \(\Delta^d/K\), node \(v\) can have at most \(\Theta(dK^2 \log 1/\delta) + 3d \log K \Delta\) heavy rounds.

Now, looking at a hyperedge hyperedge \(e = \{v_1, v_2, \ldots, v_d\}\), we claim that \(e\) cannot have more than \(d(\Theta(dK^2 \log 1/\delta) + 3d \log K \Delta)\) rounds in which \(\sum_{e' \in e, e' \neq \emptyset} p_t(e') \geq 1\). This is because in every such round, the summation in at least one of the \(d\) nodes constituting \(e\) must be at least \(1/d\). Thus, in every such round, at least one of the nodes of hyperedge \(e\) is heavy. But we just argued that each node has at most \(\Theta(dK^2 \log 1/\delta) + 3d \log K \Delta\) heavy rounds. Thus, in total edge \(e\) cannot have more than \(h = \Theta(d^2 K^2 \log 1/\delta) + 3d \log K \Delta\) rounds in which \(\sum_{e' \in e, e' \neq \emptyset} p_t(e') \geq 2\).

During each round in which \(\sum_{e' \in e, e' \neq \emptyset} p_t(e') \geq 2\), hyperedge \(e\) reduces its \(p_t(e)\) by a \(K\) factor. Each other round raises \(p_t(e)\) by a \(K\) factor, unless \(p_t(e)\) is already equal to \(1/K\). Since each \(K\)-factor raise cancels one \(K\)-factor shrinkage, and as \(p_t(e)\) starts at \(1/K\), with the exception of \(2h\) rounds, all remaining rounds have \(p_t(e) = 1/K\). Among these, at most \(h\) can be rounds in which \(\sum_{e' \in e, e' \neq \emptyset} p_t(e') \geq 2\). Thus, hyperedge \(e\) has \(T - 3h\) rounds in which \(p_t(e) = 1/K\) and \(\sum_{e' \in e, e' \neq \emptyset} p_t(e') < 2\). These are indeed good rounds for all of nodes \(\{v_1, v_2, \ldots, v_d\}\). But we capped the number of good rounds for each node to \(\Theta(dK^2 \log 1/\delta)\). This is a contradiction, if we choose the constant in \(T\) large enough.

Now, we are ready to put together these pieces and present our \((1 + \varepsilon)\)-approximation:

**Theorem 6.7.** There is a distributed algorithm in the LOCAL model that computes a \((1 + \varepsilon)\) approximation of maximum unweighted matching in \(O(\frac{\log \Delta}{\log \log \Delta})\) rounds, for any constant \(\varepsilon > 0\).

**Proof.** We have \(O(1/\varepsilon)\) many phases, in each of which we find a nearly-maximal independent set of augmenting paths of length \(d\) for \(d = 1, 2, \ldots, O(1/\varepsilon)\). Each phase takes \(\Theta(d^2 K^3 \log 1/\delta + \cdots)\)
\(d^2 \log \epsilon \Delta\) rounds. We will set \(\delta = \Theta(\epsilon^2)\) and \(K = \log^{0.1} \Delta\). Hence, this is a complexity of \(O(\frac{\log \Delta}{\log \log \Delta})\) per phase, and thus at most \(O(\frac{\log \Delta}{\epsilon^2 \log \log \Delta})\) overall, which is \(O(\frac{\log \Delta}{\log \log \Delta})\) for any constant \(\epsilon > 0\).

In each of these phases, each node might get deactivated with probability at most \(\delta\). Hence, the overall probability of a node becoming deactivated is at most \(\delta' = O(\frac{\epsilon}{\Delta})\). As 6.3.3 implies, in each phase, the found set of augmenting paths of length \(d\) is indeed maximal in the graph induced by active nodes. Hence, after \(O(1/\epsilon)\) phases, there is no augmenting path of length less than \(O(1/\epsilon)\) among the active nodes. Thus, the matching at that point is a \((1 + \epsilon/2)\)-approximation of the maximum matching in the graph induced by active nodes. Throughout the iterations, we deactivate each node with probability \(\delta' = O(\frac{\epsilon}{\Delta})\). Considering the optimal matching \(OPT\), the expected number of matching edges of \(OPT\) that we remove by deactivating their nodes is at most \(2\delta'|OPT|\).\(^3\) On the remaining nodes, the matching we have found is a \((1 + \epsilon/2)\)-approximation of the maximum matching. Hence, overall, the found matching is a \((1 + \epsilon/2)(1 + 2\delta')\)-approximation. Setting \(\delta = \Theta(\epsilon^2)\), this is a \((1 + \epsilon)\)-approximation.

6.3.3 A fast \((1 + \epsilon)\)-approximation of maximum cardinality matching in CONGEST

Here we extend a suitably modified variant of the algorithm of the previous subsection to the CONGEST model. This will provide a \((1 + \epsilon)\)-approximation of maximum cardinality matching in \(O(\frac{\log \Delta}{\log \log \Delta})\) rounds of the CONGEST model, for any constant \(\epsilon > 0\). The key component will be a CONGEST-model algorithm for computing a nearly-maximal independent set of augmenting paths of length at most \(O(1/\epsilon)\) in bipartite graphs. We then utilize a method of Lotker et al.[LPSP15] to use \(2^{O(1/\epsilon)}\) iterations of applying this component to obtain a \((1+\epsilon)\)-approximation for maximum cardinality matching in general graphs.

Finding Augmenting Paths in Bipartite Graphs. Bipartite graphs provide a nice structure for augmenting paths, which facilitates algorithmic approaches for computing them. This was first observed and used by Hopcroft and Karp[HK73] and was later used also by others, including Lotker et al.[LPSP15] in the distributed setting. Consider a bipartite graph \(H = (A,B,E)\) and a matching \(M \subseteq E\) in it. Imagine orienting all matching \(M\) edges from \(A\) to \(B\) and all others from \(B\) to \(A\). Then, a path \(P\) is an augmenting path for \(M\) if and only if it is a directed path starting in an unmatched \(A\)-node and ending in an unmatched \(B\)-node. See 6.1. The three green dashed paths show three augmenting paths.

Considering the bipartite graph \(H\) and matching \(M\), we will find a nearly-maximal set of augmenting paths of length at most \(O(1/\epsilon)\) as follows. We go over various possible lengths one by one. For each odd \(d = 1, 3, \ldots, L = O(1/\epsilon)\), we find a set of augmenting paths of length \(d\) and then deactivate the remaining nodes who have augmenting paths of length \(d\). This deactivation will be done such that overall the probability of deactivating each node is small. We then move on to augmenting paths of \(d + 2\), and so on, until length \(L = O(1/\epsilon)\). This way, each time we are looking for augmenting paths of the shortest length; this helps in the computation.

\(^3\)We indeed have also a concentration around this expectation, especially if \(\Delta \leq n^\gamma\) for an appropriately chosen constant \(\gamma\), then the statement holds with high probability, modulo a 2 factor.
We now explain our method for computing a near-maximal set of augmenting paths of length $d$, in bipartite graphs where the shortest augmenting path has length (at least) $d$. We will not construct the conflict graph between the augmenting paths explicitly. Instead, we will try to emulate the algorithm on the fly by means of simple communications on the base graph. In fact, the emulation will not be truthful and the implementation will deviate from the ideal LOCAL-model algorithm, described in the previous subsection. We will however show that these alterations do not affect the performance measures significantly.

As before, for each short augmenting path $P$ of length $d$, we will have a marking probability $p_t(P)$, for each iteration $t$. However, we will not insist on any one node knowing this probability; rather it will be known in a distributed manner such that we can still perform our necessary computations on it. For instance, a key part will be for each node $v$ to learn the summation of the probabilities $p_t(P)$ for augmenting paths $P$ of length $d$ that go through $v$.

Let us first explain our distributed representation of $p_t(P)$. We have a time-variable attenuation parameter $\alpha_t(v)$ for each node $v$ that is either an $A$-node or an unmatched $B$-node. We will change these attenuation parameters from iteration to iteration. For simplicity, let us extend the definition also to matched nodes in $B$ but keep in mind that for each such node $v$, we will always have $\alpha_t(v) = 1$. These attenuation parameters determine the marking probabilities of the augmenting paths as follows: for each augmenting path $P$ of length $d$, its marking probability is the multiplication of the attenuation parameters along the path, that is $p_t(P) = \prod_{v \in P} \alpha_t(v)$. We will later explain how we adjust these attenuation parameters over time.

We first explain how each node $v$ can learn the summation of the probabilities $p_t(P)$ for augmenting paths $P$ of length $d$ that go through $v$. For simplicity, let us first assume that $p_t(P)$ is the same for all these augmenting paths, and thus we only need to figure out the number of augmenting paths of length $d$ that go through each node $v$.

We will leverage the nice structure of the shortest augmenting paths here. This point was first observed and used by Lotker et al.[LPSP15]. They show that a clean breadth first search style of message passing allows one to compute these numbers. The method is as follows. The reader might find 6.1 helpful while following this discussion. Notice that we are looking only for
augmenting paths of length exactly $d$. Each unmatched $A$-node starts a counter 1. Then, the algorithm proceeds for $d$ rounds, to find augmenting paths of length $d$ hop by hop. In each odd round, each $A$ node passes its count to all its $B$-neighbors. In each even round $t$, each matched $B$ node $b$ passes the summation of the numbers it received from its $A$-neighbors in the previous round $t-1$ to its matching-mate in $A$. Moreover, $b$ does this only in one round $t$ as any later round indicates augmenting paths with length greater than $d$. If $t = d$, each unmatched $B$-node keeps the summation to itself without passing it on. Using an induction on the hop-counter, one can show that at the end, each unmatched node $b \in B$ learns the number of the augmenting paths of length $d$ that end at $b$. In 6.1, the black numbers next to unmatched nodes or to matching edges indicate the numbers passed along during this forward traversal of augmenting paths. The red arrows indicate edges that are not a part of any shortest augmenting path, as in the BFS layers, they go back from a deeper layer to a shallower layer. These are edges for which the message would be sent after the receiving endpoint has already received summations in the previous rounds.

We next perform a similar procedure to ensure that all nodes know the respective number of short augmenting paths that go through them. This is by a simple time-reversal of these $d$ rounds of message passing. Start from each unmatched $B$-node $b$, and send back to each of its $A$-neighbors $a$ the number that $b$ received from $a$. Alternatively, this can be phrased as splitting the number that $b$ holds among its $A$-neighbors proportional to the numbers that it received from them during the forward traversal. Then, in each even round, each $A$-node passes to its matching mate in $B$ the summation of the numbers it received in the previous round from its $B$-neighbors. In each odd round, each $B$-node $b$ that received a number $x$ from its $A$-mate in the previous round splits this number $x$ among its $A$-neighbors $a$ proportional to the numbers that $b$ received from various $a$ during the corresponding round of the forward traversal of augmenting paths. After $d$ rounds, we reach back to the starting points of the augmenting paths; at that time, each unmatched $A$-node keeps the received summation to itself without passing it on.

In 6.1, the black numbers show the numbers passed on during the forward traversal, and the purple underlined numbers on the right side show the numbers sent during the backwards traversal for a few of the nodes.

**Claim 6.3.4.** The number each node $v$ receives during the backwards traversal is equal to the number of augmenting paths that go through the node $v$.

**Proof.** We first consider the forward traversal.

All a directed path a half-augmenting path if it starts at an unmatched $A$ node and alternates between unmatched and matched edges. We prove by induction on time $t \leq d$ that the number that each node $v$ receives in round $t$ is the number of (shortest) half-augmenting paths of length $t$ that end at $v$. The base case $t = 0$ is trivial as each unmatched $A$ node starts with number 1 and other nodes have 0. In odd rounds $t$, node $v$ receives its first message in this round only if $v \in A$. Then, it receives the message from its matching mate $b \in B$, and by induction, the number that $b$ passed along is exactly the number of half-augmenting paths of length $t-1$ ending at $b$. Since adding the edge $v \rightarrow b$ to each of these paths extends them by one hop, the number that $v$ receives is also the number of half-augmenting paths of length $t$ ending at $a$. In even rounds $t$, node $v$ receives in this round only if $v \in B$. Then, it received numbers from its $A$-neighbors,
each of them indicating the number of half-augmenting paths of length \( t - 1 \) that end at each 
neighbor \( a \in A \). Each of these half-augmenting paths can be extended with the edge \( a - b \), thus 
generating a half-augmenting paths of length \( t \) ending at \( v \). Hence, the number that \( v \) received is 
again indeed the number of shortest half-augmenting paths of length \( t \) ending at \( v \).

At the end, note that for \( t = d \), we only look at the numbers received by unmatched \( B \)-node 
\( b \) in round \( d \). By the above inductive argument, the number received by \( b \) is indeed the number of 
half-augmenting paths of length \( d \) ending at \( b \). But each of these is actually an augmenting 
path, as it ends in an unmatched \( B \)-node. Hence, we know that all unmatched \( B \)-nodes learn 
the number of augmenting paths of length \( d \) ending at them.

We now consider the backwards traversal, and show by an induction on time \( t \) in the backwards 
traversal that the number that each node \( v \) receives in round \( t \) of the backwards traversal is 
indeed the number of augmenting paths of length \( d \) that go through \( v \). The induction base 
\( t = 0 \) follows from the argument above, as these are unmatched \( B \)-nodes which are the supposed 
endpoints of augmenting paths. For the inductive step, we again have two cases:

The case of even rounds follows easily because the number of augmenting paths going through 
a matched \( B \)-node \( b \) is the same as the number of them that afterwards (in following the direction 
of the path) go through its mate \( a \in A \). But this number is known to \( a \) in the previous round of 
the backwards traversal, by induction. Hence, \( b \) also learns its number of shortest augmenting 
paths.

For the case of odd rounds, let us examine an \( A \)-vertex \( a \). During the forward traversal, in 
the corresponding round, the number that \( a \) sent to each of its \( B \)-neighbors \( b \) was the number 
of half-augmenting paths ending at \( a \). Each neighbor \( b \) now knows the number of shortest 
augmenting paths that continue by going through \( b \). This might be a collection of paths, where 
various fractions come from various prior nodes \( a \in A \). But the number for each prior node \( a \in A \) 
is exactly proportional to the number of half-augmenting paths that end in \( a \). Hence, when \( b \) 
splits its number proportionally among its \( a \)-neighbors (proportional to the numbers received 
in forward traversal), each neighbor \( a \) will learn the number of shortest augmenting paths that 
reach \( a \), go from \( a \) to \( b \), and then from \( b \) all the way to an unmatched \( B \)-node. Hence, node \( a \) 
can just sum up all the numbers received from various \( B \)-neighbors \( b \), and know the number of 
shortest augmenting paths going through \( a \).

Now we explain how a simple change in the previous message forwarding method makes 
each node know the probability summation of the paths that go through it. During the forward 
traversal, each unmatched \( A \)-node—which is a potential starting point of augmenting paths—
passes its attenuation to its \( B \)-neighbors, instead of passing a fixed number 1. Then, each 
matched \( B \)-node \( b \) passes the received summation to its \( A \)-mate \( a \). The matching mate \( a \) 
then attenuates this received summation via multiplying it by \( \alpha_t(a) \), and then passes it on 
to its \( B \)-neighbors. Each unmatched \( B \)-node just applies its attenuation parameter on the 
received summation and keeps the summation to itself. The backwards traversal is essentially as 
before (without reapplying attenuations): each \( B \)-node splits its number among its \( A \)-neighbors 
proportional to the received numbers during the forward traversal. Each matched \( A \) just passes 
the number to its matching mate in \( B \).
Claim 6.3.5. The number that each node \( v \) receives during the backwards traversal is equal to the summation of the \( p_t(P) \) for all augmenting paths \( P \) that go through the node \( v \).

Proof Sketch. The proof follows by repeating the argument of 6.3.4, and noting that along the path, the numbers are multiplied by the respective attenuations.

Adjusting Attenuation Parameters Over Time. Initially, we set \( \alpha_0(v) = 1/K \) for each unmatched \( A \)-node, which is the starting node of the potential augmenting paths, and \( \alpha_0(v) = 1 \) for each matched \( A \)-node or unmatched \( B \)-node, which are the potential middle or end nodes of the augmenting paths. The updates are as follows: For each node \( v \), if \( \sum_{v,v' \in P} p_t(P) \geq \frac{1}{\log 2} \), then set \( \alpha_{t+1}(v) = \max\{\alpha_t(v) \cdot (\frac{1}{K})^{2d}, \Delta^{-20/\varepsilon}\} \), and otherwise, if \( \alpha_{t+1}(v) = \min\{\alpha_0(v), \alpha_t(v) \cdot K\} \).

Remark About the Attenuation Lower Bound and the Floating-Point Precisions. Notice that we have set a lower bound of \( \Delta^{-20/\varepsilon} \) on the attenuations. We say a node \( v \) is stuck to the bottom if \( \sum_{v,v' \in P} p_t(P) \geq \frac{1}{\log 2} \) which means \( v \) wishes to lower its attenuation, but we already have \( \alpha_t(v) = \Delta^{-20/\varepsilon} \) and thus it cannot go further down. As we will see, this limitation has no significant effect on the behavior of the algorithm, because over all the at most \( \Delta^d \ll \Delta^{-20/\varepsilon} \) shortest augmenting paths going through a node, those that are stuck to the bottom make only a very negligible fraction of the probability. However, this lower bound allows us to keep the message size necessary for passing around attenuation values small. Particularly, with this lower bound, each attenuation can fit \( O(\log \Delta/\varepsilon) \) bits of precision. Thus, even when multiplied over paths of length \( O(1/\varepsilon) \), and summed up over at most \( \Delta^{O(1/\varepsilon)} \) paths, we only need \( O(\log \Delta/\varepsilon^2) \) bits of precision to represent the resulting number. This certainly fits \( O(1/\varepsilon^2) \) messages of the standard CONGEST model. Hence, by grouping each \( \Theta(1/\varepsilon^2) \) consequent rounds of the CONGEST model and treating them as one round, we have enough space for the desired precision.

Definition 6.3.6. Call a node \( v \) heavy in a given iteration \( t \) if \( \sum_{v,v' \in P} p_t(P) \geq \frac{1}{\log 2} \). Moreover, call an augmenting path \( P \) heavy in that iteration if it goes through at least one heavy node.

Claim 6.3.7. Each heavy augmenting path of length \( d \) will have its \( p_t(P) \) decrease by a factor in \([((1/K)^{2d}, (1/K)^d)]\), unless at least one of its heavy nodes is stuck to the bottom, in which case the path’s probability will remain at most \( \Delta^{-20/\varepsilon} \). Each non-heavy augmenting path of length \( d \) will have its \( p_t(P) \) increase by a factor in \([K, K^d])\), unless \( p_t(P) = 1/K \) in which case it will remain there and we have \( p_{t+1}(P) = 1/K \).

Proof. If a path is heavy, at least one of its nodes is heavy and then that node multiplies its own attenuation parameter by \((1/K)^{2d}\), unless its attenuation is stuck to the bottom, in which case it remains there at \( \Delta^{-20/\varepsilon} \). The other \( d - 1 \) nodes might also shrink their attenuation factor similarly or they might raise it by a \( K \) factor, up to at most their original setting. In the lowest extreme, the overall multiplication of attenuation factors goes down by a \((1/K)^{2d}\) factor. In the highest extreme, \( d - 1 \) of them raise their factor by a \( K \) which means that we still have a shrinkage of \((1/K)^{2d} \cdot K^{d-1} < (1/K)^d\), if none of the heavy nodes is stuck to the bottom. If there is at least one such stuck heavy node, it will keep its attenuation at \( \Delta^{-20/\varepsilon} \), which means the overall marking probability of the path is at most \( \Delta^{-20/\varepsilon} \).
Now suppose that the path is not heavy. On the highest extreme, all $d$ nodes raise their attenuation parameters by a $K$ factor, which would mean an overall increase of a $K^d$ factor. On the lowest extreme, either the path already has all its attenuation parameters set as in the beginning, hence has $p_t(P) = 1/K$, or at least one of its nodes raises its attenuation by a $K$ factor. Since the path is not-heavy, by definition, there is no heavy node on it and thus none of the nodes will reduce its attenuation parameter. Therefore, in this case, the marking probability raises by a $K$ factor, unless it is already at $1/K$, in which case it stays the same.

