Distributed Distance Computation and Related Topics

Ami Paz
Distributed Distance Computation and Related Topics

Research Thesis

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

Ami Paz

Submitted to the Senate of the Technion — Israel Institute of Technology
Elul 5777 Haifa August 2017
Acknowledgments

This PhD research was conducted under the supervision of Professor Keren Censor-Hillel in the Computer Science department of the Technion. Keren has guided me in my research and knew how to direct me on the one hand, and allow me the freedom to pursue my own research interests on the other hand — always with kind words and a smile. I have learned from her not only about computer science, but also about how to conduct effective research. I thank her from the bottom of my heart.

Throughout my doctoral research, Keren encouraged me to collaborate with other researchers, which I found both fruitful and enjoyable. I would like to thank all my coauthors, from whom I learned a great deal.

The department of Computer Science was my second home in these years. I have benefited from great research facilities, learned from excellent researchers, and taught clever and curious students. I owe special thanks to Hagit, my Master’s supervisor. Above all, I met friends and colleagues in this department, with whom I had spent numerous lunches, tea breaks, Billiard games and water cooler conversations.

My parents and grandparents raised me to become a scientist. They have always believed in me and encouraged me to excel, without compromising on the other enjoyments in life. I thank them for that.

My beloved wife Tal always knows how to make me laugh, and shares with me the successes and challenges of academic life. Together we traveled Israel and the world, wherever the wind and the conferences took us. From Tal I learned how to love cats, and Buzh in particular — our cat, baby and boss. Tal and Buzh always remind me what is really important in life, and to them I dedicate this thesis.

The generous financial help of the Israel Science Foundation (grant 1696/14) and the Technion are gratefully acknowledged.

Part of the results in this thesis were published in several research papers:

[CKK+15, CKPY17, PS17, CPP17, CKP17].
# Contents

List of Figures .......................................................... v

List of Tables ............................................................ vii

Abstract ........................................................................... 1

1 Introduction ............................................................... 3

1.1 Prior Results and Our Results .................................... 6

1.1.1 Weighted BFS Trees ............................................. 6

1.1.2 The Construction of Graph Spanners ....................... 7

1.1.3 Quadratic and Almost-Quadratic Lower Bounds ........... 8

1.1.4 Computing Distances Using Algebraic Techniques ...... 10

1.1.5 Approximate Proof-Labeling Schemes ...................... 11

1.1.6 Maximum Weight Matching in the Semi-Streaming Model 13

2 Preliminaries ............................................................... 15

2.1 Graph Theoretic Concepts ......................................... 15

2.2 Models of Computation ............................................. 15

2.3 Communication Complexity ....................................... 16

2.3.1 Protocols and Definitions .................................... 16

2.3.2 Typical Problems ................................................. 17

3 Parallel Construction of Weighted BFS Trees ................... 19

3.1 Multiple Weighted BFS Trees .................................... 20

3.2 Discussion and Open Questions .................................. 26

4 Distributed Construction of Purely Additive Spanners ........... 27

4.1 Introduction ............................................................ 27

4.1.1 The Challenge .................................................... 27

4.1.2 Our Contribution ................................................. 29

4.1.3 Related Work ..................................................... 32

4.2 Building Spanners .................................................... 33

4.2.1 Basic Building Blocks .......................................... 34
4.2.2 A (+2)-Sourcewise Spanner ........................................... 37
4.2.3 A (+4)-All-Pairs Spanner ............................................ 40
4.2.4 A (+2)-Pairwise Spanner ............................................. 42
4.2.5 A (+4)-Pairwise Spanner ............................................. 43
4.2.6 Subsetwise Spanners ....................................................... 47
4.2.7 A (+2)-All-Pairs Spanner ............................................. 47
4.2.8 A (+6)-All-Pairs Spanner ............................................. 49
4.3 Lower Bounds ................................................................. 53
4.3.1 A Communication Complexity Problem ................................ 53
4.3.2 A Lower Bound for Constructing a (+2)-Pairwise Spanner .......... 54
4.3.3 A Lower Bound for Constructing an \((\alpha, \beta)\)-Pairwise Spanner ............ 56
4.4 Discussion ................................................................. 58