The Algorithm for Marking Augmenting Paths. We now explain how to use the marking probabilities $p_t(P)$, which are maintained implicitly by means of attenuation parameters, to mark augmenting paths and find a large independent set of them. Each free $B$-node $b$ has a summation $z = \sum_{P : b \in P} p_t(P)$ of all augmenting paths $P$ that end at $b$. If the summation $z$ is greater than $1/d$, which means node $b$ is heavy, node $b$ will not initiate any path marking. Otherwise, node $b$ tosses a coin which comes out head with probability $z$, and if head, then it initiates an augmenting path marking operation. In this case, node $b$ passes the path marking token to one of its matched $A$-neighbors, as in the backwards traversal, chosen with a probability proportional to the sums that $b$ received from those neighbors during the forward traversal. If two marking tokens arrive at the same node, they both die. If a token is the only one that has arrived at a matched node $a'$, it will be passed to the matching mate $b'$ of $a'$. Then, $b'$ passes this token to one of its $A$-neighbors, again chosen with probabilities proportional to the sums that $b'$ received from those neighbors during the forward traversal. After continuing this process for $d$ iterations, some tokens might arrive at unmatched $A$-nodes. These are the marked paths that do not have any marked intersecting paths, they will be added to the independent set of short augmenting paths.

Note that instead of this stochastic link-by-link sampling of the marked path, one could have imagined a one-shot process of sampling the marked path. In that process, the endpoint had sampled this full path at the beginning (if it knew the whole topology), we just pass the token along the path, and then the path is maintained if no intersecting path was marked. Indeed, one can easily see that the former process can only generate a larger set of isolated marked paths.

The tokens that make it through all the way and reach to an unmatched $A$ node are successful tokens. To announce that to all nodes along the path, these tokens will reverse their direction of traversal and go back all the way to their origin in the unmatched $B$-node. This can be done easily by each node remembering where from it received each of the tokens. While doing that, the token deactivates all nodes of the path, removing them from the problem. This also effectively augments $M$ with this path, by erasing the matching edges of the path, and substituting them with the unmatched edges.

Claim 6.3.8. Each non-heavy path $P$ gets marked and removed with probability at least $9p_t(P)/10$.

Proof. As stated before, we can think of the link by link creation of the marked path as a one-shot sampling by the path endpoint $b$ which is an unmatched $B$-node. Then the marking gets erased if there is another marked path intersecting it. For the non-heavy path $P$, the probability that it gets an initial marking in this sense is exactly $p_t(P)$, as its unmatched endpoint $b$ will toss a coin
with probability equal to the summation of all paths that end at $b$, because it is not heavy, and then pass this token backwards for a link by link creation. We next argue that conditioned on the initial marking, the path has a decent probability of not having any other intersecting path get marked. Since the probability of each other path $P' \neq P$ getting an initial marking is at most $p_t(P')$, for each node $v$, the probability that $v$ receives a marking token through any of the paths $P' \neq P$ is at most $\sum_{v,v' \in P'} p_t(P') \leq \frac{1}{10^2}$, simply by a union bound. Thus, a union bound over all the at most $d$ nodes of the path $P$ shows that with probability at least $1 - \frac{d}{10^2}$, none of them has any other marked intersecting path and thus, the $P$ retains its mark and gets removed. \hfill \Box

Similar to 6.3.2, we call a node $v$ good in a marking iteration $t$ if the summation of the probabilities of the light paths going through node $v$ is large. Here, the particular definition will require the summation to be at least $1/(dK^{2d})$. If a node $v$ goes through $\Theta(dK^{2d} \log 1/\delta)$ good iterations without being removed, we manually remove $v$ from the problem, knowing that the probability of this event is at most $\delta$.

**Lemma 6.3.9.** The probability that a node $v$ does not get removed during $\Theta(dK^{2d} \log 1/\delta)$ good iterations is at most $\delta$.

**Proof.** Consider one iteration $t$ that is good for node $v$. We claim that in this iteration, node $v$ gets removed from the problem with probability at least $9/(10dK^{2d})$. This is because, as 6.3.8 shows, for each light path $P$ that goes through $v$, there is a $9p_t(P)/10$ probability that this path is marked for removal while no other path intersecting it is marked. Since these events are disjoint for different light paths $P$ going through $v$, we can say that the probability of $v$ being removed because one of those light paths was removed is at least $9/(10dK^{2d})$. Now, the probability that this does not happen in the course of $\Theta(dK^{2d} \log 1/\delta)$ good iterations is at most $\delta$. \hfill \Box

Gathering probability sums of light paths and thus identifying good nodes can be done easily by a repetition of the previous forward and backwards probability passing processes, but this time just letting them pass only through light nodes.

**Lemma 6.3.10.** After $T = O(d^4 \log \Delta / \log \log \Delta)$ iterations, there is no augmenting path of length $d$ remaining.

**Proof.** We consider an arbitrary augmenting path $P = (v_1, v_2, \ldots, v_d)$ and prove that it cannot be the case that all of its nodes remain active for $T = \Theta(d^4 K^{2d} \log 1/\delta + d^3 \log K \Delta)$ iterations. Setting $K = \log 1/(3d)$, this is $O(d^4 \frac{\log \Delta}{\log \log \Delta})$ iterations for any $\delta \geq 2^{-\log^{0.3} \Delta}$. We assume that path $P$ is not removed in the first $T = O(d^4 \frac{\log \Delta}{\log \log \Delta})$ iterations and we show that this leads to a contradiction, assuming a large enough constant in the asymptotic notation definition of $T$. We emphasize that this is a deterministic guarantee, and it holds for every such augmenting path.

For a node $v \in P$, if iteration $t$ is heavy but not good, then at most $1/(dK^{2d})$ weight in the summation can come from light paths going through $v$. That is at most a $1/(dK^{2d})$ fraction of the summation, as a heavy node has summation at least $1/d$. Hence, ignoring the paths that their probability is stuck to the bottom, in every heavy but not-good iteration, the summation shrinks by a factor of $1/(K^{2d}) \cdot K^d + (1 - 1/(K^2)) \cdot (1/K)^d \leq 2/(K^d)$. At most
$\Delta^{2/\varepsilon} \cdot \Delta^{-20/\varepsilon}$ weight can come from paths that their probability is stuck to the bottom bound of $\Delta^{-20/\varepsilon}$. This is a total weight of $\Delta^{-18/\varepsilon}$, which for a heavy node is at most $\frac{\Delta^{-18/\varepsilon}}{1/\varepsilon} \ll 1/K^d$ fraction of the overall weight. Hence, even taking these paths into account, we see a shrinkage by a $3/K^d$ factor.

In each heavy and good iteration, the summation grows by at most a $K^d$ factor, which is in effect like canceling no more than 2 of the shrinkage iterations. The number of good iterations for $v$ is capped to at most $\Theta(dK^2d \log 1/\delta)$, after which $v$ gets deactivated. Therefore, since $\sum_{e,v \in E} p_t(e)$ starts with a value of at most $\Delta^d/K$, and shrinks by a $3/K^d$ factor in every non-canceled heavy but not-good iteration, node $v$ can have at most $h = \Theta(dK^2d \log 1/\delta) + 3 \log_K \Delta$ heavy iterations. Now this implies that the augmenting path $P = (v_1, v_2, \ldots, v_d)$ cannot have more than $h = \Theta(dK^2d \log 1/\delta) + 4d \log_K \Delta$ iterations in which it is heavy. This is because, in every heavy iteration for $P$, at least one of its $d$ nodes must be heavy, and each of them has at most $\Theta(dK^2d \log 1/\delta) + 4 \log_K \Delta$ heavy iterations.

As 6.3.7 shows, during each iteration that path $P$ is heavy, its probability shrinks by a $(\frac{1}{K})^{2d^2}$ factor, at worst (i.e., on the lower extreme). Each other iteration raises $p_t(P)$ by at least a $K$ factor, unless $p_t(e)$ is already equal to $1/K$. Since every $2d^2$ many of $K$-factor raises cancel one $(\frac{1}{K})^{2d^2}$-factor shrinkage, and as $p_t(e)$ starts at $1/K$, with the exception of $(2d^2 + 1)h$ iterations, all remaining iterations have $p_t(e) = 1/K$. Among these, at most $h$ can be iterations in which the path is heavy. Thus, hyperedge $e$ has $T - (2d^2 + 2)h$ iterations in which $p_t(e) = 1/K$ and it is not heavy. These are indeed good iterations for all of nodes $\{v_1, v_2, \ldots, v_d\}$. But we capped the number of good iterations for each node to $\Theta(dK^2d \log 1/\delta)$. This is a contradiction, if we choose the constant in $T$ large enough.

**Theorem 6.8.** There is a distributed algorithm in the CONGEST model that computes a $(1 + \varepsilon)$-approximation of maximum unweighted matching in $O(\frac{\log \Delta}{\log \log \Delta})$ rounds, for any constant $\varepsilon > 0$.

**Proof.** As mentioned before, we utilize a method of Lotker et al. [LPSP15] to compute a $1 + \varepsilon$ by finding (nearly-)maximal independent sets of short augmenting paths in bipartite graphs.

Lotker et al. [LPSP15] explain a clever method for $(1 + \varepsilon)$-approximation of Maximum Cardinality Matching in general graphs by, in a sense, randomly transforming the problem into bipartite graphs. Concretely, there are $2^{O(1/\varepsilon)}$ stages. In each stage, we get a bipartite graph and we need to find a (nearly-)maximal independent set of augmenting paths of length at most $2[1/\varepsilon] - 1$ in this bipartite graph. Then, we would augment the current matching with the found set of augmenting paths, and repeat. As Lotker et al. show, at the end, the found matching would be a $(1 + \varepsilon)$-approximation.

To find augmenting paths in bipartite graphs, we use the method devised and explained above. Particularly, given the bipartite graph, we work on length values $d = 1, 3, \ldots, 2[1/\varepsilon] - 1$ one by one, in each length, we deactivate a small $\delta$ fraction of nodes and find an independent set of augmenting paths of length $d$ that is maximal among active nodes. This step takes $T = O(d^{2} \log \Delta \log \Delta)$ iterations, and each iteration can be implemented in $O(d^{2}/\varepsilon^2)$ rounds, where the $d$-factor comes from the length of the path and the fact that we have to traverse it, and the $1/\varepsilon^2$ factor comes from the fact that we need to send around messages that need $O(\log n/\varepsilon^2)$ bits. Overall, this is at most $O(\frac{d^{2} \log \Delta}{\varepsilon^2 \log \log \Delta^2})$ rounds per iteration and thus at most $O(\frac{d^{2} \log \Delta}{\varepsilon^2 \log \log \Delta^2})$ rounds.
per stage, for going through all the possible length values. That is still $O(\text{poly}(1/\varepsilon) \cdot \frac{\log \Delta}{\log \log \Delta})$ rounds.

The method for generating the bipartite graphs is quite clean: color each node randomly red or blue, each with probability $1/2$. Then, keep each node in the bipartite graph if it is unmatched, or if it is matched but its matching edge becomes bi-chromatic. Also, keep all bi-chromatic edges supported on these nodes. This is clearly a bipartite graph and moreover, the step of creating this graph can be easily performed distributedly.

Each time that we use the nearly-maximal independent set algorithm, the guarantee is that, except for a negligible portion of at most $\delta |OPT|$ nodes which we excuse and deactivate, for a small $\delta \ll \varepsilon$, the found augmenting paths are maximal in that bipartite graph among the remaining active nodes. Overall, we will deactivate each node with probability at most $\delta' = \delta 2^{O(1/\varepsilon)}$. By choosing $\delta = 2^{-\Omega(1/\varepsilon)}$, which increases the round complexity of the near-maximal augmenting path algorithm only by an $O(1/\varepsilon)$ factor, we can ensure that $\delta' \ll \varepsilon$. Thus, the overall approximation would remain at most $(1 + \varepsilon)(1 + \delta') \leq 1 + 2\varepsilon$. Moreover, the round complexity over all stages is $O(2^{O(1/\varepsilon)} \cdot \frac{\log \Delta}{\log \log \Delta})$.

6.4 Alternative Fast Method for $2 + \varepsilon$ Approximation of Unweighted Maximum Matching

Here, we explain an alternative method for computing a $2 + \varepsilon$-approximation of maximum unweighted matching. This approach might be interesting especially because of its simplicity, but perhaps also because of the better probability concentration that it provides on the approximation.

6.4.1 The Algorithm for Bipartite Unweighted Graphs

The Algorithm. In each round, each node $v$ on the left side of the bipartite graph sends a matching proposal on a randomly chosen one of its remaining edges to right side nodes. Each right side node accepts the proposal from the highest id, if there is one.

Lemma 6.4.1. For any $K$, the algorithm produces a matching with approximation factor $2 + \varepsilon$, in $O(K \log 1/\varepsilon + \log \Delta / \log K)$ rounds, with high probability. Particularly, each left-node in the optimal matching remains unmatched but non-isolated with probability at most $\varepsilon/2$. Optimizing over $K$ leads to round complexity $O(\log \Delta / \log(\log \Delta / \log(1/\varepsilon)))$.

Proof. Let us call a left node unlucky if it remains unmatched but non-isolated after $O(K \log 1/\varepsilon + \log \Delta / \log K)$ rounds. We first show that for each left node $v$, the probability that $v$ is unlucky is at most $\varepsilon/2$. Moreover, this depends only on the randomness of $v$ and it holds regardless of the randomness used by other nodes.

The key part in that is to show that in each round, either $v$’s degree falls by a $K$ factor or its proposal succeeds with probability at least $1/K$. To prove this intermediate claim, let’s examine the proposals of the left nodes in descending order of ids. When we reach the turn of node $v$, either less than $1/K$ fraction of $v$’s right neighbors remain unmatched, in which case $v$’s degree has fallen by a $K$ factor, or otherwise $v$ has a chance of at least $1/K$ that its proposal was to a a currently unmatched neighbor, in which case $v$ would be successfully matched.
After $O(K \log 1/\varepsilon + \log \Delta/\log K)$ rounds, the probability that $v$ remains unmatched but non-isolated is at most $\varepsilon/2$. This is because, there can be at most $\log \Delta/\log K$ rounds where degree falls by a $K$ factor and the probability of failing in each other round is at most $1 - 1/K$.

Let us now examine the approximation factor. Consider the OPT matching. Each left-node of OPT gets unlucky with probability at most $\varepsilon/2$. Hence, at most $\varepsilon$ fraction of its left nodes get unlucky, with probability at least $1 - e^{-\Omega(\varepsilon|OPT|)}$. In the remainder, each edge of the found matching can kill at most 2 edges of the OPT matching. So the found matching is a $(2 + \varepsilon)$-approximation, with high probability.

### 6.4.2 The Algorithm for General Unweighted Graphs

We solve the general case by randomly transforming it to the bipartite case. In each iteration, randomly call each node left or right with probability half. This produces a bipartite graph which preserves each edge of the OPT matching with probability $1/2$. Run the bipartite algorithm on this graph and then remove the found matching edges, and all $G$-edges incident on their endpoints. Repeat this experiment $O(\log 1/\varepsilon)$ times.

**Lemma 6.4.2.** The algorithm finds a $(2 + \varepsilon)$-approximation of the maximum matching in general graphs in $O(\log 1/\varepsilon \cdot \frac{\log \Delta}{\log(\log \Delta/\log(1/\varepsilon))})$ rounds.

**Proof.** The time complexity is immediate as the algorithm is made of $O(\log 1/\varepsilon)$ repetitions of the bipartite algorithm. We next present the approximation analysis. Consider the OPT matching $\mathcal{M}$, and let us examine what happens to one edge $e = \{v, u\}$ where $e \in \mathcal{M}$. In the first iteration that either of nodes $v$ and $u$ is matched (not necessarily to each other), we charge the blame of $e$ not being in our matching to that found matching edge incident on $v$ or $u$ (to exactly one of them if there are two). This way, each found matching edge $e'$ might be blamed for at most 2 matching edges of OPT, one from each endpoint of $e'$. Now, in each of the first $O(\log 1/\varepsilon)$ iterations, either edge $e$ is already removed from the graph because at least one of its endpoints was matched before, or with probability $1/2$ edge $e$ gets preserved in the bipartite graph, in which case there are only two ways it is not taken into the matching: (1) one of its endpoints gets matched to a different node, (2) the left node endpoint got unlucky in the bipartite algorithm. Note that the latter happens with probability at most $\varepsilon/4$, as the bipartite algorithm guarantees. Throughout the $O(\log 1/\varepsilon)$ iterations, with probability $1 - \varepsilon/4$, edge $e \in \mathcal{M}$ is preserved in at least one iteration, unless it was already removed. Hence, we can say that with probability $1 - \varepsilon/2$, edge $e \in \mathcal{M}$ has put its blame on some other edge added to the matching. Note that we also have independence between different matching edges edge in $\mathcal{M}$, as (1) whether they get unlucky in the bipartite algorithm is independent, and (2) whether they get preserved in a bipartite transformation is independent. Thus, by Chernoff bound, at most $\varepsilon$ fraction of OPT edges have not put their blame on some edge in our found matching, with probability at least $1 - e^{-\Omega(\varepsilon|OPT|)}$. Since each found matching edge is blamed at most twice, the found matching is a $(2 + \varepsilon)$-approximation, with high probability.
Chapter 7

Distributed property testing

In this chapter we present a thorough study of distributed property testing – producing algorithms for the approximation problems of property testing in the CONGEST model. In particular, for the so-called dense graph testing model we emulate sequential tests for nearly all graph properties having 1-sided tests, while in the general model we obtain faster tests for triangle-freeness and cycle-freeness, and in the sparse model we obtain a faster test for bipartiteness. In addition, we show a logarithmic lower bound for testing bipartiteness and cycle-freeness, which holds even in the stronger LOCAL model.

In most cases, aided by parallelism, the distributed algorithms have a much shorter running time than their counterparts from the sequential querying model of traditional property testing. The simplest property testing algorithms allow a relatively smooth transitioning to the distributed model. For the more complex tasks we develop new machinery that may be of independent interest.

7.1 Distributed emulation of sequential tests in the dense model

We begin by showing that under a certain assumption of being non-disjointed, which we define below, a property $P$ that has a sequential test in the dense model that requires $q$ queries can be tested in the distributed setting within $O(q^2)$ rounds. We prove this by constructing an emulation that translates sequential tests to distributed ones. For this we first introduce a definition of a witness graph and then adapt [GT03, Theorem 2.2], restricted to 1-sided error tests, to our terminology.

Definition 7.1.1. Let $P$ be a property of graphs with $n$ vertices. Let $G'$ be a graph with $k < n$ vertices. We say that $G'$ is a witness against $P$, if it is not an induced subgraph of any graph that satisfies $P$.

Notice that if $G'$ has an induced subgraph $H$ that is a witness against $P$, then by the above definition $G'$ is also a witness against $P$.

The work of [GT03] transforms tests of graphs in the dense model to a canonical form where the query scheme is based on vertex selection. This is useful in particular for the distributed model, where the computational work is essentially based in the vertices. We require the following special case for 1-sided error tests.
Lemma 7.1.2 ([GT03, Theorem 2.2]). Let $P$ be a property of graphs with $n$ vertices. If there exists a 1-sided error $\epsilon$-test for $P$ with query complexity $q(n, \epsilon)$, then there exists a 1-sided error $\epsilon$-test for $P$ that uniformly selects a set of $q' = 2q(n, \epsilon)$ vertices, and accepts if and only if the induced subgraph is not a witness against $P$.

Our emulation leverages Lemma 7.1.2 under an assumption on the property $P$, which we define as follows.

**Definition 7.1.3.** We say that $P$ is a non-disjointed property if for every graph $G$ that does not satisfy $P$ and an induced subgraph $G'$ of $G$ such that $G'$ is a witness against $P$, $G'$ has some connected component which is also a witness against $P$. We call such components witness components.

We are now ready to formally state our main theorem for this section.

**Theorem 7.1.** Any $\epsilon$-test in the dense graph model for a non-disjointed property that makes $q$ queries can be converted to a distributed $\epsilon$-test that takes $O(q^2)$ communication rounds.

The following lemma essentially says that not satisfying a non-disjointed property cannot rely on subgraphs that are not connected, which is exactly what we need to forbid in a distributed setting.

**Lemma 7.1.4.** The property $P$ is a non-disjointed property if and only if all minimal witnesses that are induced subgraphs of $G$ are connected.

Here minimal refers to the standard terminology, which means that no proper induced subgraph is a witness against $P$.

**Proof.** First, if $P$ is non-disjointed and $G$ does not satisfy $P$, then for every subgraph $G'$ of $G$ that is a witness against $P$, $G'$ has a witness component. If $G'$ is minimal then it must be connected, since otherwise it contains a connected component which is a witness against $P$, which contradicts the minimality of $G$.

For the other direction, if all the minimal witnesses that are induced subgraphs of $G$ are connected, then every induced subgraph $G'$ that is a witness against $P$ is either minimal, in which case it is connected, or is not minimal, in which case there is a subgraph $H$ of $G'$ which is connected and a minimal witness against $P$. The connected component $C$ of $G'$ which contains $H$ is a witness against $P$ (otherwise $H$ is not a witness against $P$), and hence it follows that $P$ is non-disjointed.

Next, we give the distributed test (Algorithm 8). The test has an outer loop in which each vertex picks itself with probability $5q/n$, collects its neighborhood of a certain size of edges between picked vertices in an inner loop, and rejects if it identifies a witness against $P$. The outer loop repeats two times because not only does the sequential test have an error probability, but also with some small probability we may randomly pick too many or not enough vertices needed to emulate it. Repeating the main loop twice reduces the error probability back to below $1/3$. In the inner loop, each vertex collects its neighborhood of picked vertices and checks if
its connected component is a witness against $P$. To limit communications this is done only for components of picked vertices that are sufficiently small: if a vertex detects that it is part of a component with too many edges then it accepts and does not participate until the next iteration of the outer loop.

Algorithm 8: Emulation algorithm with input $q$ for property $P$

<table>
<thead>
<tr>
<th>Variables:</th>
<th>$U_v$ edges known to $v$, $U'_v$ edges to update and send (temporary variables)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 perform 2 times</td>
<td>reset the state for all vertices</td>
</tr>
<tr>
<td>for each vertex $v$ simultaneously</td>
<td>Vertex $v$ picks itself with probability $5q/n$</td>
</tr>
<tr>
<td>if $v$ is picked then</td>
<td>Notify all neighbors that $v$ is picked</td>
</tr>
<tr>
<td></td>
<td>Set $U'_v = {(v, u) \in E\mid u \text{ is picked}}$ and $U_v = \emptyset$</td>
</tr>
<tr>
<td>perform 10q times</td>
<td># At each iteration $U_v$ is a subgraph of $v$’s connected component</td>
</tr>
<tr>
<td></td>
<td>$U'_v = U'_v \setminus U_v$ # only need recently discovered edges</td>
</tr>
<tr>
<td></td>
<td>$U_v = U_v \cup U'_v$ # add them to $U_v$</td>
</tr>
<tr>
<td>if $</td>
<td>U_v</td>
</tr>
<tr>
<td></td>
<td>Send $U'_v$ to all picked neighbours of $v$ # propagate known edges</td>
</tr>
<tr>
<td></td>
<td>Wait until the time bound for all other vertices to finish this iteration</td>
</tr>
<tr>
<td></td>
<td>Set $U'_v$ to the union of edge sets received from neighbors</td>
</tr>
<tr>
<td>if $U_v \cup U'_v$ is a witness against $P$ then</td>
<td>Vertex $v$ outputs reject (ending all operations)</td>
</tr>
<tr>
<td>else</td>
<td>Wait until the time bound for all other vertices to finish this iteration of the outermost loop</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>19 Every vertex $v$ that did not reject outputs accept</td>
<td></td>
</tr>
</tbody>
</table>

To analyze the algorithm, we begin by proving that there is a constant probability for the number of picked vertices to be sufficient and not too large.

**Lemma 7.1.5.** *The probability that the number of vertices picked by the algorithm is between $q$ and $10q$ is more than $2/3$.*

**Proof.** For every $v \in V$, we denote by $X_v$ the indicator variable for the event that vertex $v$ is picked. Note that these are all independent random variables. Using the notation $X = \sum_{v \in V} X_v$ gives that $E[X] = 5q$, because each vertex is picked with probability $5q/n$. Using the Chernoff Bound from Fact 3.4.1 with $\delta = 4/5$ and $\mu = 5q$, we can bound the probability of having too few picked vertices:

$$Pr[X < q] = Pr[X < (1 - \delta)\mu] < \left(\frac{e^{-\delta/5}}{(1 - (4/5))(1 - (4/5))}\right)^{5q} = \left(\frac{5}{e^4}\right)^q < \frac{1}{10}.$$ 

For bounding the probability that there are too many picked vertices, we use the other direction
of the Chernoff Bound with $\delta = 1$ and $\mu = 5q$, giving:

$$Pr[X > 10q] = Pr[X > (1 + \delta)\mu] < \left(\frac{e}{22}\right)^{5q} = \left(\frac{e^5}{210}\right)^q < \frac{2}{10}.$$ 

Thus, with probability at least $2/3$ it holds that $q \leq X \leq 10q$. 

Now, we can use the guarantees of the sequential test to obtain the guarantees of our algorithm.

**Lemma 7.1.6.** Let $P$ be a non-disjointed graph property. If $G$ satisfies $P$ then all vertices output accept in Algorithm 8. If $G$ is $\epsilon$-far from satisfying $P$, then with probability at least $2/3$ there exists a vertex that outputs reject.

**Proof.** First, assume that $G$ satisfies $P$. Vertex $v$ outputs reject only if it is part of a witness against $P$, which is, by definition, a component that cannot be extended to some $H$ that satisfies $P$. However, every component is an induced subgraph of $G$ itself, which does satisfy $P$, and thus every component can be extended to $G$. This implies that no vertex $v$ outputs accept.

Now, assume that $G$ is $\epsilon$-far from satisfying $P$. Since the sequential test rejects with probability at least $2/3$, the probability that a sample of at least $q$ vertices induces a graph that cannot be extended to a graph that satisfies $P$ is at least $2/3$. Because $P$ is non-disjointed, the induced subgraph must have a connected witness against $P$. We note that a sample of more than $q$ vertices does not reduce the rejection probability. Hence, if we denote by $A$ the event that the subgraph induced by the picked vertices has a connected witness against $P$, then $Pr[A] \geq 2/3$, conditioned on that at least $q$ vertices were picked.

However, a sample that is too large may cause a vertex to output accept because it cannot collect its neighborhood. We denote by $B$ the event that the number of vertices sampled is between $q$ and $10q$, and by Lemma 7.1.5 its probability is at least $2/3$. We bound $Pr[A \cap B]$ using Bayes’ Theorem, obtaining $Pr[A \cap B] = Pr[A|B]Pr[B] \geq (2/3)^2$. Since the outer loop consists of 2 independent iterations, this gives a probability of at least $1 - (1 - 4/9)^2 \geq 2/3$ for having a vertex that outputs reject.

We now address the round complexity. Each vertex only sends and receives information from its $q$-neighborhood about edges between the chosen vertices. If too many vertices are chosen we detect this and accept. Otherwise we only communicate the chosen vertices and their edges, which requires $O(q^2)$ communication rounds using standard pipelining. Together with Lemma 7.1.6, this proves Theorem 7.1.

### 7.1.1 Applications: $k$-colorability and perfect graphs

Next, we provide some examples of usage of Theorem 7.1. A result by Alon and Shapira [AS08] states that all graph properties closed under induced subgraphs are testable in a number of queries that depends only on $\epsilon^{-1}$. We note that, except for certain specific properties for which there are ad-hoc proofs, the dependence is usually a tower function in $\epsilon^{-1}$ or worse (asymptotically larger).

---

1Pipelining means that each vertex has a buffer for each edge, which holds the information (edges between chosen vertices, in our case) it needs to send over that edge. The vertex sends the pieces of information one after the other.
From this, together with Lemma 7.1.2 and Theorem 7.1, we deduce that if $P$ is a non-disjointed property closed under induced subgraphs, then it is testable, for every fixed $\epsilon$, in a constant number of communication rounds.

**Example – $k$-colorability:** The property of being $k$-colorable is testable in a distributed manner by our algorithm. All minimal graphs that are witnesses against $P$ (not $k$-colorable) are connected, and therefore according to Lemma 7.1.4 it is a non-disjointed property. It is closed under induced subgraphs, and by [AK02] there exists a 1-sided error $\epsilon$-test for $k$-colorability that uniformly picks $O(k \log(k)/\epsilon^2)$ vertices, and its number of queries is the square of this expression (note that the polynomial dependency was already known by [GGR98]). Our emulation implies a distributed 1-sided error $\epsilon$-test for $k$-colorability that requires $O(poly(k\epsilon^{-1}))$ rounds.

**Example – perfect graphs:** A graph $G$ is said to be perfect if for every induced subgraph $G'$ of $G$, the chromatic number of $G'$ equals the size of the largest clique in $G'$. Another characterization of a perfect graph is via forbidden subgraphs: a graph is perfect if and only if it does not have odd holes (induced cycles of odd length at least 5) or odd anti-holes (the complement graph of an odd hole) [CRST06]. Both odd holes and odd anti-holes are connected graphs. Since these are all the minimal witnesses against the property, according to Lemma 7.1.4 it is a non-disjointed property. Using the result of Alon-Shapira [AS08] we know that the property of a graph being perfect is testable. Our emulation implies a distributed 1-sided error $\epsilon$-test for being a perfect graph that requires a number of rounds that depends only on $\epsilon$.

### 7.2 Distributed test for triangle-freeness

In this section we show a distributed $\epsilon$-test for triangle-freeness. Notice that since triangle-freeness is a non-disjointed property, Theorem 7.1 gives a distributed $\epsilon$-test for triangle-freeness under the dense model with a number of rounds that is $O(q^2)$, where $q$ is the number of queries required for a sequential $\epsilon$-test for triangle-freeness. However, for triangle-freeness, the known number of queries is a tower function in $\log(1/\epsilon)$ [Fox10].

Here we leverage the inherent parallelism that we can obtain when checking the neighbors of a vertex, and show a test for triangle-freeness that requires only $O(\epsilon^{-2})$ rounds (Algorithm 9). Importantly, our algorithm works not only for the dense graph model, but for the general graph model (where distances are relative to the actual number of edges), which subsumes it. In the sequential setting, a test for triangle-freeness in the general model requires a number of queries that is some constant power of $n$ by [AKKR08]. Our proof actually follows the groundwork laid in [AKKR08] for the general graph model – their algorithm picks a vertex and checks two of its neighbors for being connected, while we perform the check for all vertices in parallel.

**Theorem 7.2.** There exists a distributed $\epsilon$-test in the general graph model for the property of containing no triangles, that requires $O(\epsilon^{-2})$ rounds.

Our line of proof follows that of [AKKR08], by distinguishing edges that connect two high-degree vertices from those that do not. Formally, let $b = 2\sqrt{\epsilon^{-1}m}$, where $m$ is the number of edges in the graph, and denote $B = \{v \in V \mid \deg(v) \geq b\}$. We say that an edge $e = (u, v)$ is
Algorithm 9: Triangle freeness test

1. for each vertex \(v\) simultaneously
2. perform 32\(\epsilon^{-2}\) times
3. Pick \(w_1, w_2 \in N(v), w_1 \neq w_2\) uniformly at random
4. Send \(w_2\) to \(w_1\) # Ask \(w_1\) if it is a neighbor of \(w_2\)
5. foreach \(w_u\) sent by \(u \in N(v)\) do # Asked by \(u\) if \(v\) is a neighbor of \(w_u\)
6. if \(w_u \in N(v)\) then
7. Send “yes” to \(u\)
8. else
9. Send “no” to \(u\)
10. if received “yes” from \(w_1\) then
11. reject (ending all operations)
12. accept (for vertices that did not reject)

light if \(v \notin B\) or \(u \notin B\), and otherwise, we say that it is heavy. That is, the set of heavy edges is \(H = \{(u,v) \in E \mid u \in B, v \in B\}\). We begin with the following simple claim about the number of heavy edges.

Claim 7.2.1. The number of heavy edges, \(|H|\), is at most \(cm/2\).

Proof. The number of heavy edges is \(|H| \leq |B|(|B| - 1)/2 < |B|^2/2\). Since \(|B|b \leq 2m\), we get that \(|B| \leq \frac{2m}{b} = \frac{3m}{2\sqrt{\epsilon} - m} = \sqrt{\epsilon}m\). This gives that \(|H| \leq \frac{1}{2}|B|^2 \leq cm/2\). \(\square\)

Next, we fix an iteration \(i\) of the algorithm. Every vertex \(v\) chooses two neighbors \(w_1, w_2\). Let \(A = \{(v, w_1) \in E \mid v \in V \setminus B\}\), where \(w_1\) is the first of the two vertices chosen by the low-degree vertex \(v\). Let \(T = \{e \in E \mid e\) is a light edge in a triangle\}, and let \(A_T = T \cap A\). We say that an edge \((v, w_1) \in A_T\) is matched if \((v, w_2)\) is in the same triangle as \((v, w_1)\). If \((v, w_1) \in A_T\) is matched then \(\{v, w_1, w_2\}\) is a triangle that is detected by \(v\).

We begin with the following lemma that states that if \(G\) is \(\epsilon\)-far from being triangle-free, then in any iteration \(i\) we can bound the expected number of matched edges from below by \(\epsilon^2/8\). Let \(Y\) be the number of matched edges.

Lemma 7.2.2. The expected number of matched edges by a single iteration of the algorithm, \(E[Y]\), is greater than \(\epsilon^2/8\).

Proof. For every \(e \in A_T\), let \(Y_e\) be a random variable indicating whether \(e\) is matched. Then \(Y = \sum_{e \in A_T} Y_e\), giving the following bound:

\[ E[Y | A_T] = E[\sum_{e \in A_T} Y_e | A_T] = \sum_{e \in A_T} Pr[e \text{ is matched}] \geq |A_T|/b, \] (7.1)

where the last inequality follows because a light edge in \(A_T\) is chosen by a vertex with degree at most \(b\), hence the third triangle vertex gets picked with probability at least \(1/b\).

Next, we argue that \(E[|A_T|] \geq |T|/b\). To see why, for every edge \(e\), let \(X_e\) be a random
variable indicating whether \( e \in A \). Let \( X = \sum_{e \in T} X_e = |A_T| \). Then,

\[
E[|A_T|] = E[X] = E[\sum_{e \in T} X_e] = \sum_{e \in T} E[X_e] = \sum_{e \in T} Pr[e \in A] \geq |T|/b,
\]

(7.2)

where the last inequality follows because a light edge has at least one endpoint with degree at most \( b \). Hence, this edge gets picked by it with probability at least \( 1/b \).

It remains to bound \(|T|\) from below, for which we claim that \( |T| \geq \epsilon m/2 \). To prove this, first notice that, since \( G \) is \( \epsilon \)-far from being triangle free, it has at least \( \epsilon m \) triangle edges, since otherwise we can just remove all of them and make the graph triangle free with less than \( \epsilon m \) edge changes. By Claim 7.2.1, the number of heavy edges satisfies \( |H| \leq \epsilon m/2 \). Subtracting this from the number of triangle edges gives that at least \( \epsilon m/2 \) edges are light triangle edges, i.e.,

\[
|T| \geq \epsilon m/2.
\]

(7.3)

Finally, by Inequalities (7.1), (7.2) and (7.3), using iterated expectation we get:

\[
E[Y] = E_{A_T}[E[Y|A_T]] \geq E[\frac{|A_T|}{b}] \geq \frac{|T|}{b^2} \geq \frac{\epsilon m}{2} \cdot \frac{1}{4\epsilon^{-1} m} = \epsilon^2/8.
\]

We can now prove the correctness of our algorithm, as follows.

**Lemma 7.2.3.** If \( G \) is triangle-free then all vertices output \textit{accept} in Algorithm 9. If \( G \) is \( \epsilon \)-far from being triangle-free, then with probability at least \( 2/3 \) there exists a vertex that outputs \textit{reject}.

**Proof.** If \( G \) is triangle free then in each iteration \( v \) receives “no” from \( w_1 \) and after all iterations it returns \textit{accept}.

Assume that \( G \) is \( \epsilon \)-far from being triangle-free. Let \( Z_{i,v} \) be an indicator variable for the event that vertex \( v \) detects a triangle at iteration \( i \). First, we note that the indicators are independent, since a vertex detecting a triangle does not affect the chance of another vertex detecting a triangle (note that the graph is fixed), and the iterations are done independently. Now, let \( Z = \sum_{i=1}^{32\epsilon^{-2}} \sum_{v \in V} Z_{i,v} \), and notice that \( Z \) is the total number of detections over all iterations. Lemma 7.2.2 implies that for a fixed \( i \), it holds that \( E[\sum_{v \in V} Z_{i,v}] = E[Y] \geq \epsilon^2/8 \), so \( Z \) sums to:

\[
E[Z] = E \left[ \sum_{i=1}^{32\epsilon^{-2}} \sum_{v \in V} Z_{i,v} \right] = \sum_{i=1}^{32\epsilon^{-2}} E \left[ \sum_{v} Z_{i,v} \right] \geq \sum_{i=1}^{32\epsilon^{-2}} \epsilon^2/8 = 4.
\]

Using the Chernoff Bound from Fact 3.4.1 with \( \delta = 3/4 \) and \( \mu \geq 4 \) gives

\[
Pr[Z < 1] \leq Pr[Z < (1 - \delta)\mu] < \left( \frac{e^{-3/4}}{(1 - (3/4))(1-(3/4))} \right)^4 = 4/e^3 < 2/3,
\]

and hence with probability at least \( 2/3 \) at least one triangle is detected and the associated vertex outputs \textit{reject}, which completes the proof. \( \square \)
In every iteration, an edge needs to carry only two messages of size $O(\log n)$ bits, one sent from each vertex $v$ to $w_1$ and one sent back by $w_1$. Since there are $O(\epsilon^{-2})$ iterations, this implies that the number of rounds is $O(\epsilon^{-2})$ as well. This, together with Lemma 7.2.3, completes the proof of Theorem 7.2.

### 7.3 Distributed bipartiteness test for bounded degree graphs

In this section we show a distributed $\epsilon$-test for being bipartite for graphs with degrees bounded by $d$. Our test builds upon the sequential test of [GR99] and, as in the case of triangle freeness, takes advantage of the ability to parallelize queries. While the number of queries of the sequential test is $\Omega(\sqrt{n})$ [GR02], the number of rounds in the distributed test is only polylogarithmic in $n$ and polynomial in $\epsilon^{-1}$. As in [GR99], we assume that $d$ is a constant, and omit it from our expressions (it is implicit in the $O$ notation for $L$ below).

Let us first outline the algorithm of [GR99], since our distributed test borrows from its framework and our analysis is in part derived from it. The sequential test basically tries to detect odd cycles. It consists of $T$ iterations, in each of which a vertex $s$ is selected uniformly at random and $K$ random walks of length $L$ are performed starting from the source $s$. If, in any iteration with a chosen source $s$, there is a vertex $v$ which is reached by an even prefix of a random walk and an odd prefix of a random walk (possibly the same walk), then the algorithm rejects, as this indicates the existence of an odd cycle. Otherwise, the algorithm accepts. To obtain an $\epsilon$-test the parameters are chosen to be $T = O(\epsilon^{-1})$, $K = O(\epsilon^{-4} \sqrt{n} \log^{1/2} (n \epsilon^{-1}))$, and $L = O(\epsilon^{-8} \log^6 n)$.

The main approach of our distributed test is similar, except that a key ingredient is that we can afford to perform much fewer random walks from every vertex, namely $O(\text{poly}(\epsilon^{-1} \log n \epsilon^{-1}))$. This is because we can run random walks in parallel originating from all vertices at once. However, a crucial challenge that we need to address is that several random walks may collide on an edge, violating its congestion bound. To address this issue, our central observation is that lazy random walks (chosen to have a uniform stationary distribution) provide for a very low probability of having too many of these collisions at once. The main part of the analysis is in showing that with high probability there will never be too many walks concurrently in the same vertex, so we can comply with the congestion bound. We begin by formally defining the lazy random walks that we use.