5 Quadratic and Near-Quadratic Lower Bounds for the Congest Model 59
5.1 Introduction ................................................................. 59
5.1.1 The Challenge ............................................................ 59
5.1.2 Additional Related Work ............................................... 61
5.2 Lower Bound Graphs ...................................................... 62
5.3 Near-Quadratic Lower Bounds for NP-Hard Problems ................. 63
5.3.1 Minimum Vertex Cover .................................................... 63
5.3.2 Graph Coloring ............................................................ 66
5.4 Quadratic and Near-Quadratic Lower Bounds for Problems in P .......... 71
5.4.1 Weighted Cycle Detection ............................................. 71
5.4.2 Identical Subgraphs Detection ........................................ 74
5.5 Weighted APSP ............................................................... 76
5.5.1 A Linear Lower Bound for Weighted APSP ......................... 77
5.5.2 The Alice-Bob Framework Cannot Give a Super-Linear Lower Bound for Weighted APSP ........................................ 78
5.6 Discussion ................................................................. 81

6 Algebraic Methods in the Congested Clique 83
6.1 Introduction ................................................................. 83
6.1.1 Applications in Subgraph Detection ................................... 84
6.1.2 Applications in Distance Computation ................................ 85
6.1.3 Additional Related Work ............................................... 86
6.2 Matrix Multiplication Algorithms ....................................... 87
6.2.1 Semiring matrix multiplication ........................................ 88
6.2.2 Fast Matrix Multiplication ............................................ 90
6.3 Upper Bounds .............................................................. 94
6.3.1 Subgraph Detection and Counting ................................... 94
6.3.2 Girth ................................................................. 99
6.3.3 Routing and Shortest Paths ........................................ 101
6.3.4 Witness Detection for Distance Product ....................... 106
6.4 Lower Bounds ......................................................... 107
6.5 Conclusions .......................................................... 109

7 Approximate Proof-Labeling Schemes ................................. 111
7.1 Introduction ............................................................ 111
  7.1.1 Our Contribution ............................................... 111
  7.1.2 Related Work ................................................... 113
7.2 Model and Definitions ............................................... 114
  7.2.1 Computational Framework .................................... 114
  7.2.2 Proof-Labeling Schemes ....................................... 114
  7.2.3 The New Model: Approximate Proof-Labeling Schemes ... 115
  7.2.4 Problem Definitions .......................................... 115
7.3 PLS and APLS for Diameter .......................................... 116
7.4 Maximum Weight Matching .......................................... 123
7.5 Discussion ............................................................ 125

8 A (2 + $\epsilon$)-Approximation for Maximum Weight Matching in the Semi-
  Streaming Model ......................................................... 127
  8.1 Introduction .......................................................... 127
    8.1.1 Related Work ................................................ 129
  8.2 Preliminaries ........................................................ 130
  8.3 Approximating Maximum Weight Matching ...................... 131
    8.3.1 A Simple Local-Ratio Approximation Algorithm for MWM ... 131
    8.3.2 Extending the Local-Ratio Technique .......................... 132
  8.4 A Semi-Streaming Algorithm ..................................... 134
    8.4.1 Algorithm $\text{MWM-seq}$ .................................... 135
    8.4.2 Implementing Algorithm $\text{MWM-seq}$ in the Semi-Streaming Model 140