**Definition 7.3.1.** A **lazy** random walk over a graph $G$ with degree bound $d$ is a random walk, that is, a (memory-less) sequence of random variables $Y_1, Y_2, \ldots$ taking values from the vertex set $V$, where the transition probability $\Pr[Y_k = v | Y_{k-1} = u]$ is $1/2d$ if $uv$ is an edge of $G$, $1 - \deg(u)/2d$ if $u = v$, and 0 in all other cases.

The stationary distribution for the lazy random walk of Definition 7.3.1 is uniform [Ron09, Section 8]. Next, we describe a procedure to handle one iteration of moving the random walks (Algorithm 10), followed by our distributed test for bipartiteness using lazy random walks from every vertex concurrently (Algorithm 11).

It is quite immediate that Algorithm 10 takes $O(\xi)$ communication rounds.

Our main result here is that Algorithm 11 is indeed a distributed $\epsilon$-test for bipartiteness.
Algorithm 10: Move random walks once with input $\xi$

**Variables:** $W_v$ walks residing in $v$ (multiset), $H_v$ history of walks through $v$

**Input:** $\xi$, the maximum congestion per vertex allowed

# each walk is characterized by $(i, u)$ where $i$ is the number of actual moves and $u$ is the origin vertex

1. for each vertex $v$ simultaneously
2. if $|W_v| \leq \xi$ then # give up if exceeded the maximum allowed
3. for every $(i, u)$ in $W_v$ do
4. draw next destination $w$ (according to the lazy walk scheme)
5. if $w \neq v$ then # walk exits $v$
6. send $(i + 1, u)$ to $w$
7. remove $(i, u)$ from $W_v$
8. wait until the maximum time for all other vertices to process up to $\xi$ walks
9. add the walks received by $v$ to $W_v$ and $H_v$ # walks entering $v$

Algorithm 11: Distributed bipartiteness test

**Variables:** $W_v$ walks residing in $v$ (multiset), $H_v$ history of walks through $v$

1. perform $\eta = O(\varepsilon^{-9} \log (n\varepsilon^{-1}))$ times
2. for each vertex $v$ simultaneously
3. initialize $H_v$ and $W_v$ with two copies of the walk $(0, v)$
4. perform $L = O(\varepsilon^{-8} \log^6 n)$ times
5. move walks using Algorithm 10 with input $\xi = \gamma + 2 = 3(2 \ln n + \ln L) + 2$
6. for each vertex $v$ simultaneously
7. if $H_v$ contains $(i, u)$ and $(j, u)$ for some $u$, even $i$ and odd $j$ then
8. reject (ending all operations) # odd cycle found
9. accept (for vertices that did not reject)
Theorem 7.3. There exists a distributed $\epsilon$-test in the bounded degree graph model for the property of being bipartite, that requires $O(\text{poly}(\epsilon^{-1}\log(ne^{-1})))$ rounds.

The number of communication rounds is immediate from the algorithm – it is dominated by the $L$ calls to Algorithm 10, making a total of $O(\xi L)$ rounds, which is indeed $O(\text{poly}(\epsilon^{-1}\log(ne^{-1})))$.

To prove the rest of Theorem 7.3 we need some notation, and a lemma from [GR99] that bounds from below the probabilities for detecting odd cycles if $G$ is $\epsilon$-far from being bipartite.

Given a source $s$, if there is a vertex $v$ which is reached by an even prefix of a random walk $w_i$ from $s$ and an odd prefix of a random walk $w_j$ from $s$, we say that walks $w_i$ and $w_j$ detect a violation. Let $p_s(k,\ell)$ be the probability that, out of $k$ random walks of length $\ell$ starting from $s$, there are two that detect a violation. Using this notation, $p_s(K,L)$ is the probability that the sequential algorithm outlined in the beginning rejects in an iteration in which $s$ is chosen. Since we are only interested in walks of length $L$, we denote $p_s(k) = p_s(k,L)$. A good vertex is a vertex for which this probability is bounded as follows.

Definition 7.3.2. A vertex $s$ is called good if $p_s(K) \geq 1/10$.

In [GR99] it was proved that being far from bipartite implies having many good vertices.

Lemma 7.3.3 ([GR99]). If $G$ is $\epsilon$-far from being bipartite then at least an $\epsilon/16$-fraction of the vertices are good.

In contrast to [GR99], we do not perform $K$ random walks from every vertex in each iteration, but rather only 2. Hence, what we need for our analysis is a bound on $p_s(2)$. To this end, we use $K$ as a parameter, and express $p_s(2)$ in terms of $K$ and $p_s(K)$.

Lemma 7.3.4. For every vertex $s$, $p_s(2) \geq 2p_s(K)/K(K-1)$.

Proof. Fix a source vertex $s$. For every $i,j \in [K]$, let $q_{i,j}$ be the probability of walks $w_i, w_j$ from $s$ detecting a violation. Because different walks are independent, we conclude that for every $i \neq j$ it holds that $q_{i,j} = p_s(2)$. Let $A_{i,j}$ be the event of walks $w_i, w_j$ detecting a violation. We have

$$p_s(K) = \Pr[\bigcup_{i,j} A_{i,j}] \leq \sum_{i,j} \Pr[A_{i,j}] = p_s(2)K(K-1)/2,$$

which implies that $p_s(2) \geq 2p_s(K)/K(K-1)$.

Using this relationship between $p_s(2)$ and $K$ and $p_s(K)$, we prove that our algorithm is an $\epsilon$-test. First we prove this for the random walks themselves, ignoring the possibility that Algorithm 10 will skip moving random walks due to its condition in Line 2.

Lemma 7.3.5. If $G$ is $\epsilon$-far from being bipartite, and we perform $\eta$ iterations of starting 2 random walks of length $L$ from every vertex, then the probability that no violation is detected is bounded by $1/4$.

Proof. Assume that $G$ is $\epsilon$-far from being bipartite. By Lemma 7.3.3, at least $n\epsilon/16$ vertices are good, which means that for each of these vertices $s$, $p_s(K) \geq 1/10$. This implies that
\[ \sum_{s \in V} p_s(K) \geq n \epsilon / 160. \] Now, let \( X_{i,s} \) be a random variable indicating whether there are two random walks starting at \( s \) that detect a violation. Let \( X = \sum_{i=0}^{\eta} \sum_{s \in V} X_{i,s} \). We prove that \( \Pr[X < 1] < 1/4 \). First, we bound \( E[\sum_{s \in V} X_{i,s}] \) for some fixed \( i \):

\[
E[X] = E \left[ \sum_{i=0}^{\eta} \sum_{s \in V} X_{i,s} \right] = \sum_{i=0}^{\eta} \sum_{s \in V} E[X_{i,s}]
\]

\[
= \sum_{i=0}^{\eta} \sum_{s \in V} p_s(2) \geq \sum_{i=0}^{\eta} \sum_{s \in V} \frac{2p_s(K)}{K(K-1)}
\]

\[
= \frac{2}{K(K-1)} \sum_{i=0}^{\eta} \sum_{s \in V} p_s(K) \geq \frac{2}{K(K-1)} \sum_{i=0}^{\eta} \frac{n \epsilon}{160}
\]

\[
= \frac{\eta \epsilon n}{80K(K-1)} \geq \frac{\eta \epsilon n}{80K^2}.
\]

For \( \eta = 320K^2/\epsilon n = O(e^{-9 \log(\epsilon n^{-1})}) \) it holds that \( E[X] \geq 4 \). Using the Chernoff Bound of Fact 3.4.1 with \( \delta = 3/4 \) and \( \mu \geq 4 \) gives:

\[
\Pr[X < 1] \leq \Pr[X < (1-\delta)\mu] < \left( \frac{e^{-3/4}}{(1-(3/4))(1-(3/4))} \right)^4 = \frac{4}{e^3} < 1/4,
\]

which completes the proof. \( \square \)

As explained earlier, the main hurdle on the road to prove Theorem 7.3 is in proving that the allowed congestion will not be exceeded. We prove the following general claim about the probability for \( k \) lazy random walks of length \( \ell \) from each vertex to exceed a maximum congestion factor of \( \xi \) walks allowed in each vertex at the beginning of each iteration. Here, an iteration is a sequence of rounds in which all walks are advanced by one step (whether or not they actually switch vertices).

**Lemma 7.3.6.** With probability at least \( 1 - 1/n \), running \( k \) lazy random walks of length \( \ell \) originating from every vertex will not exceed the maximum congestion factor of \( \xi = \gamma + k = 3(2 \ln n + \ln \ell) + k \) walks allowed in each vertex at the beginning of each iteration, if \( \gamma > k \).

We show below that plugging \( k = 2 \), \( \ell = L \) and \( \gamma = 3(2 \ln n + \ln L) \) in Lemma 7.3.6, together with Lemma 7.3.5, gives the correctness of Algorithm 11.

To prove Lemma 7.3.6, we argue that it is unlikely for any vertex to have more than \( k + \gamma \) walks in any iteration. Given that this is indeed the case in every iteration, the lemma follows by a union bound. We denote by \( X_{v,i} \) the random variable whose value is the number of random walks at vertex \( v \) at the beginning of the \( i \)-th iteration. That is, it is equal to the size of the set \( W_v \) in the description of the algorithm.

**Lemma 7.3.7.** For every vertex \( v \in V \) and every iteration \( i \) it holds that \( E[X_{v,i}] = k \).

**Proof.** Let us first define random variables for our walks. Enumerating our \( kn \) walks (\( k \) from each of the \( n \) vertices) arbitrarily, let \( Y^r_v, Y^r_f \ldots \) denote the sequence corresponding to the \( r \)-th walk, that is, \( Y^r_v \) is the vertex where the \( r \)-th walk is stationed at the beginning of the \( i \)-th iteration. In particular, \( X_{v,i} = |\{ r : Y^r_v = v \}|. \)
Now let us define new random variables \( Z^t_i \) in the following manner: First, we choose uniformly at random a permutation \( \sigma : [kn] \rightarrow [kn] \). Then we set \( Y_i^t = Y_i^{\sigma(t)} \) for all \( 1 \leq i \leq \ell \) and \( 1 \leq t \leq kn \). The main thing to note is that for any fixed \( t \), \( Z^t_1, Z^t_2, \ldots \) is a random walk (as it is equal to one of the random walks \( Y^t_1, Y^t_2, \ldots \)). But also, for every \( t \), \( Z^t_i \) is uniformly distributed over the vertex set of \( G \), because we started with exactly \( k \) random walks from every vertex. Additionally, since the uniform distribution is stationary for our lazy walks, this means that the unconditioned distribution of each \( Z^t_i \) is also uniform.

Now, since \( \sigma \) is a permutation, it holds that \( X_{v,i} = |\{r : Y^r_i = v\}| = |\{t : Y_i^{\sigma(t)} = v\}| = |\{t : Z^t_i = v\}| \). The expectation (by linearity of expectation) is thus \( E[X_{v,i}] = \sum_{t=1}^{kn} \Pr[Z^t_i = v] = k \).

We can now prove Lemma 7.3.6.

**Proof of Lemma 7.3.6.** We first claim that for every iteration \( i \in [\ell] \) and every vertex \( v \in V \), with probability at least \( 1 - 1/\ell n \) it holds that \( X_{v,i} \leq k + \gamma \). To show this, first fix some \( v \in V \). Let \( Z_{j,i} \) be the indicator variable for the event of the \( j \)'th walk residing at vertex \( v \) at the beginning of iteration \( i \), where \( j \in [kn] \). Then \( X_{v,i} = \sum_{j=1}^{kn} Z_{j,i} \), and the variables \( Z_{j,i} \), where \( j \in [kn] \), are all independent. We use the Chernoff Bound of Fact 3.4.1 with \( \delta = \gamma/k \geq 1 \) and \( \mu = k \) as proven in Lemma 7.3.7, obtaining:

\[
\Pr[X_{v,i} > k + \gamma] = \Pr[X_{v,i} > (\gamma/k + 1)k] < e^{-\delta \mu / 3} = e^{-\gamma / 3} = e^{-2 (\ln n + \ln \ell)} = 1/\ell n^2.
\]

Applying the union bound over all vertices \( v \in V \) and all iterations \( i \in [\ell] \), we obtain that with probability at least \( 1 - 1/n \) it holds that \( X_{v,i} \leq k + \gamma \) for all \( v \) and \( i \).

**Lemma 7.3.8.** If \( G \) is bipartite then all vertices output **accept** in Algorithm 11. If \( G \) is \( \epsilon \)-far from being bipartite, then with probability at least \( 2/3 \) there exists a vertex that outputs **reject**.

**Proof.** If \( G \) is bipartite then all vertices output **accept** in Algorithm 11, because there are no odd cycles and thus no violation detecting walks.

If \( G \) is \( \epsilon \)-far from bipartite, we use Lemma 7.3.5, in conjunction with Lemma 7.3.6 with parameters \( k = 2 \), \( \ell = L \) and \( \gamma = 3(2 \ln n + \ln L) \) as used by Algorithm 11. By a union bound the probability to accept \( G \) will be bounded by \( 1/4 + 1/n < 1/3 \) (assuming \( n > 12 \)), providing for the required bound on the rejection probability.

Lemma 7.3.8, with the communication complexity analysis of Algorithm 11, gives Theorem 7.3.

### 7.4 Distributed test for cycle-freeness

In this section, we give a distributed algorithm to test if a graph \( G \) with \( m \) edges is cycle-free or if at least \( \epsilon m \) edges have to be removed to make it so. Intuitively, in order to search for cycles, one can run a breadth-first search (BFS) and have a vertex output **reject** if two different paths reach it. The downside of this exact solution is that its running time depends on the diameter of the graph. To overcome this, a basic approach would be to run a BFS from each vertex of the graph, but for shorter distances. However, running multiple BFSs simultaneously is expensive, due to the congestion on the edges. Instead, we use a prioritization rule that drops
BFS constructions with lower priority, which makes sure that one BFS remains alive, but for this to work we need to carefully choose our prioritization.\footnote{A more involved analysis of multiple prioritized BFS executions was used in [HW12], allowing all BFS executions to fully finish in a short time without too much delay due to congestion. Since we require a much weaker guarantee, we can avoid the strong full-fledged prioritization algorithm of [HW12] and settle for a simple rule that keeps one BFS tree alive. Also, the multiple BFS construction of [LP13] does not fit our demands as it may not reach all desired vertices within the required distance, in case there are many vertices that are closer.}

Our technique consists of three parts. First, we make the graph $G$ sparser, by removing each of its edges independently with probability $\epsilon/2$. We denote the sampled graph by $G'$ and prove that if $G$ is far from being cycle-free then so is $G'$, and in particular, $G'$ contains a cycle.

Then, we run a partial BFS over $G'$ from each vertex, while prioritizing by ids: each vertex keeps only the BFS that originates in the vertex with the largest id and drops the rest of the BFSs. The length of this procedure is according to a threshold $T = 20\log n/\epsilon$. This gives detection of a cycle that is contained in a component of $G'$ with a low diameter of up to $T$, if such a cycle exists, since a surviving BFS covers the component. Such a cycle is also a cycle in $G$. If no such cycle exists in $G'$, then $G'$ has a some component with diameter larger than $T$. For large components, we take each surviving BFS that reached some vertex $v$ at a certain distance $\ell$, and from $v$ we run a new partial BFS in the original graph $G$. These BFSs are again prioritized, this time according to the distance $\ell$. Our main tool here is a claim that says that with high probability, if there is a shortest path in $G'$ of length $T/2$ between two vertices, then there is a cycle in $G$ between them of length at most $T$. This allows our BFSs on $G$ to find such a cycle.

We start with the following combinatorial lemma that shows the above claim.

**Lemma 7.4.1.** Given a graph $G$, let $G'$ be obtained by deleting each edge in $G$ with probability $\epsilon/2$, independently of other edges. Then, with probability at least $1 - 1/n^3$, every vertex $v \in G'$ that has a vertex $w \in G'$ at a distance $10\log n/\epsilon$, has a closed path passing through it in $G$, that contains a simple cycle, of length at most $20\log n/\epsilon$.

**Proof.** First, we show that for every pair $u, v$ of vertices in $G$ that are at a distance of $\ell = 10\log n/\epsilon$, one of the shortest paths between $u$ and $v$ is removed in the graph $G'$ with high probability.

For a pair of vertices $u$ and $v$ at a distance $\ell$ in $G$, the probability that a shortest path is not removed is $(1 - \epsilon/2)^\ell$, which is at most $1/n^5$. Therefore, by a union bound over all pairs of vertices, with probability at least $1 - 1/n^3$, at least one edge is removed from at least one shortest path between every pair of vertices that are at a distance of $10\log n/\epsilon$. Conditioned on this, we prove the lemma.

Now, suppose that $v$ and $w$ are two vertices in $G'$ at a distance of $10\log n/\epsilon$. Let $P'$ be this shortest path in $G'$. Suppose that $P$ is the shortest path between $v$ and $w$ in $G$. If $|P| < 10\log n/\epsilon$, then this path is no longer present in $G'$ (and thus is distinct from $P'$) and $P \cup P'$ is a closed path in $G$, passing through $v$ that has a simple cycle of length at most $20\log n/\epsilon$. If $|P| = 10\log n/\epsilon$, then there are at least two shortest paths between $v$ and $w$ in $G$ of length $10\log n/\epsilon$, the one in $G'$ and one that was removed, which we choose for $P$. Therefore, $P \cup P'$ is a closed path passing through $v$ of length at most $20\log n/\epsilon$, and hence contains a simple cycle of length at most $20\log n/\epsilon$ in it. \hfill $\square$

Next, we prove that indeed there is a high probability that $G'$ contains a cycle if $G$ is far from being cycle-free.
Claim 7.4.2. If $G$ is $\epsilon$-far from being cycle-free, then with probability at least $1 - e^{-\epsilon^2 m/32}$, $G'$ is $\epsilon/4$-far from being cycle-free.

Proof. The graph $G'$ is obtained from $G$ by deleting each edge with probability $\epsilon/2$ independently of other edges. The expected number of edges that are deleted is $\epsilon m/2$. Therefore, by the Chernoff Bound from Fact 3.4.1, the probability that at least $3\epsilon m/4$ edges are deleted is at most $\exp(-\epsilon^2 m/32)$, and the claim follows. \hfill\Box

We now describe a multiple-BFS algorithm that takes as input a length $t$ and a priority condition $P$ over vertices, and starts performing a BFS from each vertex of the graph. This is done for $t$ steps, in each of which a vertex keeps only the BFS with the highest priority while dropping the rest. Each vertex also maintains a list $L_v$ of BFSs that have passed through it. The list $L_v$ is a list of 3-tuples $(id_u, \ell, id_p)$, where $id_u$ is the id of the root of the BFS, $\ell$ is the depth of $v$ in this BFS tree and $id_p$ is the id of the parent of $v$ in the BFS tree. Initially, each vertex $v$ sets $L_v$ to include a BFS starting from itself, and then continues this BFS by sending the tuple $(id_v, 1, id_v)$ to all its neighbors, where $id_v$ is the identifier of the vertex $v$. In an intermediate step, each vertex $v$ may receive a BFS tuple from each of its neighbors. The vertex $v$ then adds these BFS tuples to the list $L_v$ and chooses one among $L_v$ according to the priority condition $P$, proceeding with the respective BFS and discontinuing the rest. Even when a BFS is discontinued, the information that the BFS reached $v$ is stored in the list $L_v$.

Algorithm 12 gives a formal description of the breadth-first search that we use in the testing algorithm for cycle-freeness.