9 Conclusions ............................................................. 143

Bibliography ............................................................. 145

Hebrew Abstract
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>BFS-Trees guarantees a (+2) stretch for paths with many missing edges</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>Algorithm 2S guarantees a (+2) stretch for each pair $(s,v) \in S \times V$</td>
<td>39</td>
</tr>
<tr>
<td>4.3</td>
<td>Algorithm 4AP guarantees a (+4) stretch for each pair $(u,v) \in V \times V$</td>
<td>41</td>
</tr>
<tr>
<td>4.4</td>
<td>Algorithm 4P guarantees a (+4) stretch for each pair $(u,v) \in P$</td>
<td>45</td>
</tr>
<tr>
<td>4.5</td>
<td>Algorithm 2AP guarantees a (+2) stretch for each pair $(u,v) \in V \times V$</td>
<td>48</td>
</tr>
<tr>
<td>4.6</td>
<td>Algorithm 6AP guarantees a (+6) stretch for each pair $(u,v) \in V \times V$</td>
<td>51</td>
</tr>
<tr>
<td>5.1</td>
<td>The family of lower bound graphs for deciding the size of a vertex cover</td>
<td>64</td>
</tr>
<tr>
<td>5.2</td>
<td>The family of lower bound graphs for coloring</td>
<td>67</td>
</tr>
<tr>
<td>5.3</td>
<td>The family of lower bound graphs for detecting weighted cycles</td>
<td>72</td>
</tr>
<tr>
<td>6.1</td>
<td>Semiring matrix multiplication: partitioning scheme for matrix entries</td>
<td>89</td>
</tr>
<tr>
<td>6.2</td>
<td>Fast matrix multiplication: partitioning schemes for matrix entries</td>
<td>92</td>
</tr>
<tr>
<td>6.3</td>
<td>4-cycle detection: how $P(<em>,</em>,*)$ is partitioned among the nodes</td>
<td>98</td>
</tr>
<tr>
<td>7.1</td>
<td>The diameter lower bound construction for $s = 6$</td>
<td>117</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Spanner—number of edges: our constructions versus prior, sequential work</td>
<td>29</td>
</tr>
<tr>
<td>4.2</td>
<td>Spanner construction running time: algorithms versus lower bounds</td>
<td>30</td>
</tr>
<tr>
<td>6.1</td>
<td>Algorithms in the CLIQUE: our results versus prior work</td>
<td>84</td>
</tr>
<tr>
<td>7.1</td>
<td>APLS for ((D \leq k)) on general graphs—upper and lower bounds on proof size</td>
<td>112</td>
</tr>
<tr>
<td>7.2</td>
<td>APLS for ((w(M) \geq w(\text{MWM})))—upper and lower bounds on proof size</td>
<td>112</td>
</tr>
</tbody>
</table>
Abstract

The wide spread of distributed systems in recent years poses new challenges to computer scientists. This thesis studies several problems in distributed systems, focusing on distance computation problems and other, related topics.

One of the central distance computation problems is the computation of all-pair-shortest-paths, where each unit of a distributed system should learn its distances to all other units. For this problem, we present fast algorithms in networks with all-to-all communication, using fast matrix multiplication algorithms. In general networks, this problem seems much harder, and for this setting we prove that improving the state-of-the-art lower bounds will require new tools.

Another distance-related problem is the construction of sparse subgraphs that approximately preserve distances, called spanners. Spanners have a wide range of applications in distributed systems, but the study of their distributed construction still lags behind the study of sequential spanner-construction algorithms. We present distributed algorithms for the construction of spanners that preserve the distances up to an additive factor, and prove lower bounds on the time needed for their construction.

We continue the study of lower bounds, and prove almost quadratic time lower bounds for several natural problems in distributed graph algorithms, including the construction of a minimum vertex cover, a maximum independent set and the computation of the chromatic number. In addition, we prove quadratic time lower bounds for problems in P, and show the largest possible gap between the running times of deterministic and randomized algorithms in this setting.

Apart from the construction of subgraphs and computation of graph parameters, we also consider the verification of given properties of the graph. To this end, we study the notion of proof-labeling schemes, which allow fast verification using predefined node labels. In this setting, we use nondeterministic communication complexity in order to prove lower bounds on the label sizes, and then suggest that approximation algorithms can help in circumventing these bounds. To achieve this goal, we define a new notion called approximate proof-labeling schemes, provide several such schemes with labels smaller than those necessary for exact proof-labeling schemes, and present a tradeoff between the approximation ratio and the label sizes.

Finally, we turn our attention to a different model of solving graph problems with uncertainty regarding the graph structure. This is the semi-streaming model, where a
single processing unit with a limited memory has to process a graph too large to fit in memory. In this model, we consider the maximum weighted matching problem and provide a \((2 + \epsilon)\)-approximation algorithm for it, matching the state-of-the-art result for unweighted graphs.
Chapter 1

Introduction

In the last few decades we witness a dramatic increase in the use of distributed systems, from computer networks through mobile phone networks to multicore processors. As these systems become larger and more widespread, increasingly sophisticated algorithmic solutions are needed for the new challenges they pose.

Classically, the theory of computing considers a single processing unit, set in one place, which handles information saved in local memory. The new advances in distributed systems bring up many new challenges to computer scientists, in various aspects: algorithms, communication, privacy, storage, power consumption and more. In this thesis we focus on algorithmic problems that arise from the inherent uncertainty in such large-scale distributed systems.