<table>
<thead>
<tr>
<th>Algorithm 12: BFS with a priority condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Length $L$, Priority condition $P$</td>
</tr>
<tr>
<td><strong>Variables:</strong> $L_v$ list of BFS tuples passing through $v$</td>
</tr>
<tr>
<td>1 for each vertex $v$ simultaneously</td>
</tr>
<tr>
<td>2 Initialize $L_v$ to ${(id_v, 0, id_v)}$.</td>
</tr>
<tr>
<td>3 Send $(id_v, 1, id_v)$ to all neighbors of $v$.</td>
</tr>
<tr>
<td>4 perform $L$ times times</td>
</tr>
<tr>
<td>5 for each vertex $v$ simultaneously</td>
</tr>
<tr>
<td>6 if $v$ receives $(id_{u1}, \ell_1, id_{p1}), \ldots, (id_{ur}, \ell_r, id_{pr})$ from its neighbors then</td>
</tr>
<tr>
<td>7 Add $(id_{u1}, \ell_1, id_{p1}), \ldots, (id_{ur}, \ell_r, id_{pr})$ to $L_v$.</td>
</tr>
<tr>
<td>8 Select $(id_{u1}, \ell_j, id_{pj})$ from $L_v$ according to $P$ over $id_v$</td>
</tr>
<tr>
<td>9 Send $(id_{u1}, \ell_j + 1, id_v)$ to all neighbors of $v$ except $p_j$.</td>
</tr>
</tbody>
</table>

We now give more informal details of the test for cycle-freeness. By Lemma 7.4.1, we know that if there is a vertex $v$ in $G'$ that has a vertex $w$ at a distance of $T/2 = 10 \log n/\epsilon$, then there is a closed path in $G$ starting from $v$ that contains a cycle of length $20 \log n/\epsilon$. In the first part, each vertex gets its name as its vertex id, and performs a BFS on the graph $G'$ in the hope of finding a cycle. The BFS is performed using Algorithm 12, where the priority condition in the intermediate steps is selecting the BFS with the lowest origin id. If the cycle is present in a component of diameter at most $20 \log n/\epsilon$ in $G'$, then it is discovered during this BFS. To check if there is a cycle, one needs to find if there are appropriate tuples $(id_u, \ell, id_p)$ and $(id_u, \ell', id_{p'})$ in $L_v$, for some vertex $v$. 
If no cycle is discovered in this step, then we change the ids of the vertices in the following way: The id of each vertex \( v \) is now a tuple \((\ell, v)\), where \( \ell \) is the largest depth at which \( v \) occurs in a BFS tree among all the breadth-first searches that reached \( v \). We perform a BFS in \( G \) using Algorithm 12, where the priority condition is to pick the BFS whose root has the lexicographically highest id. If there is some vertex with \( \ell \geq 10 \log n/\epsilon \), then the highest priority vertex is such a vertex, and by Lemma 7.4.1, the BFS starting from that vertex will detect a cycle in \( G \).

Algorithm 13 gives a formal description of the tester for cycle-freeness.

**Algorithm 13: Cycle-freeness test**

<table>
<thead>
<tr>
<th>Variables: ( L_v ) list of BFS tuples passing through ( v ), vertex identifier id_v</th>
</tr>
</thead>
<tbody>
<tr>
<td># Construct ( G' ) by deleting edges with probability ( \epsilon/2 ).</td>
</tr>
<tr>
<td>1 for each vertex ( v ) simultaneously</td>
</tr>
<tr>
<td>2 For each neighbor ( u &lt; v ), mark the edge ( e = (u, v) \in G ) with probability ( \epsilon/2 ) for deletion.</td>
</tr>
<tr>
<td>3 Send each marked edge ( e = (u, v) ) to its corresponding ( u ).</td>
</tr>
<tr>
<td>4 Set id_v = (( v )).</td>
</tr>
<tr>
<td>5 for each vertex ( v ) simultaneously</td>
</tr>
<tr>
<td>6 Delete all edges incident on ( v ) that have been marked for deletion.</td>
</tr>
<tr>
<td># Search for cycles in small diameter components.</td>
</tr>
<tr>
<td>7 use Algorithm 12 to</td>
</tr>
<tr>
<td>8 perform BFS on ( G' ) for ( 20 \log n/\epsilon ) steps, with the priority condition being choosing the BFS with the lowest root id.</td>
</tr>
<tr>
<td>9 for each vertex ( v ) simultaneously</td>
</tr>
<tr>
<td>10 If ( L_v ) contains two tuples ((id_u, \ell, id_p)) and ((id_u, \ell', id_p')), output reject.</td>
</tr>
<tr>
<td>11 Set id_v = (( \ell_j ), ( v )) where ( \ell_j ) is the highest among all tuples ((id_u, \ell_i, id_p_i)) in ( L_v ).</td>
</tr>
<tr>
<td>12 use Algorithm 12 to</td>
</tr>
<tr>
<td>13 perform BFS on ( G ) for ( 10 \log n/\epsilon ) steps, with the priority condition being choosing the BFS with the lexicographically highest root id.</td>
</tr>
<tr>
<td>14 for each vertex ( v \in G ) simultaneously</td>
</tr>
<tr>
<td>15 If ( L_v ) contains two tuples ((id_u, \ell_j, id_p)) and ((id_u, \ell', id_p')), output reject.</td>
</tr>
<tr>
<td>16 output accept, if ( v ) did not output reject earlier.</td>
</tr>
</tbody>
</table>

We now prove the correctness of the algorithm.

**Theorem 7.4.** Algorithm 13 is a distributed \( \epsilon \)-test in the general graph model for the property of being cycle-free, that requires \( O(\log n/\epsilon) \) rounds.

**Proof.** Notice that a vertex in Algorithm 13 outputs reject only when it detects a cycle. Therefore, if \( G \) is cycle-free, then every vertex outputs accept with probability 1.

Suppose that \( G \) is \( \epsilon \)-far from being cycle-free. Notice that, with probability at least \( 1 - 1/n^3 \), the assertion of Lemma 7.4.1 holds. Furthermore, from Claim 7.4.2, we know that \( G' \) is \( \epsilon/4 \)-far from being cycle-free, with probability \( 1 - e^{-v^2m/32} \), and hence contains at least one cycle. This cycle could be in a component of diameter less than \( 20 \log n/\epsilon \), or it could be in a component of diameter at least \( 20 \log n/\epsilon \) in \( G' \). We analyse the two cases separately.

Suppose there is a cycle in a component \( C \) of \( G' \) of diameter at most \( 20 \log n/\epsilon \). Let \( u \) be the vertex with the smallest id in \( C \). In Algorithm 13, the BFS starting at \( u \) is always propagated at any intermediate vertex due to the priority condition. Furthermore, since the diameter of \( C \) is
at most $20 \log n/\epsilon$, this BFS reaches all vertices of $C$. Hence, this BFS detects the cycle and at least one vertex in $C$ outputs reject.

On the other hand, if the cycle is present in a component in $G$ of diameter at least $20 \log n/\epsilon$, then after Step 11 of the algorithm, each vertex $v$ gets the length of the longest path from the origin, among all the BFSs that reached $v$, as the first component of its id. The vertex $v$ that gets the lexicographically highest id in the component has a vertex $w$ that is at least $10 \log n/\epsilon$ away in $G'$, since the radius of the component is at least half the diameter. Therefore, by Lemma 7.4.1, it is part of a cycle of length at most $20 \log n/\epsilon$ in $G$. Hence, the vertex with the highest priority in the BFS on $G$ is a vertex $u$ that has a vertex at a distance of at least $10 \log n/\epsilon$ in $G'$, and there is a closed path through $u$ that contain a simple cycle of length at most $20 \log n/\epsilon$. At least one vertex on this simple cycle will output reject when Algorithm 13 is run on $G$.

The number of rounds is $O(\log n/\epsilon)$, since Algorithm 13 performs two breadth-first searches in the graph with this number of rounds.

### 7.5 Lower bounds for testing bipartiteness and cycle-freeness

In this section, we prove that any distributed algorithm for $\epsilon$-testing bipartiteness or cycle-freeness in bounded-degree graphs requires $\Omega(\log n)$ rounds of communication\(^3\). We construct bounded-degree graphs that are $\epsilon$-far from being bipartite, such that all cycles are of length $\Omega(\log n)$. We argue that any distributed algorithm that runs in $O(\log n)$ rounds does not detect a witness for non-bipartiteness. We also show that the same construction proves that every distributed algorithm for $\epsilon$-testing cycle-freeness requires $\Omega(\log n)$ rounds of communication. Formally, we prove the following theorem.

**Theorem 7.5.** Any distributed $1/100$-test for the property of being bipartite requires $\Omega(\log n)$ rounds of communication.

To prove Theorem 7.5, we show the existence of a graph $G'$ that is far from being bipartite, but all of its cycles are of at least logarithmic length. Since in $T$ rounds of a distributed algorithm, the output of every vertex cannot depend on vertices that are at distance greater than $T$ from it, no vertex can detect a cycle in $G'$ in less than $O(\log n)$ rounds, which proves Theorem 7.5. To prove the existence of $G'$ we use the probabilistic method with alterations, and prove the following.

**Lemma 7.5.1.** Let $G$ be a random graph on $n$ vertices where each edge is present with probability $1000/n$. Let $G'$ be obtained by removing all edges incident with vertices of degree greater than 2000, and one edge from each cycle of length at most $\log n/\log 1000$. Then, with probability at least $1/2 - e^{-100} - e^{-n}$, $G'$ is $1/100$-far from being bipartite.

Since a graph that is $\epsilon$-far from being bipartite is also $\epsilon$-far from being cycle-free, we immediately obtain the same lower bound for testing cycle-freeness, as follows.

\(^3\)Our lower bound applies even to the less restricted LOCAL model of communication, which does not limit the size of the messages.
Theorem 7.6. Any distributed 1/100-test for the property of being cycle-free requires $\Omega(\log n)$ rounds of communication.

The rest of this section is devoted to proving Lemma 7.5.1. We need to show three properties of $G'$: (a) that it is far from being bipartite, (b) that it does not have small cycles, and (c) that its maximum degree is bounded. We begin with the following definition, which is similar in spirit to being far from satisfying a property and which will assist us in our proof.

**Definition 7.5.2.** A graph $G$ is $k$-removed from being bipartite if at least $k$ edges have to be removed from $G$ to make it bipartite.

Note that a graph $G$ with maximum degree $d$ is $\varepsilon$-far from being bipartite if it is $\varepsilon dn$-removed from being bipartite.

Let $G$ be a random graph on $n$ vertices where for each pair of vertices, an edge is present with probability $1000/n$, independently of other pairs. The expected number of edges in the graph is $500(n-1)$. Since the edges are sampled independently with probability $1000/n$, by the Chernoff Bound from Fact 3.4.1, with probability at least $1 - e^{-10n}$ the graph has at least $400n$ edges. We now show that $G$ is far from being bipartite, with high probability.

**Lemma 7.5.3 (far from being bipartite).** With probability at least $1 - e^{-199n}$, $G$ is $20n$-far from being bipartite.

**Proof.** Fix a bipartition $(L,R)$ of the vertex set of $G$ such that $|L| \geq n/2$. For each pair of vertices $u, v \in L$, let $X_{u,v}$ be a random variable which is 1 if the edge $(u, v)$ is present in $G$ and 0 otherwise. Its expected value is $E[X_{u,v}] = 1000/n$. The random variable $X = \sum_{u,v \in L} X_{u,v}$ counts the number of edges within $L$. By the linearity of expectation, $E[X] \geq \left(\frac{n}{2}\right)1000/n \geq 30n$. Since the random variables $X_{u,v}$ are independent, by the Chernoff Bound from Fact 3.4.1, we have that $\Pr[X < 20n] \leq \exp(-200n)$. Therefore, with probability at least $1 - \exp(-200n)$, there are at least $20n$ edges within $L$. The total number of such bipartitions of $G$ is at most $2^{n-1}$. Taking a union bound over all such bipartitions, the probability that at least one of the bipartitions contains less than $20n$ edges within its $L$ side is at most $\exp(-199n)$, and the lemma follows.

The expected degree of a vertex $v$ in $G$ is $1000(1 - 1/n)$. Therefore, by the Chernoff Bound from Fact 3.4.1, the probability that the degree of $v$ is greater than 2000 is at most $\exp(-300(1 - 1/n))$. We now show that, with sufficiently high probability, the number of edges that are incident with high degree vertices is small. We can remove all such edges to obtain a bounded-degree graph that is still far from being bipartite.

**Lemma 7.5.4 (mostly bounded degrees).** With probability at least $1 - e^{-100}$, there are at most $n$ edges that are incident with vertices of degree greater than 2000 in $G$.

**Proof.** For a pair $u, v$ of vertices, the probability that there is an edge between them and that one of $u$ or $v$ is of degree greater than 2000 is $\Pr[(u, v) \in E] \cdot \Pr[u$ or $v$ has degree $\geq 2000|(u, v) \in E]$. This is at most $(1000/n) \cdot 2 \cdot \exp(-300(1 - 1/n))$. Therefore, the expected number of edges that are incident with a vertex of degree greater than 2000 is at most $1000n \cdot \exp(-300(1 - 1/n))$.  

93
By Markov’s inequality, the probability that there are at least $n$ edges that are incident with vertices of degree greater than 2000 is at most $1000 \cdot \exp(-300(1 - 1/n))$. This completes the proof of the lemma.

We now bound the number of cycles of length at most $O(\log n)$ in the graph $G$.

**Lemma 7.5.5 (few small cycles).** With probability at least $1/2$, there are at most $2n$ cycles of length at most $\log n / \log 1000$ in $G$.

*Proof.* For any $k$ fixed vertices, the probability that there is a cycle among the $k$ vertices is at most $k!(1000/n)^k$. Therefore the expected number of cycles in $G$ of length at most $k$ is at most $1000^k$. For $k = \log n / \log 1000$, this means that the expected number of cycles in $G$ of length at most $\log n / \log 1000$ is $n$. Therefore, with probability at least $1/2$ there are at most $2n$ cycles of length at most $\log n / \log 1000$ in $G$. 

We are now ready to prove Lemma 7.5.1, completing our lower bounds. Intuitively, since $G$ does not contain many high degree vertices and many small cycles, removing them to obtain $G'$ only changes the distance from being bipartite by a small term.

*Proof.* With probability $1 - e^{-n}$, there are at least $400n$ edges in $G$ and by Lemma 7.5.3 $G$ is $20n$-removed from being bipartite. By Lemma 7.5.4, with probability at least $1 - e^{-100}$, there are at most $n$ edges incident with vertices of degree greater than 2000 and by Lemma 7.5.5 with probability at least $1/2$ there are at most $2n$ cycles of length at most $\log n / \log 1000$. Hence, with probability at least $1/2 - e^{-100} - e^{-n}$, $G'$ is a graph with degree at most 2000, that is $17n$-removed from being bipartite. Therefore, $G'$ is $1/100$-far from being bipartite. 

}$
Chapter 8

Derandomizing local distributed algorithms

In this chapter we address a cornerstone family of problems in distributed computing are the so-called local problems. These include finding a maximal independent set (MIS), a $(\Delta + 1)$-coloring where $\Delta$ is the maximal degree in the network graph, finding a maximal matching, constructing multiplicative spanners, and more. Intuitively, as opposed to global problems, local problems admit solutions that do not require communication over the entire network graph.

One fundamental characteristic of distributed algorithms for local problems is whether they are deterministic or randomized. Currently, there exists a curious gap between the known complexities of randomized and deterministic solutions for local problems. Interestingly, the main indistinguishability-based technique used for obtaining the relatively few lower bounds that are known seems unsuitable for separating these cases.

Our main contribution is in providing tools for derandomizing solutions to local problems, when the $n$ nodes can only send $O(\log n)$-bit messages in each round of communication. We combine bounded independence, which we show to be sufficient for some algorithms, with the method of conditional expectations and with additional machinery, to obtain the following results.

Our techniques give a deterministic maximal independent set (MIS) algorithm in the CONGEST model, where the communication graph is identical to the input graph, in $O(D \log^2 n)$ rounds, where $D$ is the diameter of the graph. The best known running time in terms of $n$ alone is $2^{O(\sqrt{\log n})}$, which is super-polylogarithmic, and requires large messages. For the CONGEST model, the only known previous solution is a coloring-based $O(\Delta + \log^* n)$-round algorithm, where $\Delta$ is the maximal degree in the graph. To the best of our knowledge, ours is the first deterministic MIS algorithm for the CONGEST model, which for polylogarithmic values of $D$ is only a polylogarithmic factor off compared with its randomized counterparts.

On the way to obtaining the above, we show that in the Congested Clique model, which allows all-to-all communication, there is a deterministic MIS algorithm that runs in $O(\log \Delta \log n)$ rounds, where $\Delta = O(n^{1/3})$, the bound improves to $O(\log \Delta)$ and holds also for $(\Delta + 1)$-coloring.

In addition, we deterministically construct a $(2k - 1)$-spanner with $O(kn^{1+1/k} \log n)$ edges in $O(k \log n)$ rounds. For comparison, in the more stringent CONGEST model, the best deterministic algorithm for constructing a $(2k - 1)$-spanner with $O(kn^{1+1/k})$ edges runs in $O(n^{1-1/k})$ rounds.
8.1 Deterministic MIS

We consider the following modification of the randomized algorithm of Ghaffari [Gha16], where the constants are slightly changed. The algorithm of Ghaffari consists of two parts. The first part (shown to have a good local complexity) consists of $O(\log \Delta)$ phases, each with $O(1)$ rounds. After this first part, it is shown that sufficiently many nodes are removed from the graph. The MIS for what remains is computed in the second part deterministically in time $O(2\sqrt{\log \log n})$.

The modification to the first part of Ghaffari’s MIS Algorithm
Set $p_0(v) = 1/4$.

\[
p_{t+1}(v) = \begin{cases} 
1/2 \cdot p_t(v), & \text{if } d_t(v) \geq 1/2 \\
\min\{2p_t(v), 1/4\}, & \text{if } d_t(v) < 1/2,
\end{cases}
\]

where $d_t(v) = \sum_{u \in N(v)} p_t(u)$ is the effective degree of node $v$ in phase $t$. In each phase $t$, the node $v$ gets marked with probability $p_t(v)$ and if none of its neighbors is marked, $v$ joins the MIS and gets removed along with its neighbors.

8.1.1 An $O(\log \Delta)$ round randomized MIS algorithm in the congested clique

We begin by observing that in the congested clique, what remains after $O(\log \Delta)$ phases of Ghaffari’s algorithm can be solved in $O(1)$ rounds. This provides an improved randomized runtime compared to [Gha16], and specifically, has no dependence on $n$.

The algorithm consists of two parts. In the first part, we run Ghaffari’s algorithm for $O(\log \Delta)$ phases. We emphasize that this works with both Ghaffari’s algorithm and with our modified Ghaffari’s algorithm, since the values of the constants do not affect the running time and correctness of the randomized first part of the algorithm.

Then, in the second part, a leader collects all surviving edges and solves the remaining MIS deterministically on that subgraph. We show that the total number of edges incident to these nodes is $O(n)$ w.h.p., and hence using the deterministic routing algorithm of Lenzen [Len13], the second part can be completed in $O(1)$ rounds w.h.p.

**Lemma 8.1.1** ([Gha16, Theorem 3.1]). For every $\epsilon \in (0, 1)$, there exists a constant $c'$, such that for each node $v$, the probability that $v$ has not made its decision within the first $c' \cdot (\log \Delta + \log 1/\epsilon)$ phases is at most $\epsilon$.

Since the decision whether to join the MIS or to be removed by having a neighbor in the MIS depends only on the 2-neighborhood of a node, decisions made by nodes that are at least 4 hops from each other are independent. We make use of the following variant of Chernoff’s bound.

A Chernoff bound with bounded dependency [Pem01] 8.1.2. Let $X_1, \ldots, X_n$ denote a set of binary random variables with bounded dependency $\hat{d}$, and let $\mu = E(\sum_i X_i)$. Then:

\[
\Pr[\sum_i X_i \geq (1 + \delta)\mu] \leq O(\hat{d}) \cdot e^{-\Theta(\delta^2\mu/\hat{d})}.
\]
Corollary 8.1. After $O(\log \Delta)$ phases, the number of edges incident to nodes that have not made their decisions is $O(n)$.

Proof. By Lemma 8.1.1, there is a constant $c$, such that the probability that a node has not made its decision after $O(\log \Delta)$ phases is at most $1/\Delta^c$. Letting $X$ denote the random variable of the number of nodes that have not made their decision after $O(\log \Delta)$ phases gives that

$$\mu = \mathbb{E}[X] = n/\Delta^c.$$ 

Each node is mutually independent from all nodes except in its 4-neighborhood, in which there are up to $\Delta^4$ nodes. By using Fact 8.1.2 with $\delta = \Theta(\Delta^{-c})$ and $\hat{d} = \Delta^4$, and taking $c$ to be large enough, we get that

$$\Pr[X \geq n/\Delta] \leq O(\Delta^4) \cdot e^{-\Theta(n)}.$$ 

Hence, w.h.p., there are $O(n/\Delta)$ remaining nodes and therefore the number of their incident edges is at most $O(n)$.