We consider systems composed of many computational units, which communicate with one another by exchanging peer-to-peer messages. We model these systems as simple, undirected graphs, where each node represents a computational unit, and each edge represents a communication link. The communication is assumed to be synchronous: it proceeds in rounds, where in each round each node may exchange messages with all its graph neighbors. When the computation starts, each node has only local information, such as its own unique identifier, and the identifiers of the nodes it is directly connected to. While we assume that nodes have unique identifiers, and that the communication is synchronous, we do not assume the network’s topology is known in advance, in this research we focus on algorithms for learning properties of this topology. The measure of complexity considered is the round complexity, i.e. the number of communication rounds needed in order to solve a specifics problem, whereas the local computation time at each node is generally being neglected.

This model, when assuming no bandwidth restrictions, is known as the LOCAL model [Lin92]. To study systems with limited bandwidth, we sometimes assume that messages are bounded to be of $O(\log n)$ bits, where $n$ is the number of nodes. This allows, e.g., the exchange of a constant number node identifiers in each communication round, and known as the CONGEST model [Pel00]. A third case considered is that of a network augmented with an overly network, which allows logarithmic-size communication between
each pair of nodes in each round, and known as the CLIQUE model \[LPPP05\].

The first type of questions we consider is distance-related problems. In such problems, the communication graph not only represents the topology of the distributed system, but is also the main subject of investigation. The goal of the algorithms is to study simple parameters of the topology of the communication graph, such as the diameter \(D\) of the network, which is the largest distance between a pair of nodes, or all the distances between pairs of nodes.

A basic distance-related problem in this setting is the construction of a BFS tree in an unweighted graph. In the CONGEST model, this task is easily solvable in \(O(D)\) rounds \[Pel00\], and it is not hard to show that \(\Omega(D)\) rounds are also necessary. A more intriguing problem is the construction of many BFS trees in parallel, from a set \(S\) of source nodes. This problem is easily solvable in \(O(|S|D)\) rounds without using parallelization, and Lenzen and Peleg \[LP13b\] showed how to build BFS trees in parallel in this model in only \(O(|S| + D)\) rounds, which is known to be optimal (see also \[HW12\] for an earlier discussion of this problem). In this work, we extend the algorithm of Lenzen and Peleg to work in weighted graphs. That is, for each source node \(s \in S\), we want to construct a tree such that for each node \(v \in V\), the path from \(s\) to \(v\) in the tree is not only the shortest in terms of hops, but also has the minimal sum of weights among all such paths.

Another family of distributed graph problems we study has to do with finding subgraphs of the communication graph that possess some desired properties. In the context of distances in graphs, one type of a desired subgraph is a spanner: a sparse subgraph that has approximately the same distances between pairs of nodes, but much less edges. In this work, we focus mainly on purely additive spanners, in which the distances are stretched by at most an additive factor. We present distributed algorithms for the construction of such spanners, which uses the parallel-BFS algorithm of Lenzen and Peleg in some cases and our new parallel-BFS algorithm for weighted graphs in others. We complement our constructive result with lower bounds on the time needed for the construction of some families of spanners, bounds that use a new communication complexity lower bound and thus also enrich the lower bounds toolbox used for the CONGEST model.

One archetypical distance-related problem is the all-pair-shortest-paths problem (APSP), where each node should learn its distances to all other nodes. In unweighted graphs, this problem can be solved in linear time, e.g. by building a BFS tree from each node \[PRT12\][HW12], which is also known to be optimal \[FHW12\]. On the other hand, finding fast algorithms for the weighted APSP problem in the CONGEST model is an important open question — only recently a sub-quadratic algorithm for it was designed \[Elk17\] and the question whether a linear-time algorithm exists is still widely open. We later discuss the hardness of proving lower bounds for the weighted APSP problem, and also provide interesting solutions for it in the related CLIQUE model.

The fastest sequential algorithms known for distance computation are algorithms that use algebraic techniques, and specifically matrix multiplication. However, these algorithms
largely utilize the fact that all the graph information resides in one place, and it was generally believed that implementing them in a distributed setting will require too much communication, and hence will take too long. Here, we show that matrix multiplication can be implemented in the CLIQUE model, which we then use in order to solve APSP and many other distance-related problems. We also show how to use matrix multiplication to more efficiently solve other well studied problems in the CLIQUE model, such as triangle and 4-cycle counting.

Proving lower bounds for explicit problems in the CLIQUE model is a well known open problem, not susceptible to standard lower bounds techniques. Moreover, it is known to be equivalent, under certain conditions, to the well-studied problem of finding depth lower bounds for Boolean circuits [DKOL14]. Thus, we reduce our attention to some commonly-used families of distance-related algorithms, families that contain all known algorithmic techniques for the problems, and show tight lower bounds for implementing these algorithms in the CLIQUE model.

The distance-related problems discussed so far are known to be global, meaning that they take at least $\Omega(D)$ rounds to solve. Some problems, such as the construction of a BFS tree, take exactly $\Theta(D)$ rounds, while for other problems there is a variety of upper and lower bounds, e.g. $\Theta(\log^* n)$ rounds for 3-coloring a ring, $\tilde{\Theta}(\sqrt{n} + D)$ rounds for constructing a minimum spanning tree and $\Theta(n/\log n)$ rounds for computing unweighted APSP. Every decision problem on unweighted graphs is known to be solvable in $O(n^2)$ rounds, yet prior to our work there were no natural problems for which a super-linear lower bound was known. We remedy this state of affairs, presenting not only super-linear lower bounds, but even quadratic and near-quadratic lower bounds, for decision problems related to the chromatic number of the graph, the maximum cardinality of an independent set and the minimal cardinality of a vertex cover.

Another distributed setting of interest is that of verification: assume an algorithm was executed in the network, and some predicate of the communication graph, e.g. its diameter, was computed. As the network may change over time, we sometimes wish to verify that the computed value is still correct. To this end, we consider the case where the graph nodes, in addition to the claimed value of the predicate, also have labels that help them to verify the computation result. Such an assignment of labels is called a proof-labeling scheme, or a PLS for short [KKP10]. We consider PLSs for distance-related problems, and show a PLS with small labels for claims of the form $D \geq k$, for a constant $k$. Then, we consider PLS for the complementary predicate, $D \leq k$, and use nondeterministic communication complexity in order to show that any PLS for $D \leq k$ must have very large labels. To overcome this lower bound, we present a new concept of approximate proof-labeling schemes (APLS). In an $\alpha$-APLS for $D \leq k$, the verification process must distinguish cases where $D \leq k$ from cases where $D > \alpha k$, and it may err on intermediate values of $D$. We show that as the value of $\alpha$ increases, smaller label sizes suffice.

Another problem we consider in the APLS model is that of maximum weight matching
Here, the graph edges have weights, and the goal is to find a set of non-intersecting edges that have a maximum sum of weights. While a PLS for the problem on restricted families of graphs is known \[\text{KKP10}\], we present a 2-APLS for it that is valid for all graphs, using its primal-dual formulation.

Finally, we turn our attention to a different model of computation. In the semi-streaming model \[\text{FKM}^{+05}\], a single computational unit is used in order to solve a graph-related problem on an \(n\)-node input graph that is given as a stream of edges. In each time unit, one edge is revealed and processed by the algorithm, which uses a memory of only \(O(n \ polylog n)\) bits. After all edges are revealed, the algorithm uses the stored information in order produce an output. Albeit this model is not a model of distributed computation, the main challenge is common to both frameworks: decision making under uncertainty. We consider the MWM problem in the semi-streaming model, and show a simple \((2 + \epsilon)\)-approximation algorithm for it.

**Structure of this Thesis** We continue this introduction by sketching our main results and techniques and surveying some of the relevant background. Chapter 2 presents some definitions and notations common to many parts of this work, while other, more specific definitions are given in the relevant chapters. Chapter 3 presents an algorithm for the construction of weighted BFS trees. Chapter 4 discusses the construction of spanners: Section 4.2 presents algorithms for the construction of purely additive spanners, partially using the aforementioned algorithm, and Section 4.3 gives lower bounds for spanner construction. Chapter 5 is dedicated to lower bounds in the CONGEST model, and presents quadratic and almost-quadratic lower bounds in it. In Chapter 6, we turn to the CLIQUE model, and discuss matrix multiplication and its applications in this model. Chapter 7 presents the APLS model, and provides schemes and lower bounds in it. Finally, in Chapter 8, we turn our attention to the semi-streaming model, where we give a \((2 + \epsilon)\)-approximation algorithm for the MWM problem. Chapter 9 contains a discussion and open questions arising from this work.