We conclude:

Theorem 8.2. There is a randomized algorithm that computes MIS in the congested clique model within $O(\log \Delta)$ rounds with high probability.

Note that the proof of Corollary 8.1 cannot be extended to the case of pairwise independence, which is needed for derandomization, since the concentration guarantees are rather weak. For this, we need to develop in the following section new machinery.

8.1.2 Derandomizing the modified MIS algorithm

We first turn to consider the modified Ghaffari’s algorithm when the random decisions made by the nodes are only pairwise independent.

Ghaffari’s algorithm with pairwise independence We review the main terminology and notation from [Gha16], up to our modification of constants. Recall that $d_t(v) = \sum_{u \in N(v)} p_t(u)$. A node $v$ is called light if $d_t(v) < 1/4$.

Golden phases and golden nodes. We define two types of golden phases for a node $v$. This definition is a modification of the corresponding definitions in [Gha16].

Definition 8.1.3.

Type-1 golden phase: $p_t(v) = 1/4$ and $d_t(v) \leq 1/2$;

Type-2 golden phase: $d_t(v) > 1/4$ and at least $d_t(v)/10$ of it arises from light nodes.

A node $v$ is called golden in phase $t$, if phase $t$ is a golden phase for $v$ (of either type). Intuitively, a node $v$ that is golden in phase $t$ is shown to have a constant probability of being removed. Specifically, in a golden phase of type-1, $v$ has a constant probability to join the MIS and in a golden phase of type-2, there is a constant probability that $v$ has a neighbor that joins the MIS and hence $v$ is removed.
We now prove the analogue of Lemma 3.3 in [Gha16] for the setting in which the coin flips made by the nodes are not completely independent but are only pairwise independent. We show that a golden node for phase \( t \) is still removed with constant probability even under this weaker bounded independence guarantee.

**Lemma 8.1.4** (Type-1 golden nodes with pairwise independence). Consider the modified Ghaf-fari’s algorithm with pairwise independent coin flips. If \( t \) is a type-1 golden phase for a node \( v \), then \( v \) joins the MIS in phase \( t \) with probability at least \( 1/8 \).

**Proof.** In each type-1 golden phase, \( v \) gets marked with probability \( p_t(v) = 1/4 \). By the inclusion-exclusion principle, the probability that \( v \) is marked but none of its neighbors is marked in phase \( t \) can be bounded by:

\[
\Pr[\text{\( v \) \text{is the only marked node in } N(v)\}] \geq p_t(v) - \sum_{u \in N(v)} p_t(v) \cdot p_t(u) \\
\geq p_t(v)(1 - d_t(v)) \geq 1/4(1 - 1/2) = 1/8.
\]

Hence, \( v \) joins the MIS with probability at least \( 1/8 \). \( \square \)

We next show that also the golden nodes of type-2 are removed with constant probability assuming only pairwise independence.

**Lemma 8.1.5** (Type-2 golden nodes with pairwise independence). Consider the modified Ghaf-fari’s algorithm with pairwise independent coin flips. If \( t \) is a type-2 golden phase for a node \( v \), then \( v \) is removed in phase \( t \) with probability at least \( \alpha = 1/160 \).

**Proof.** For node \( v \), fix a subset of light neighbors \( W(v) \subseteq N(v) \) satisfying that \( \sum_{w \in W(v)} p_t(u) \in [1/40, 1/4] \). Such a set exists because by the definition of a type-2 golden phase, the sum of probabilities of light neighbors of \( v \) is at least \( 1/40 \), and every single probability is at most \( 1/4 \) by the constants taken in the algorithm (this probability either halves in each phase, or is bounded from above by \( 1/4 \)).

For \( u \in W(v) \), let \( T_t(v, u) \) denote the indicator variable of the event that in phase \( t \) the following happens: \( u \) gets marked, none of \( u \)'s neighbors get marked and \( u \) is the only neighbor of \( v \) in \( W(v) \) that got marked. For node \( u, v \) and phase \( t \), let \( m_{u,v,t} \) be the indicator random variable that both \( u \) and \( v \) get marked in phase \( t \) due to the pairwise independence, we have:

\[
\Pr[m_{u,v,t} = 1] = p_t(u) \cdot p_t(v).
\]

Hence, the probability of the event indicated by \( T_t(v, u) \) can be bounded by:

\[
\Pr[T_t(v, u) = 1] \geq p_t(u) - \sum_{w \in N(u)} \Pr[m_{u,w,t} = 1] - \sum_{u' \in W \setminus \{u\}} \Pr[m_{u,w,t} = 1] \\
= p_t(u) - \sum_{w \in N(u)} p_t(u)p_t(w) - \sum_{u' \in W \setminus \{u\}} p_t(u)p_t(u') \\
\geq p_t(u) \left( 1 - \sum_{w \in N(u)} p_t(w) - \sum_{u' \in W \setminus \{u\}} p_t(u') \right) \\
\geq p_t(u) \left( 1 - d_t(u) - 1/4 \right) \geq p_t(u) \left( 1 - 1/2 - 1/4 \right) = 1/4 \cdot p_t(u).
\]

98
Since the events indicated by $\Upsilon_t(v,u)$ are mutually exclusive for different $u,u' \in N(v)$, it holds that the probability that $v$ gets removed is at least

$$\Pr[v \text{ is removed in phase } t] \geq \sum_{u \in W(v)} \Pr[\Upsilon_t(v,u) = 1] \geq 1/4 \cdot \sum_{u \in W(v)} p_t(u) \geq 1/160.$$  

Finally, we claim the analogue of Lemma 3.2 in [Gha16].

**Lemma 8.1.6.** Consider the modified Ghaffari’s algorithm with pairwise independent randomness and $\epsilon = 1/\Delta^c$. For a large enough $c$, for every $v$, at the end of $c' \cdot \log \Delta$ phases, either $v$ has joined the MIS, or it has a neighbor in the MIS, or at least one of its golden phase counts reached $c \cdot \log \Delta$.

The proof here is exactly that same as in [Gha16]. The reason is that the proof does not assume independence and is only affected by the update rule of the probabilities. Note that similarly to [Gha16], we had to define type-2 with a threshold on $d_t(v)$ which is factor 2 smaller than that of type-1. As a result, the following holds in the pairwise independence setting:

**Lemma 8.1.7.** Within $O(\log \Delta)$ phases, every node remains with probability at most $1/\Delta$.

Recall that the proof of Lemma 8.1 cannot be extended to pairwise independence since the concentration guarantees are rather weak. Our algorithm will use pairwise independence but with some crucial modifications required in order to guarantee that after $O(\log \Delta)$ phases, only $O(n/\Delta)$ nodes remain undecided.

**Deterministic $O(\log \Delta \log n)$-Round Algorithm in the Congested Clique** We now turn to consider the derandomization procedure. We show the following:

**Theorem 8.3.** There is a deterministic MIS algorithm for the broadcast congested clique model that completes in $O(\log \Delta \log n)$ rounds.

**The challenge.** Consider phase $t$ in the modified Ghaffari’s algorithm and let $V_t$ be the set of golden nodes in this phase. Our goal is to select additional nodes into the MIS so that at least a constant fraction of the golden nodes are removed. Let $v_1,\ldots,v_{n'}$ be the nodes that are not removed in phase $t$. For each node, define corresponding random variables $x_1,\ldots,x_{n'}$ indicating whether $v_i$ is marked in phase $t$ or not. Let $X_i = (x_1 = b_1,\ldots,x_i = b_i)$ define a partial assignment for the nodes $v_1,\ldots,v_i$ (i.e., whether or not they are in the MIS in phase $t$). Let $X_0 = \emptyset$ denote the case where none of the decisions is fixed.

For a golden node $v$, let $r_{v,t}$ be the random variable indicating whether $v$ gets removed in phase $t$, and let $R_t$ be the random variable of the number of removed golden nodes. By linearity of expectation, $E(R_t) = \sum_v E(r_{v,t})$ is the expected number of removed golden nodes in phase $t$. By Lemmas 8.1.4 and 8.1.5, there is a constant $c$ such that $E(R_t) \geq c \cdot |V_t|$.

Potentially, we could aim for the following: Given the partial assignment $X_i$, compute the two expectations of the number of removed golden nodes conditioned on the two possible assignments for $x_{i+1}$, $E(R_t \mid X_i, x_{i+1} = 0)$ and $E(R_t \mid X_i, x_{i+1} = 1)$, and choose $x_{i+1}$ according to the larger expectation.

However, towards the above goal we face the following main challenges.
The value of $R_t$ cannot be easily computed, since when using probabilities of neighboring nodes we might be double-counting: a node might be removed while having more than single neighbor that joins the MIS.

The search space of size $2^n$ is too large and in particular, the conditional expectation computation consists of $n$ steps.

Even when using pairwise independence to enjoy a $O(\log n)$-bit seed, searching a good point in a space of size $O(\text{poly } n)$ in a brute force manner cannot be done efficiently in the congested clique.

Despite our proof that golden nodes are removed with constant probability even with pairwise independence, it is still not clear how to implement the second part of the MIS algorithm, because showing that only $O(n/\Delta)$ nodes survive cannot be done with pairwise independence. That is, the proof of Corollary 8.1 inherently needs full independence.

Addressing (C4) requires the use of a priority-based scheme for choosing the nodes that join the MIS, which requires a novel age-based weighting approach to be added to the MIS algorithm. Next, we describe our main derandomization tools and then provide our algorithm.

**Derandomization tools.** We define a pessimistic estimator to the conditional expectation $E(R_t \mid X_i)$, which can be computed efficiently in our model. Then, we describe how to reduce the search space using pairwise independence. In our algorithm, the nodes will apply the method of conditional expectations on the estimator in order to find a “good” seed of length $O(\log n)$.

**Tool 1: The pessimistic estimator function.** Consider phase $t$ and recall that $V_t$ are the golden nodes in this phase. Similarly to the clever approach of [Lub86, Lub93], we define a variable $\psi_{v,t}$ that will satisfy that $r_{v,t} \geq \psi_{v,t}$. The idea is to account for a removed node of type-2 only in the case that it is removed because a single one of its neighbors joins the MIS. Since this can only occur for one of its neighbors, we avoid double-counting when computing the probabilities. This allows us to cope with challenge (C1).

Let $m_{v,t}$ be the random variable indicating the event that $v$ is marked. Let $m_{u,v,t}$ indicate the event that both $u$ and $v$ are marked. Define

$$
\psi_{v,t} = \begin{cases} 
m_{v,t} - \sum_{u \in N(v)} m_{v,u,t}, & \text{if } v \text{ is of type-1,} \\
\sum_{u \in N(v)} \left( m_{v,u,t} - \sum_{w \in N(u)} m_{u,w,t} - \sum_{w' \in N(v) \setminus \{u\}} m_{u,w',t} \right), & \text{if } v \text{ is of type-2.} 
\end{cases}
$$

Denoting $\Psi_t = \sum_{v \in V_t} \psi_{v,t}$ gives that $\Psi_t$ is a lower bound on the number of removed golden nodes, i.e., $\Psi_t \leq R_t$. For a partial assignment $X_i = (x_1 = b_1, \ldots, x_i = b_i)$ indicating which of the
nodes are in the MIS, we have

\[
\mathbb{E}(\psi_{v,t} \mid X_i) = \begin{cases} 
\Pr[m_{v,t} = 1 \mid X_i] - \sum_{u \in N(v)} \Pr[m_{v,u,t} \mid X_i], & \text{if } v \text{ is of type-1.} \\
\sum_{u \in N(v)} (\Pr[m_{v,u,t} = 1 \mid X_i] - \sum_{w \in N(u)} \Pr[m_{u,w,t} = 1 \mid X_i] - \sum_{w' \in W(v) \setminus \{u\}} \Pr[m_{u,w',t} = 1 \mid X_i]), & \text{if } v \text{ is of type-2,}
\end{cases}
\]  

(8.1)

where \( W(v) \subseteq N(v) \) is a subset of \( v \)'s neighbors satisfying that \( \sum_{w \in W(v)} p_t(w) \in [1/40, 1/4] \) (as used in the proof of Lemma 8.1.5). By Lemmas 8.1.4 and 8.1.5, it holds that \( \mathbb{E}(\psi_{v,t}) \geq \alpha \) for \( v \in V_t \). Hence, we have that:

\[ \mathbb{E}(r_{v,t}) \geq \mathbb{E}(\psi_{v,t}) \geq \alpha. \]

Since \( r_{v,t} \geq \psi_{v,t} \) even upon conditioning on the partial assignment \( X_i \), we get:

\[ \mathbb{E}(R_{v,t} \mid X_i) \geq \mathbb{E}(\Psi_t \mid X_i) = \sum_{v \in V_t} \mathbb{E}(\psi_{v,t} \mid X_i) \geq \alpha \cdot |V_t|. \]

Our algorithm will employ the method of conditional expectations on a \textit{weighted} version of \( \mathbb{E}(\Psi_t \mid X_i) \), as will be discussed later.

\section*{Tool 2: Pairwise independence.}

We now combine the method of conditional expectations with a small search space. We use Lemma 3.3.2 with \( d = 2 \), \( \gamma = \Theta(\log n) \) and a prime number \( \beta = O(\log \Delta) \). This is because we need the marking probability, \( p_t(v) \), to be \( \Omega(1/\text{poly } \Delta) \).

Consider phase \( t \). Using the explicit construction of Lemma 3.3.2, if all nodes are given a shared random seed of length \( \gamma \), they can sample a random hash function \( h : \{0, 1\}^\gamma \rightarrow \{0, 1\}^\beta \) from \( \mathcal{H}_{\gamma, \beta} \) which yields \( n \) pairwise independent choices. Specifically, flipping a biased coin with probability \( p_t(v) \) can be trivially simulated using the hash value \( h(ID_v) \) where \( ID_v \) is a \( O(\log n) \)-bit ID of \( v \).\(^1\) Since \( h \) is a random function in the family, all random choices are pairwise independent and the analysis of the golden phases goes through. This standard approach takes care of challenge (C2).

Even though using a seed of length \( O(\log n) \) reduces the search space to be of polynomial size, still, exploring all possible \( 2^{O(\log n)} = O(n^c) \) seeds in a brute force manner is too time consuming. Instead, we employ the method of conditional expectations to find a \textit{good seed}. That is, we will consider \( \mathbb{E}(\Psi_t \mid Y_i) \) where \( Y_i = (y_1 = b_1, \ldots, y_i = b_i) \) is a partial assignment to the seed \( Y = (y_1, \ldots, y_a) \). The crux here is that since a \textit{random seed} is good, then so is the expectation over seeds that are sampled uniformly at random. Hence, the method of conditional expectations will find a seed that is at least as good as the random selection. Specifically, we still use the pessimistic estimator of Equation (8.1), but we condition on the small seed \( Y_i \) rather than on \( X_i \). This addresses challenge (C3).

\(^1\)Flipping a biased coin with probability \( 1/2^i \), is the same as getting a uniformly distributed number \( y \) in \([1, b]\) and outputting 1 if and only if \( y \in [1, 2^{b-i}] \).
Tool 3: An age-based weighted adaptation. To handle challenge (C4), we compute the expectation of a weighted version of $\Psi_t$, which favors old nodes where the age of a node is counted as the number of golden phases it experienced. Let $age(v)$ be the number of golden phases $v$ has till phase $t$ and recall that a golden node is removed with probability at least $\alpha$. Define $\psi'_{v,t} = (1/(1-\alpha))^ {age(v)} \cdot \psi_{v,t}$, and $\Psi'_t = \sum_{v \in V_t} \psi'_{v,t}$. We use the method of conditional expectations for:

$$E(\Psi'_t \mid Y_i) = \sum_{v \in V_t} E(\psi'_{v,t} \mid Y_i),$$

rather than for $E(\Psi'_t \mid Y_i)$. The choice of this function will be made clear in the proof of Lemma 8.1.8.

Algorithm Description. The first part of the algorithm consists of $\Theta(\log \Delta)$ phases, where in phase $t$, we derandomize phase $t$ in the modified Ghaffari’s algorithm using $O(\log n)$ deterministic rounds. In the second part, all nodes that remain undecided after the first part, send their edges to the leader using the deterministic routing algorithm of Lenzen. The leader then solves locally and notifies the relevant nodes to join the MIS. In the analysis section, we show that after the first part, only $O(n/\Delta)$ nodes remain undecided, and hence the second part can be implemented in $O(1)$ rounds.

From now on we focus on the first part. Consider phase $t$ in the modified Ghaffari’s algorithm. Note that at phase $t$, some of the nodes are already removed from the graph (either because they are part of the MIS or because they have a neighbor in the MIS). Hence, when we refer to nodes or neighboring nodes, we refer to the remaining graph induced on the undecided nodes.

Let $Y = (y_1, \ldots, y_\gamma)$ be the $\gamma$ random variables that are used to select a hash function and hence induce a deterministic algorithm. We now describe how to compute the value of $y_i$ in the seed, given that we already computed $y_1 = b_1, \ldots, y_{i-1} = b_{i-1}$. By exchanging IDs (of $\Theta(\log n)$ bits), as well as the values $p_t(v)$ and $d_t(v)$ with its neighbors, a node can check if it is a golden type-1 or type-2 node according to the conditions of Definition 8.1.3. In addition, every node maintains a counter, $age(v)$ referred to as the age of $v$, which measures the number of golden phases it had so far.

Depending on whether the node $v$ is a golden type-1 or type-2 node, based on Equation (8.1), it computes a lower bound on the conditional probability that it is removed given the partial seed assignment $Y_{i,b} = (y_1, \ldots, y_i = b)$ for every $b \in \{0,1\}$. These lower bound values are computed according to the proofs of Lemmas 8.1.4 and 8.1.5.

Specifically, a golden node $v$ of type-1, uses the IDs of its neighbors and their $p_t(u)$ values to compute the following:

$$E(\psi_{v,t} \mid Y_{i,b}) = Pr[m_{v,t} = 1 \mid Y_{i,b}] - \sum_{u \in N(v)} Pr[m_{u,t} = 1 \mid Y_{i,b}],$$

where $Pr[m_{v,t} = 1 \mid Y_{i,b}]$ is the conditional probability that $v$ is marked in phase $t$.

For a golden node $v$ of type-2 the lower bound is computed differently. First, $v$ defines a subset of neighbors $W(v) \subseteq N(v)$, satisfying that $\sum_{w \in W(v)} p_t(w) \in [1/40, 1/4]$, as in the proof of Lemma 8.1.5. Let $M_{t,b}(u)$ be the conditional probability on $Y_{i,b}$ that $u$ is marked but none of
its neighbors are marked. Let $M_{t,b}(u, W(v))$ be the conditional probability on $Y_{i,b}$ that another node other than $u$ is marked in $W(v)$. By exchanging the values $M_{t,b}(u)$, $v$ computes:

$$
\mathbb{E}(\psi_{v,t} \mid Y_{i,b}) = \sum_{u \in W(v)} \Pr[m_{u,t} = 1 \mid Y_{i,b}] - M_{t,b}(u) - M_{t,b}(u, W(v)).
$$

Finally, as in Equation (8.2), the node sends to the leader the values $\mathbb{E}(\psi_{v,t} \mid Y_{i,b}) = 1/(1 - \alpha)^{age(v)} \cdot \mathbb{E}(\psi_{v,t} \mid Y_{i,b})$ for $b \in \{0, 1\}$. The leader computes the sum of the $\mathbb{E}(\psi_{v,t} \mid Y_{i,b})$ values of all golden nodes $V_t$, and declares that $y_t = 0$ if $\sum_{v \in V_t} \mathbb{E}(\psi_{v,t} \mid Y_{i,b}) \geq \sum_{v \in V_t} \mathbb{E}(\psi_{v,t} \mid Y_{i,b})$, and $y_t = 1$ otherwise. This completes the description of computing the seed $Y$.

Once the nodes compute $Y$, they can simulate phase $t$ of the modified Ghaffari’s algorithm. In particular, the seed $Y$ defines a hash function $h \in \mathcal{H}_{\gamma, \beta}$ and $h(ID(v))$ can be used to simulate the random choice with probability $p_t(v)$. The nodes that got marked send a notification to neighbors and if none of their neighbors got marked as well, they join the MIS and notify their neighbors. Nodes that receive join notification from their neighbors are removed from the graph. This completes the description of the first part of the algorithm.