### 1.1 Prior Results and Our Results

#### 1.1.1 Weighted BFS Trees

Constructing a breadth-first search (BFS) tree is a central task in many computational settings. In the classic synchronous distributed setting, constructing a BFS tree from a given source is straightforward. Due to its importance, this task has received much attention in additional distributed settings, such as the asynchronous setting (see, e.g., \[\text{Pel00}\] and references therein).

Moreover, at the heart of many distributed applications lies a graph structure that represents the edges of multiple BFS trees, which are rooted at the nodes of a given subset.
$S \subseteq V$, where $G = (V, E)$ is the underlying communication graph. Such a structure is used in distance computations and estimations [HW12, HPRW14, LP15, LP13b], routing table constructions [LP13b], spanner constructions [CKPY17, LP13b], and more. In some cases, different edges of the graph may have different attributes, which can be represented using edge weights. The existence of edge weights has been extensively studied in various tasks, such as finding a minimum spanning tree (MST) in the graph [Elk06, PR99, GP16, HPP15, LPPP05, Awe87, GHS83, DSHK12], finding or approximating lightest paths [Elk17, Nan14, HKN16, EN16, LP13b], finding or approximating a maximum matching [WW04, LPP15, LPR09, DSHK12], and more. However, as far as we are aware, no study addresses the problem of constructing multiple weighted BFS (WBFS) trees, where the goal is not to find the lightest paths from the sources to the nodes, but rather the lightest shortest paths. That is, the path in a WBFS tree from the source $s$ to a node $v$ is the lightest among all the minimum-hops paths from $s$ to $v$.

In this thesis, we provide an efficient construction of multiple WBFS trees from a set of source nodes $S$ in the CONGEST model. Our algorithm completes in $|S| + D$ rounds, which implies that no overhead is needed for incorporating the existence of weights.

The significance of our construction of multiple WBFS trees is illustrated in Section 4.2.8, where we give a distributed algorithm for constructing a $(+6)$-all-pairs spanner, which uses our multiple WBFS construction as a key ingredient. The distributed construction of such spanners seem to be harder than the construction of other spanners, as discussed in Chapter 4 and it was left as an open question in [CKPY17].

1.1.2 The Construction of Graph Spanners

Graph spanners, introduced in the late 1980’s [PS89, PU89a], are fundamental graph structures which are central for many applications, such as synchronizing distributed networks [PU89a], information dissemination [CHKM17], compact routing schemes [Che13a, PU89b, TZ01] and more.

Due to the importance of spanners, the trade-offs between their possible sparsity and stretch have been the focus of a huge amount of literature. Moreover, finding time-efficient constructions of spanners with optimal guarantees has been a major goal for the distributed computing community, with ingenious algorithms given in many studies (see, e.g., [EP04, Elk05, EZ06, DGPV09, BKMP10, BS07, DG08, DGP07, DGPV08, DMP05, Pet10]). The particular type of spanners we are interested at are purely additive spanners, in which the distances are promised to be stretched by no more than an additive term. Despite their importance, distributed constructions of such spanners have been scarce, with the only known construction being a $(+2)$-additive spanner construction with $O(n^{3/2} \log n)$ edges in $O(n^{1/2} \log n + D)$ rounds [HW12]. The absence of distributed constructions of purely additive spanners is explicitly brought into light by Pettie [Pet10], and implicitly mentioned by Derbel et al. [DG07].
Spanner Construction Algorithms  We remedy this state of affairs, by providing a study of the complexity of constructing sparse purely additive spanners in the CONGEST model. We start by providing new, distributed algorithms for the distributed construction of many families of spanners. These algorithms have three main parameters: the stretch, the number of edges, and the running time. One of the most important families of additive spanners is that of all-pairs spanners, where the distance between all the pairs of nodes must be preserved up to a given additive factor. We provide algorithms that construct all-pairs spanners with additive stretch of 2, 4 and 6, with the minimal number of edges up to logarithmic factors. Moreover, our algorithms present a tradeoff between the running time and the number of edge: if one is willing to settle for a larger number of edges, then a shorter running time can be guaranteed.