**Analysis.** The correctness proof of the algorithm uses a different argument than that of Ghaffari [Gha16]. Our proof does not involve claiming that a constant fraction of the golden nodes are removed, because in order to be left with $O(n/\Delta)$ undecided nodes we have to favor removal of old nodes. The entire correctness is based upon the following lemma, which justifies the definition of the expectation given in Equation (8.2).

**Lemma 8.1.8.** The number of undecided nodes after $\Theta(\log \Delta)$ phases is $O(n/\Delta)$ and hence the total number of edges incident to these nodes is $O(n)$.

**Proof.** Consider phase $t$, and let $V_t$ be the set of golden nodes in this phase. For a golden node $v \in V_t$, let $age(v)$ be the number of golden phases it had so far, excluding the current one. We refer to this quantity as the *age* of the node. Hence, intuitively, a node is *old* if it has experienced many golden phases.

By Lemma 8.1.6, a node that remains undecided after $\beta \log \Delta$ phases (for a sufficiently large constant $\beta$) is of age at least $\Omega(\log \Delta)$. Our goal is to show that the number of old nodes (with $age(v) \geq \Omega(\log \Delta)$) is bounded by $n/\Delta$. At a high level, we show that each old node can blame or charge a distinct set of $\Delta$ nodes that are removed. Hence, there are total of at most $n/\Delta$ old nodes.

Recall that in phase $t$, given the golden nodes $V_t$, the original goal is to find an assignment to the nodes so that the number of removed nodes is at least as the expected number of the removed nodes, say, at least $\alpha \cdot |V_t|$. Our algorithm modifies this objective by favoring the removal of *older* nodes. To do that, recall that $r_{x,t}$ is the random variable indicating the event that a golden node $v$ is removed at phase $t$, and $\psi_{x,t}$ is used to obtain a pessimistic estimator for $r_t$ being removed. The age of $v$ at phase $t$, $age(v)$, is the number of golden phases $v$ had till phase $t$. The nodes

---

2The term $M_{t,b}(u, W(v))$ is important as it is what prevents double counting, because the corresponding random variables defined by the neighbors of $v$ are mutually exclusive.
compute the conditional expectation for:

$$E(\Psi'_t) = \sum_{v \in V_t} \Pr[|\psi_{v,t} | \geq 1] \cdot (1/(1 - \alpha))^{age(v)}.$$

By Lemma 8.1.4 and 8.1.5, \(\Pr[|\psi_{v,t} | \geq 1] \geq \alpha\), hence:

$$E(\Psi'_t) \geq \alpha \sum_{v \in V_t} (1/(1 - \alpha))^{age(v)}.$$

For every phase \(t\), let \(sb_t = \sum_{v \in V_t} (1/(1 - \alpha))^{age(v)}\) be the total weighted age of the nodes in \(V_t\). Let \(V_{t,0} \subseteq V_t\) be the set of unremoved nodes in \(V_t\) and let \(V_{t,1} \subseteq V_t\) be the set of removed nodes in \(V_t\) (i.e., the decided ones). Let \(sb_{t,0} = \sum_{v \in V_{t,0}} (1/(1 - \alpha))^{age(v)}\) and \(sb_{t,1} = \sum_{v \in V_{t,1}} (1/(1 - \alpha))^{age(v)}\) be the total weighted age of the unremoved nodes and the removed nodes respectively.

The method of conditional expectations guarantees finding a good assignment to the nodes such that

$$sb_{t,1} \geq E(\Psi'_t) \geq \alpha \cdot sb_t.$$

We now employ an accounting method to show that each remaining node (which by Lemma 8.1.6 is proven to be old) can charge a distinct set of \(O(\Delta)\) nodes that are removed. For the sake of the analysis, we imagine every node having a bag containing removed nodes or fractions of removed nodes, and that the bags of all nodes are disjoint (no piece of removed node is multiply counted in the bags). Let \(b_t(v)\) be the size of the bag of node \(v\) at the beginning of phase \(t + 1\). The size of the bag corresponds to the number of removed nodes (or fractions of removed nodes) that are charged due to \(v\). As mentioned, our charging is fractional in the sense that we may split a removed node \(u\) into several bags \(v_1, \ldots, v_k\), meaning that each \(v_i\) partially blames \(u\) for not being removed yet, but the total charge for a given removed node \(u\) is 1.

Recall that \(\alpha\) is the bound that a golden node is removed (in the unconditional setting). For every \(t \in \{0, \ldots, \Theta(\log \Delta)\}\), we maintain the following invariants.

\[(I1) \ b_t(v) \geq (1/(1 - \alpha))^{age(v)}.
\]

\[(I2) \ \sum_{v \in V_{t,0}} (b_t(v) - b_{t-1}(v)) = \sum_{u \in V_{t,1}} b_t(u).
\]

In other words, our accounting has the property that the number of removed nodes that can be blamed for the fact that a node \(v\) is not yet removed, grows exponentially with the age of \(v\).

We now describe the charging process inductively and show that it maintains the invariants. For the induction base, let \(b_0(v) = 1\) correspond to \(v\) itself and consider the first phase in which \(age(v) = 0\) for every \(v\). Note that \(sb_0 = |V_0| = \sum_{v \in V_0} b_0(v)\).

The method of conditional expectations with respect to \(\Psi'_t\) guarantees that we find an assignment to the nodes such that \(sb_{0,1} \geq \alpha \cdot sb_0 = \alpha \cdot |V_0|\). The age of the unremoved nodes of \(V_{0,1}\) becomes 1 and hence to establish (I1), we need to grow their bags.

Towards this end, we empty the content of the bags of the removed nodes \(V_{0,1}\) into the bags of the unremoved nodes \(V_{0,0}\). In this uniform case of \(t = 0\), since all nodes are of the same age, it simply means that every unremoved node in \(V_{0,0}\) adds into its bag an \(\alpha/(1 - \alpha)\) fraction of a removed node (as \(sb_{0,0} \leq (1 - \alpha) \cdot |V_0|\), this is doable). Hence, the size of each bag of unremoved
node $v \in V_{0,0}$ becomes at least $b_1(v) \geq 1 + \frac{\alpha}{1 - \alpha}$ and property (I) follows. Note that since every unremoved node gets a disjoint fraction of a removed node, property (I2) follows as well. This completes the description for phase 1.

We continue this process phase by phase, by considering phase $t$ and assuming that the invariants hold up to $t - 1$. Again, by the method of conditional expectations, we find an assignment such that $\frac{\mathbb{E}(\Psi_t)}{t} \geq \alpha \cdot sb_t$ and that $sb_{t,1} \geq \alpha \cdot sb_t$. By the induction hypothesis, $b_t(v) \geq \frac{1}{(1 - \alpha)^{age(v)}}$ and hence

$$sb_t \leq \sum_{v \in V_t} |b_t(v)|.$$  

Since the age of the unremoved nodes in $V_{t,0}$ is increased by one, their bag content should be increased. Again, this is done by emptying the content of the bags of the removed nodes $V_{t,1}$ into the bags of the unremoved nodes. Since the total amount in the bags of the removed nodes is at least $\alpha \cdot sb_t$, their content can be divided into the bags of the unremoved nodes $V_{t,0}$ such that each unit in the bag of size $\frac{1}{(1 - \alpha)^{age(v)}}$ is given an addition of $\frac{\alpha}{1 - \alpha}$ fraction of the size of that unit, and hence overall that new size of the bag is $\frac{1}{(1 - \alpha)^{age(v)+1}}$.

Let $t^* = \Theta(\log \Delta)$. By Lemma 8.1.6, every node that remains undecided after $t^*$ phases is of age $\Omega(\log \Delta)$. By Invariant (II), we have that $b_{t^*}(v) \geq \Delta$ for every $v \in V_{t^*}$. Since the contents of $b_{t^*}(v)$ and $b_{t^*}(v')$ for every $t$ and every $v \neq v' \in V_t$ are distinct, we get that $|V_{t^*}| \leq n/\Delta$, and overall the total number of edges left in the graph in $O(n)$. □

The remaining $O(n)$ edges incident to the undecided nodes can be collected at the leader in $O(1)$ rounds using the deterministic routing algorithm of Lenzen [Len13]. The leader then solves MIS for the remaining graph locally and informs the nodes. This completes the correctness of the algorithm. Theorem 8.3 follows.

### 8.1.3 An $O(\log \Delta)$ deterministic MIS algorithm for $\Delta = O(n^{1/3})$

In the case where the maximal degree is bounded by $\Delta = O(n^{1/3})$, our deterministic bounds match the randomized ones.

**Theorem 8.4.** If $\Delta = O(n^{1/3})$ then there is a deterministic MIS algorithm (and a $(\Delta + 1)$-coloring algorithm) for the congested clique model that completes in $O(\log \Delta)$ rounds.

**Proof.** The algorithm consists of two parts as before, namely, $O(\log \Delta)$ phases that simulate the modified Ghaffari’s algorithm and collecting the remaining topology at a leader and solving MIS for it locally. The second part works exactly the same as before, and so we focus on the first part which simulates the $O(\log \Delta)$ phases of the modified Ghaffari’s algorithm in $O(\log \Delta)$ deterministic rounds. The main advantage of having a small degree $\Delta = O(n^{1/3})$ is that in the congested clique, it is possible for each node to collect the entire topology of its 2-neighborhood in $O(1)$ rounds. This because the 2-neighborhood of a node contains $O(\Delta^2)$ nodes, and hence there are $O(\Delta^3) = O(n)$ messages a node needs to send or receive, which can be done in $O(1)$ rounds using Lenzen’s routing algorithm [Len13].

We now consider phase $t$ of the modified Ghaffari’s algorithm and explain how the seed of length $O(\log n)$ can be computed in $O(1)$ rounds. Unlike the algorithm of the previous section,
which computes the seed bit by bit, here the nodes compute the assignment for a chunk of
$z = \lfloor \log n \rfloor$ variables at a time.

To do so, consider the $i$'th chunk of the seed $Y'_i = (y'_1, \ldots, y'_z)$. For each of the $n$ possible assignments $(b'_1, \ldots, b'_z) \in \{0, 1\}^z$ to the $z$ variables in $Y'$, we assign a node $u$ that receives the conditional expectation values from all the golden nodes, where the conditional expectation is computed based on assigning $y'_1 = b'_1, \ldots, y'_z = b'_z$. The node $u$ then sums up all these values and obtains the expected number of removed nodes conditioned on the assignment $y'_1 = b'_1, \ldots, y'_z = b'_z$.

Finally, all nodes send to the leader their computed sum and the leader selects the assignment $(b^*_1, \ldots, b^*_z) \in \{0, 1\}^z$ of largest value.

Using a well known reduction from the $(\Delta + 1)$-coloring to MIS [Lov93, Lub86]$^3$, by collecting the topology of the 2-neighborhood, we can obtain the same bounds for $(\Delta + 1)$-coloring.  

\subsection*{8.1.4 An $O(D \log^2 n)$ deterministic MIS algorithm for CONGEST}

Here we provide a fast deterministic MIS algorithm for the harsher CONGEST model. For comparison, in terms of $n$ alone, the best deterministic MIS algorithm is by Panconesi and Srinivasan [PS92] from more than 20 years ago is bounded by $2^{O(\sqrt{\log n})}$ rounds. While we do not improve this for the general case, we significantly improve it for graphs with a polylogarithmic diameter. The following is our main result for CONGEST.

**Theorem 8.5.** There is a deterministic MIS algorithm for the CONGEST model that completes in $O(D \log^2 n)$ rounds.

The algorithm is very similar to Theorem. 8.3 with two main differences. First, we run Ghaffari’s algorithm for $O(\log n)$ rounds instead of $O(\log \Delta)$ rounds. Each round is simulated by a phase with $O(D \log n)$ rounds. Specifically, in each phase, we need to compute the seed of length $O(\log n)$, this is done bit by bit using the method of conditional expectations and aggregating the result at some leader node. The leader then notifies the assignment of the bit to the entire graph. Since, each bit in the seed is computed in $O(D)$ rounds, overall the run time is $O(D \log^2 n)$.

For the correctness, assume towards contradiction that after $\Omega(\log n)$ rounds, at least one node remains undecided. Then, by the proof Lemma 8.1.8, every node that survives, can charge $\Omega(n^c)$ nodes that are removed, contradiction as there only $n$ nodes.

\subsection*{8.2 Deterministic spanner construction}

In this section we present a derandomization algorithm in the congested clique for the spanner construction of Baswana-Sen [BS07]. In general, we use the same general outline as in the MIS derandomization: We first reduce the dependence between the coins used by the algorithm and then use the method of conditional expectations for every iteration of the algorithm. However, here we face different challenges that we need to overcome.

The following is the main theorem of this section.

---

$^3$In this reduction, every node $v$ is replaced by a clique of $\Delta + 1$ nodes, and we add a complete matching between cliques of neighboring nodes. When computing an MIS in this graph, there must be exactly one copy, say, $j$, for node $v$ in the MIS, which implies that the color of $v$ is $j$. 

106

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-08 - 2017
Theorem 8.6. There is a deterministic \((2k - 1)\)-spanner algorithm for the congested clique model that completes in \(O(k \log n)\) rounds and produces a spanner with \(O(kn^{1+1/k} \log n)\) edges.

We first present the original algorithm of Baswana-Sen [BS07], which constructs a \((2k - 1)\)-spanner with \(O(kn^{1+1/k})\) edges in \(O(k^2)\) rounds. Next, we consider the same algorithm with only limited independence between its coin tosses. We prove some properties of the algorithm and show it can be derandomized. Finally we present our deterministic algorithm for the congested clique which constructs a \((2k - 1)\)-spanner with \(O(kn^{1+1/k} \log n)\) edges within \(O(k \log n)\) rounds.

The randomized spanner algorithm. We begin by presenting a simplified version of the Baswana-Sen algorithm. For the full details of the Baswana-Sen algorithm we refer the reader to [BS07].

At each iteration the algorithm maintains a clustering of the nodes. A cluster is a subset of nodes, and a clustering is a set of disjoint clusters. In the distributed setting each cluster has a leader, and a spanning tree rooted at the leader is maintained inside the cluster. We will abuse notation and say that a cluster performs a certain action. When we say this, it means that the leader gathers the required information from the cluster nodes to make a decision, and propagates relevant data down the cluster tree. We will also refer at times to the ID of a cluster, which is the ID of the cluster leader.

We denote the clustering maintained at iteration \(i\) by \(C_i\), where initially \(C_0 = \{\{v\} \mid v \in V\}\). At each iteration, \(C_i\) is sampled from \(C_{i-1}\), by having every cluster in \(C_{i-1}\) join \(C_i\) with probability \(n^{-1/k}\). In the final iteration we force \(C_k = \emptyset\). A node \(v\) that belongs to a cluster \(C \in C_i\) is called \(i\)-clustered, and otherwise it is \(i\)-unclustered.

The algorithm also maintains a set of edges, \(E'\), initialized to \(E\). For every edge \((u, v)\) removed from \(E'\) during the algorithm, it is guaranteed that there is a path from \(u\) to \(v\) in the constructed spanner, \(H_i\), consisting of at most \(2k - 1\) edges, each one of weight not greater than the weight of \((u, v)\).

Let \(v \in V\) be a node that stops being clustered at iteration \(i\), and let \(E'(v, C) = \{(v, u) \in E' \mid u \in C\}\) be the set of edges between a node and a cluster \(C\), for every \(C \in C_i\). Let \(e_{v,C}\) be the lightest edge in \(E'(v, C)\). Let \(L\) be the set of lightest edges between \(v\) and the clusters in \(C_{i-1}\). We go over \(L\) in ascending order of edge weights and add every edge connecting \(v\) to cluster \(C\) and then we discard \(E'(v, C)\) from \(E'\). We say that \(v\) adds these edges at iteration \(i\). If we reach a cluster \(C \in C_i\), we continue to the next node. The pseudocode appears in Algorithms 14 and 15.

Algorithm 14 is guaranteed to finish after \(O(k^2)\) communication rounds in the distributed setting, and return a \((2k - 1)\)-spanner of expected size \(O(kn^{1+1/k})\).

d-wise independence and derandomization. The Baswana-Sen algorithm does not work as is with reduced independence, because the bound on the spanner size relies on full independence between the coin flips of the clusters. However, we proceed by establishing properties of the Baswana-Sen algorithm (Algorithm 14) that do hold in the case of limited independence between its coin tosses. We use following result of Benjamini et al. [BGGP12].
Algorithm 14: Randomized \((2k - 1)\)-spanner construction

1. \(H = \emptyset\)
2. \(C_0 = \{\{v\} | v \in V\}\)
3. \(E' = E\)

4. for \(i\) from 1 to \(k\) do
5. if \(i = k\) then
6. \(C_k = \emptyset\)
7. else
8. \(C_i\) is sampled from \(C_{i-1}\) by sampling each cluster with probability \(n^{-1/k}\)
9. Run Algorithm 15

Algorithm 15: Baswana-Sen iteration

1. foreach node \(v\) that stopped being clustered at iteration \(i\) do
2. \(L = \{e_{v,C} | C \in C_{i-1}\}\)
3. Let \(e_j\) be the \(j\)-th edge in \(L\) in ascending weight
4. for \(j = 1\) to \(j = \ell\) do
5. \(H = H \cup \{e_j\}\)
6. Let \(C\) be the cluster corresponding to \(e_j\)
7. \(E' = E' \setminus E'(v, C)\)
8. if \(C \in C_i\) then
9. break

Theorem 8.7. Let \(M(n, d, p)\) be the maximal probability of the AND event for \(n\) binary \(d\)-wise independent random variables, each with probability \(p\) of having the value 1. If \(d\) is even, then:

\[
M(n, d, p) \leq \frac{p^n}{\Pr[Bin(n, 1 - p) \leq d/2]}.
\]

and if \(d\) is odd, then:

\[
M(n, d, p) = pM(n - 1, d - 1, p).
\]

We also use the following Chernoff bound for \(d\)-wise independent random variables from [SSS95].

Theorem 8.8. Let \(X_1, \ldots, X_n\) be \(d\)-wise independent random variables taking values in \([0, 1]\), where \(X = \sum_{i=1}^{n} X_i\) and \(E[X] = \mu\). Then for all \(\epsilon \leq 1\) we have that if \(d \leq \lfloor \epsilon^2 \mu e^{-1/3} \rfloor\) then:

\[
\Pr[|X - \mu| \geq \epsilon \mu] \leq e^{-|d/2|}.
\]

And if \(d > \lfloor \epsilon^2 \mu e^{-1/3} \rfloor\) then:

\[
\Pr[|X - \mu| \geq \epsilon \mu] \leq e^{-|\epsilon^2 \mu/3|}.
\]

We consider Algorithm 14 with only \(O(\log n)\)-wise independence between the coin tosses of clusters at each iteration. We also assume that \(k \leq 0.5 \log n\). We prove the following two lemmas that will be used for derandomization.

Let \(d = 2 \log 2n\) be the independence parameter, and define \(\xi = e^{1/3}2\log 2n\), and \(\alpha_i = \prod_{j=1}^{i} (1 + 1/(k - j))\).
Lemma 8.2.1. For every $1 \leq i \leq k-1$, if $|C_{i-1}| \leq \xi \alpha_{i-1} n^{1-(i-1)/k}$, then $\Pr[|C_i| \geq \xi \alpha_i n^{1-i/k}] < 0.5$. In addition, $\xi \alpha_{k-1} n^{1/k} = O(k n^{1/k} \log n)$.

Proof. We define for every cluster $C$ the indicator random variable $X(C)$ for the event that the cluster remains for the next iteration. Note that $|C_i| = \sum X(C)$ and $\mathbb{E}[X(C)] = n^{-1/k}$.