Most of our algorithms are based on prior, sequential algorithms for spanner construction [Kav15, KV13, Pet09, CGK13, BKMP10, Che13b, ACIM99, DHZ00]. However, these algorithms do not directly translate into distributed algorithms, and distributed techniques such as the parallel construction of BFS trees must be used in order to achieve efficient algorithm.

Constructing (+6)-All-Pairs Spanners  Not for all families of spanners there are sequential algorithms that can be implemented in the CONGEST model. This is the case with the (+6)-all-pairs spanner, for which three different sequential algorithms are known [BKMP10, Woo10, Knu14], but none of them seems to be suitable for a distributed implementation. Thus, we present a new sequential algorithm for the problem, and then implement it in the CONGEST model, using our new parallel WBFS algorithm.

Lower Bounds  In addition to algorithms, we present new lower bounds on the number of rounds required for the construction of additive spanners of specific families. Usually, lower bounds for the CONGEST model are proven using reductions from decision problems in two-party communication complexity function, such as disjointness, but these techniques do not seem to work for lower sounds on spanner construction. Thus, we present a new communication complexity problem, which has to do with computing a relation and not a function, prove a lower bound for it, and then use it to prove a lower bounds for spanner construction.

1.1.3 Quadratic and Almost-Quadratic Lower Bounds

It is well-known and easily proven that many graph problems are global for distributed computing, in the sense that solving them necessitates communication throughout the network. This implies tight $\Theta(D)$ complexities for global problems in the LOCAL model, where all nodes can learn the entire topology in $D$ rounds. In the CONGEST model, the $\Omega(D)$ lower bounds still hold, but sometimes the bandwidth restrictions imply higher lower bounds, and thus these problems are widely studied in the CONGEST model as well.
The trivial complexity of learning the entire topology in the CONGEST model is $O(m)$, and since $m$ can be as large as $\Theta(n^2)$, one of the most basic questions for a global problem is how fast in terms of $n$ it can be solved in the CONGEST model.

Some global problems admit fast $O(D)$-round solutions in the CONGEST model, such as constructing a breadth-first search tree \cite{Pel00}. Some others have complexities of $\tilde{\Theta}(n^{1/2} + D)$, such as constructing a minimum spanning tree, and various approximation and verification problems \cite{FHW12,DSHK12,KKP11,PR00,PRT12,HPRW14}. Some problems are yet harder, with complexities that are near-linear in $n$ \cite{PRT12,ACK16,LP13,FHW12,HW12}. For some problems, no $O(n)$ solutions are known and they are candidates to being even harder than the ones with linear-in-$n$ complexities.

**Super-Linear Lower Bounds** A major open question about global graph problems in the CONGEST model is whether natural graph problems for which a super-linear number of rounds is required indeed exist. In this work, we answer this question in the affirmative. That is, our conceptual contribution is that there exist super-linearly hard problems in the CONGEST model. In fact, the lower bounds that we prove are as high as quadratic in $n$, or quadratic up to logarithmic factors, and hold even for networks of a constant diameter. Our lower bounds also imply linear and near-linear lower bounds for the CLIQUE-Bcast model.

High lower bounds for the CONGEST model may be obtained rather artificially, by forcing large inputs and outputs that must be exchanged. However, we emphasize that all the problems for which we show our lower bounds can be reduced to simple decision problems, where each node needs to output a single bit. All inputs to the nodes, if any, consist of edge weights that can be represented by $O(\text{polylog } n)$ bits.

Technically, we prove a lower bound of $\Omega(n^2 / \log^2 n)$ on the number of rounds required for computing an exact minimum vertex cover, which also extends to computing an exact maximum independent set (Section 5.3.1). This is in stark contrast to the recent $O(\log \Delta / \log \log \Delta)$-round algorithm of \cite{BCS16} for obtaining a $(2 + \epsilon)$-approximation to the minimum vertex cover. Similarly, we give an $\Omega(n^2 / \log^2 n)$ lower bound for 3-coloring a 3-colorable graph, which extends also for deciding whether a graph is 3-colorable, and also implies the same hardness for computing the chromatic number $\chi$ or computing a $\chi$-coloring (Section 5.3.2). These lower bounds hold even for randomized algorithms which succeed with high probability.