By the assumption of the lemma, for the $(i-1)$-th iteration we know that we have at most $\xi \alpha_{i-1} n^{1-(i-1)/k}$ clusters left from the previous iteration. Thus $\mathbb{E}[\sum X(C)] \leq \xi \alpha_{i-1} n^{1-(i-1)/k}$. Now we show (b), that is, that

$$n^{-1/k} \leq \xi \alpha_{i-1} n^{1-i/k}.$$

We wish to apply Theorem 8.8 with $d = 2 \log 2n, \mu_i = \xi \alpha_{i-1} n^{1-i/k}$ and $\epsilon_i = 1/(k-i)$. We note that $\alpha_i \geq 1$ for every $i$. We now show that it is always the case that $d \leq \lfloor \epsilon_i^2 \mu_i e^{-1/3} \rfloor$, so we can use the first case of Theorem 8.8. Plugging in $\epsilon_i, \mu_i, d$ gives that we need to prove that:

$$2 \log 2n \leq 2 \log(2n) \cdot e^{1/3} e^{-1/3} \alpha_{i-1} n^{1-i/k} / (k-i)^2,$$

which holds if and only if

$$\alpha_{i-1} n^{1-i/k} / (k-i)^2 \geq 1.$$

We bound the left hand side from below by

$$\alpha_{i-1} n^{1-i/k} / (k-i)^2 \geq n^{1-i/k} / (k-i)^2.$$

To prove that the above is at least 1, we claim that (a) the function $n^{1-i/k} / (k-i)^2$ is monotonically decreasing for $1 \leq i \leq k-1$, and (b) that $n^{1-i/k} / (k-i)^2 \geq 1$ when $i = k-1$. To prove (a), we prove that $n^{1-i/k} / (k-i)^2 \leq n^{1-(i-1)/k} / (k-(i-1))^2$. Taking the square root of both sides gives that we need to prove that

$$n^{1/2-i/2k} / (k-i) \leq n^{1/2-(i-1)/2k} / (k-(i-1)),$$

which holds if and only if

$$(k-(i-1)) / (k-i) \leq n^{1/2k}.$$  

For the left hand side of the above, it holds that

$$(k-(i-1)) / (k-i) \leq 1 + 1 / (k-i) \leq 2,$$

and since we assumed that $k < 0.5 \log n$, we have that $n^{1/2k} \geq n^{1/\log n} = 2$. Therefore, $n^{1/2k} \geq (k-(i-1)) / (k-i)$ as required for (a).

We now show (b), that is, that $n^{1-i/k} / (k-i)^2 \geq 1$ when $i = k-1$. This holds since for $i = k-1$ we have $n^{1-i/k} / (k-i)^2 = n^{1-(k-1)/k} / (k-(k-1))^2 = n^{1/k} \geq 1$, giving (b). This establishes that $d \leq \lfloor \epsilon_i^2 \mu_i e^{-1/3} \rfloor$, and thus the first condition of Theorem 8.8 always holds.

Since $\alpha_i = (1+1/(k-i))\alpha_{i-1} = (1+\epsilon_i)\alpha_{i-1}$ we have

$$\Pr[|C_i| \geq \xi \alpha_i n^{1-i/k}] = \Pr[\sum X(C) \geq \xi (1+\epsilon_i)\alpha_{i-1} n^{1-i/k}] = \Pr[\sum X(C) \geq (1+\epsilon_i)\mu_i].$$
We now apply Theorem 8.8 and obtain

$$\Pr[\sum X(C) - \mu_i \geq \epsilon_i \mu_i] < e^{-[d/2]} < 0.5,$$

which proves the first part of the lemma, that $\Pr[|C_i| \geq \xi \alpha n^{1-i/k}] < 0.5$.

Finally, we have

$$\alpha_k = \prod_{j=1}^{k-1} (1 + 1/(k - j)) \leq e^{\sum_{j=1}^{k-1} 1/(k-j)} = e^{\sum_{j=1}^{k-1} 1/j} = O(k).$$

Which implies the second part of the lemma, that $\xi \alpha_k n^{1/k} = O(k n^{1/k} \log n)$, and completes the proof.

Fix an iteration $i$ and consider an $i$-unclustered node $v$. Denote by $X_v$ the indicator variable for the event that node $v$ adds more than $t = 2n^{1/k} \log n$ edges in this iteration.

**Lemma 8.2.2.** The probability that there exists a node $v$ at some iteration which adds more than $t$ edges to the spanner is less than 0.5. Formally, $\Pr[\bigvee X_v = 1] < 0.5$.

**Proof.** From the union bound it holds that $\Pr[\bigvee X_v = 1] \leq \Pr[X_v = 1]$. Next, we bound $\sum \Pr[X_v = 1]$. We show that every $\Pr[X_v = 1]$ is smaller than $1/2n$, completing the proof by applying a union bound over all nodes. Let $\ell$ be the number of neighboring clusters of $v$ in $C_{i-1}$. If $\ell \leq t$ then $\Pr[X_v = 1] = 0$. Otherwise, we might add $t$ edges to $H$, if and only if the clusters corresponding to the $t$ lightest edges in $L$ are not in $C$. This is the value $M(t, 2d, p)$ (we use $2d$ to avoid fractions in the binomial coefficient) with $p = 1 - n^{-1/k}$. Let us bound $M(t, 2d, p)$ as follows.

$$M(t, 2d, p) \leq \frac{p^t}{\Pr(\text{Bin}(t, 1-p)) \leq \binom{t}{d} (1-p)^d (1-p)^{-d} = \frac{p^d}{\binom{t}{d} (1-p)^d} \leq \frac{1}{\binom{t}{d} (1-p)^d} \leq \frac{1}{(t/d)^d (1-p)^d} \leq \frac{d^d}{t^d (1-p)^d}.$$

Plugging in $p = 1 - n^{-1/k}$ and $t = 2n^{1/k} \log n$ gives

$$M(2n^{1/k} \log n, 2d, 1 - n^{-1/k}) \leq \frac{d^d}{(2n^{1/k} \log n)^d (n^{-1/k})^d} = \frac{d^d}{(2 \log n)^d}.$$

Now let us plug in $d = 2 \log 2n$ and we get:

$$M(2n^{1/k} \log n, 2 \log 2n, 1 - n^{-1/k}) \leq (1/2)^{2 \log 2n} < 1/2n.$$

Finally, as explained, we use a union bound to get that $\Pr[\bigvee X_v = 1] \leq \sum \Pr[X_v = 1] < 0.5$. 

The above lemmas do not guarantee that the algorithm yields the same expected spanner size as the algorithm with full independence, but using these lemmas we can now construct a deterministic algorithm.

Let us define two bad events that can occur during some iteration $i$ of the algorithm. Let $A$ be the event that not enough clusters were removed during the iteration, and let $B$ be the event
that there exists a node that adds too many edges to the spanner. We will define these events formally later on. Let $X_A, X_B$ be the corresponding indicator random variables for the events. Assume that it holds that $E[X_A] + E[X_B] < 1$. In this case we can use the method of conditional expectations in order to get an assignment to our random coins such that no bad event occurs.

Let $\tilde{\rho}$ be the vector of coin flips used by the clusters. Let $Y$ be the seed randomness from Lemma 3.3.2 used to generate $\tilde{\rho}$ such that its entries are $d$-wise independent, where $d = O(\log n)$. We use $Y$ to select a function $h \in \mathcal{H}_{\gamma, \beta}$, where $\gamma = \log n$ and $\beta = \log n^{1/k}$. Each node $v$ uses $Y$ to generate $h$ and then uses the value $h(ID(v))$ to generate $\tilde{\rho}[v]$.

Let $Z = (z_1, \ldots, z_n)$ be the final assignment generated by the method of conditional expectations. Then, $E[X_A \mid Y = Z] + E[X_B \mid Y = Z] < 1$. Because $X_A$ and $X_B$ are binary variables that are functions of $Y$, it must be the case that both are zero. We can write our expectation as follows:

$$E[X_A] + E[X_B] = \Pr[X_A = 1] + \Pr[X_B = 1] = \Pr[X_A = 1] + \Pr[\forall v \in Z = 1]$$

At every iteration of the algorithm we would like to keep $E[X_A] + E[X_B]$ below 1, which would guarantee both bad events do not occur. Unfortunately, it is unclear how to compute $\Pr[\forall v \in Z = 1]$ conditioned on some assignment to $Y$. Thus, we must use a pessimistic estimator. We consider $\sum_v \Pr[X_v = 1]$, and we have that:

$$\Pr[X_A = 1] + \Pr[\forall v \in Z = 1] \leq \Pr[X_A = 1] + \sum_v \Pr[X_v = 1].$$

We define our pessimistic estimator $\Psi = X_A + \sum_v X_v$. Note that the above inequality holds conditioned on any partial assignment to $Y$, because it is derived via a union bound. Thus, if we show that $E[\Psi] = \Pr[X_A = 1] + \sum \Pr[X_v = 1] < 1$, it is enough to apply the method of conditional expectations for $\Psi$, keeping the expression below 1. For the assignment $Z$ resulting from this process it will hold that $E[X_A \mid Y = Z] + E[X_B \mid Y = Z] < E[\Psi \mid Y = Z] < 1$, as required.

It remains only to bound the pessimistic estimator $\Psi$. This can be achieved using Lemma 8.2.1 and Lemma 8.2.2. In each iteration of the algorithm, because the bad event $A$ did not occur in the previous iteration, the condition that $|C_{i-1}| \leq \xi \alpha_{i-1} n^{1-(i-1)/k}$ holds for Lemma 8.2.2. This yields $\Pr[X_A = 1] + \sum \Pr[X_v = 1] < 1$.

The deterministic spanner construction in the congested clique. We are now ready to describe our algorithm in the congested clique. We first show the we can indeed compute the conditional expectation of our pessimistic estimator $\Psi$.

We are interested in $\Pr[X_A = 1 \mid y_1 = b_1, \ldots, y_i = b_i]$ and $\Pr[X_v = 1 \mid y_1 = b_1, \ldots, y_i = b_i]$. Knowing some partial assignment to $Y$, we can iterate over all possible selections of $h \in \mathcal{H}_{\gamma, \beta}$ and compute the coin flip for every cluster using its ID alone. The expression $\Pr[X_A = 1 \mid y_1 = b_1, \ldots, y_i = b_i]$ is just the probability of enough clusters getting removed given some partial assignment. It does not depend on the graph topology, and can easily be computed by a node only knowing the IDs of clusters currently active. To compute $\Pr[X_v = 1 \mid y_1 = b_1, \ldots, y_i = b_i]$ the node $v$ can collect all of the IDs from neighboring clusters and go over all possibilities for
calculate the probability of adding too many edges.

Algorithm 16: deterministic \((2k-1)\)-spanner algorithm

1. \(H = \emptyset\)
2. \(C_0 = \{\{v\} | v \in V\}\)
3. \(E' = E\)
4. \textbf{for} \(i\) \textbf{from} 1 \textbf{to} \(k\) \textbf{do}
5. \(\phi = \emptyset\) //partial assignment
6. \textbf{foreach} \(v \in V\) \textbf{simultaneously do}
7. \textbf{if} \(v\) is cluster leader for \(C \in C_{i-1}\) \textbf{then}
8. Send \(ID(v)\) to all nodes
9. \textbf{for} \(j \in [(\log n)]\) \textbf{do}
10. \textbf{for} \(\tau \in [\log n]\) \textbf{do}
11. Compute \(x_\tau = \Pr[X_v | \phi, y_j = \tau]\)
12. Send \((x_\tau, \tau)\) to \(u \in V, ID(u) = \tau\)
13. \textbf{if} \(ID(v) = \tau, \tau \in [\log n]\) \textbf{then}
14. \(s = \Pr[X_A | \phi, y_j = \tau] + \sum(x, \tau) x\)
15. Send \((\tau, s)\) to main leader
16. \textbf{if} leader \textbf{then}
17. \(\tau_{\min} = \arg\min_\tau \{s | (\tau, s)\}\)
18. \(\phi = \phi \cup \{y_j = \tau_{\min}\}\)
19. send updated \(\phi\) to all nodes
20. \textbf{if} \(i = k\) \textbf{then}
21. \(C_i = \emptyset\)
22. \textbf{else}
23. \(C_i\) sampled from \(C_{i-1}\) using \(Y\)
24. Run Algorithm 15

Algorithm 16 is the pseudocode, where before running an iteration of the Baswana-Sen algorithm we first find a seed randomness \(Y\), such that both bad events \(A\) and \(B\) do not occur. We then execute an iteration of the Baswana-Sen algorithm using the seed to assign a random coin for each cluster. Because neither of the bad events occur in any iteration, no node adds more than \(2n^{1/k}\log n\) edges in any iteration, and we reach the final iteration with \(O(n^{1/k})\) clusters. Therefore, each iteration adds no more than \(O(n^{1+1/k}\log n)\) edges, and the final iteration adds no more than \(O(kn^{1+1/k}\log n)\) edges (assuming a loose bound of having all nodes connect to all remaining clusters). We conclude that our spanner has \(O(kn^{1+1/k}\log n)\) edges.

We find \(Y\) via the method of conditional expectations, keeping the pessimistic estimator below 1. We consider the value of the pessimistic estimator under some partial assignment to \(Y\), and extend the assignment such that the pessimistic estimator is kept below 1.

When finding \(Y\) we bring the power of the congested clique to our aid. The sequential approach would go over \(Y\) bit by bit, setting it to the value which optimizes the pessimistic estimator until all values of \(Y\) are fully set. In the congested cliques we can go over blocks of \(Y\) of size \(\log n\), calculating the value of the pessimistic estimator for each one of the \(n\) possible assignments of the block. We achieve this by assigning each node to be responsible for aggregating
the data in order to calculate the pessimistic estimator for one of the possible \( n \) values. This speeds up our calculation by a \( \log n \) factor.

The above is implemented in the algorithm as follows: Each node \( v \in V \) iterates over all \( \log n \) blocks of \( Y \), each of size \( \log n \). For each block it computes \( \Pr[X_v] \) conditioned on all \( n \) values of the block. For every value \( \tau \) of the block it sends each the conditional probability to \( u_\tau \), which is responsible for computing the value of the pessimistic estimator conditioned on the value \( \tau \) for the block. Knowing the conditional value of \( \Pr[X_v] \) for every \( v \in V \) and the IDs of the active clusters, the node \( u_\tau \) can now compute the value of the conditional pessimistic estimator. All of the conditional values of the pessimistic estimator are then aggregated to a leader node which picks the value that minimizes the pessimistic estimator. Finally, the leader broadcasts the selected value for the block to all nodes. All nodes then continue to the next iteration. After computing \( Y \) we run an iteration of Baswana-Sen where the coin tosses of clusters are generated from \( Y \).

Another benefit of running the Baswana-Sen algorithm in the congested clique is that we save an \( O(k) \) factor in our round complexity. This is because cluster nodes may now communicate with the cluster leader directly, instead of propagating their message via other cluster nodes. This takes \( O(k) \) in the standard distributed setting because the distance to the center of each cluster is at most the iteration number.

We conclude that the round complexity of our algorithm is the number of iterations of the Baswana-Sen main loop in the congested clique, which is \( O(k) \), multiplied by the overhead of guaranteeing the bad events \( A, B \) will not happen during the iteration. We guarantee this by applying the method of conditional expectation over \( Y \), using a block of size \( \log n \) at each step of the method of conditional expectations.

We note that each cluster flips a biased coin with probability \( n^{-1/k} \), and we require \( d \)-wise independence between the coin flips. We conclude from Lemma 3.3.2 that the size of \( Y \) is \( O(d \max \{ \log n^{1/k}, \log n \}) = O(\log^2 n) \) bits. Because, we pay \( O(1) \) rounds for every \( \log n \) chunk of \( Y \), we conclude from the above that our algorithm takes a total of \( O(k \log n) \) communication rounds. This completes the proof of Theorem 8.6.
Chapter 9

Conclusions

We have shown the local-ratio framework to be a powerful tool for designing approximation algorithms under conditions of uncertainty. We believe that it is possible to achieve more state of the art results using our techniques. We also believe that these techniques can be extended beyond the distributed and streaming environments.

Our refutation of [KMW10] implies that some algorithms that were previously thought to be optimal may not be so. Specifically, the MIS algorithm of [Gha16] and the maximal matching algorithm of [BEPS12] are now log log factors apart from their respective known lower bound. This gives rise to the intriguing question of whether these algorithms can be improved upon, or whether there is a tighter lower bound that can be proven for them.

In the semi-streaming environment, our maximum matching approximation algorithm almost closes the gap between the best known approximation ratio for the weighted and unweighted matching problem. The best known algorithm for the unweighted case is the trivial greedy algorithm which provides a 2-approximation, while our algorithm provides a \((2 + \epsilon)\)-approximation. This gives rise to two questions, the first one is can we achieve a 2-approximation for the weighted case, completely closing the gap. The second is, can we achieve a better approximation ratio for the unweighted case, thus widening the gap.

We also explore derandomization in the congested clique, arriving to a deterministic MIS algorithm in the CONGEST, which has a favorable running time in graphs with low diameter. Currently there is an exponential gap between the deterministic and randomized running time of many distributed problems, chief among them is computing an MIS. A central problem is exploring this gap with respect to the MIS problem, either providing a fast deterministic algorithm or proving the gap cannot be bridged.
Bibliography


Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *53rd Annual IEEE Symposium*


James W. Hegeman, Sriram V. Pemmaraju, and Vivek Sardeshmukh. Near-
constant-time distributed algorithms on a congested clique. In Distributed
Computing - 28th International Symposium, DISC 2014, Austin, TX, USA,

Magnús M Halldórsson and Jaikumar Radhakrishnan. Greed is good: Approx-
imating independent sets in sparse and bounded-degree graphs. Algorithmica,

Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large cuts

Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest
paths and applications. In Proceedings of the 2012 ACM Symposium on
Principles of Distributed Computing, PODC ’12, pages 355–364, New York,
NY, USA, 2012. ACM.

Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm

of a symposium on the Complexity of Computer Computations, held March 20-
22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights,


Jarkko Kari, Martín Matamala, Ivan Rapaport, and Ville Salo. Solving
the induced subgraph problem in the randomized multiparty simultaneous
messages model. In Structural Information and Communication Complexity -
22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July

David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph
coloring by semidefinite programming. Journal of the ACM (JACM), 45(2):246–

Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot
be computed locally! In Proceedings of the Twenty-Third Annual ACM
Symposium on Principles of Distributed Computing, PODC 2004, St. John’s,

Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of
being near-sighted. In Proceedings of the seventeenth annual ACM-SIAM


and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011), pages 85–94, 2011.


maximum unweighted matching \((1 + \epsilon)\) for the problem \((\Delta + \epsilon)\). Additionally, we present an algorithm \([KMW10]\) for the lower bound of maximum weighted matching.

The algorithms presented in this thesis are \((3.5 + \epsilon)\).

Our work also applies to semi-streaming property testing.

In summary, the techniques that we have developed are useful for many properties of graphs:

- Graph connectivity
- Property testing
- Congested Clique
- \(k\)-Spanner calculation

Moreover, we also develop tools that may be used in general applications in the congested model.

Additional insights into the properties of graphs can be found in the following:

- No, semi-streaming property testing
- Congested Clique
- Maximum weighted matching

Furthermore, we also present bounds for some of our properties.
 Inhalation study focuses on computing in semi-streaming environments, where the computing context is affected by congestion, where each node in the graph is semi-streaming in its environment. The congested environment aims to develop algorithms that solve problems on the communication graph (for example, finding a subgraph with specific properties within the communications network). In the congested environment, messages can be sent to a global landmark, different from the local model. In the model CONGEST and LOCAL, models for centralized constraints, the additional model CONGESTION is available, which limits the messages transmitted. In the models, we aim to solve problems on the communication graph (for example, finding a subgraph with specific properties within the communications network). In the congested environment, messages can be sent to a global landmark, different from the local model.

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-08 - 2017

Such a setting is known as semi-streaming, where messages flow between nodes, but are not persisted. In the model semi-streaming, messages flow between nodes, but are not persisted. In the model semi-streaming, messages flow between nodes, but are not persisted. In the model semi-streaming, messages flow between nodes, but are not persisted.

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-08 - 2017


אלגוריתמים בשני יーズ הוס ודודת

חבר על מחקר

לשמ שלカフェ חלפי הדרישות לתבולה התיהר
ודקוטר לפלוספסיה

גרוגריו שורצמן

הגה לכנפיו הטכניון - מכון טכנולוגי לישראל
סמךaltı"ב התשע"ז יוני 2017

חיפה

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-08 - 2017
אלגוריתמים בתנאי חוסר נתונים

גרגוריו שורצמן