**Bounds for Problems in P** An immediate question that arises is whether only NP-hard problems are super-linearly hard in the CONGEST model. In Section 5.4, we provide a negative answer to such a postulate, by showing two simple problems that admit polynomial-time sequential algorithms, but in the CONGEST model require $\Omega(n^2)$ rounds (identical subgraph detection) or $\Omega(n^2 / \log n)$ rounds (weighted cycle detection). The latter also holds for randomized algorithms, while for the former we show a randomized
algorithm that completes in $O(D)$ rounds, providing the strongest possible separation between deterministic and randomized complexities for global problems in the CONGEST model.

**The Weighted APSP Problem** Finally, we address the intriguing open question of the complexity of computing exact weighted all-pairs-shortest-paths (APSP) in the CONGEST model. While the complexity of the unweighted version of APSP is $\Theta(n/\log n)$ \cite{FW12,HFQ16}, the complexity of weighted APSP remains largely open, and only recently the first sub-quadratic algorithm was given in \cite{Elk17}. With the current state-of-the-art, this problem could be considered as a suspect for having a super-linear complexity in the CONGEST model. While we do not pin-down the complexity of weighted APSP, we provide a truly linear lower bound of $\Omega(n)$ rounds for it, which separates its complexity from that of the unweighted case. Moreover, we argue that it is not a coincidence that we are currently unable to show super-linear lower bound for weighted APSP, by formally proving that the commonly used framework of reducing a 2-party communication problem to a problem in the CONGEST model cannot provide a super-linear lower bound for weighted APSP, regardless of the function and the graph construction used (Section 5.5). This implies that obtaining any super-linear lower bound for weighted APSP provably requires a new technique.

### 1.1.4 Computing Distances Using Algebraic Techniques

Algebraic methods have become a recurrent tool in centralized algorithms, employing a wide range of techniques (e.g., \cite{Bjo14,BH14,BHKK07,BHK09,BKK13,BKK14,BCKN13,CKN13,CNP11,CL07,EG04,FLR12,FLS14,Kou08,KLL11,LN10,NP85,Wil09,VWW13}). In this work, we bring techniques from the algebraic toolbox to the aid of distributed computing, by leveraging fast matrix multiplication in the CLIQUE.

The CLIQUE masks away the effect of *distances* on the computation and focuses on the limited *bandwidth*. As such, it has been recently gaining increasing attention \cite{PS14,DL12,DKO13,PPP05,Nan14,LI11,PT11,HSI13,HP14}, in an attempt to understand the relative computational power of distributed computing models.

The key insight of this work is that matrix multiplication algorithms from parallel computing can be adapted to obtain an $O(n^{1-2/\omega})$ round matrix multiplication algorithm in the CLIQUE, where $\omega < 2.3728639$ is the matrix multiplication exponent \cite{LG14}. Combining this with well-known centralized techniques allows us to use fast matrix multiplication to solve various combinatorial problems, immediately giving $O(n^{0.158})$-time algorithms in the CLIQUE for many classical graph problems. Indeed, while most of the techniques we use in this work are known beforehand, their combination gives significant improvements over the best previously known upper bounds.

More specifically, we show fast algorithms in the CLIQUE that compute unweighted APSP in $O(n^{0.158})$ rounds, a $(1 + \epsilon)$-approximation algorithm for weighted APSP with the
same running time, and exact weighted APSP algorithm running for $O(n^{1/3})$ rounds. The only prior APSP algorithm was a $(2 + \epsilon)$-approximation algorithm running for $\tilde{O}(n^{1/2})$ rounds. In addition, we show that $O(n^{0.158})$ rounds are sufficient for counting triangles and 4-cycles, which improves upon the prior $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{1/2})$ rounds algorithms, respectively [DLP12]. In $O(n^{0.158})$ rounds, it is also possible to compute the girth of the graph, i.e. the length of the shortest cycle in it, a problem that was not considered beforehand in this model.

Matrix Multiplication on a Congested Clique As a basic primitive, we consider the computation of the product $P = ST$ of two $n \times n$ matrices $S$ and $T$ in the CLIQUE model. We will tacitly assume that the matrices are initially distributed so that node $v$ has row $v$ of both $S$ and $T$, and it will receive row $v$ of $P$ in the end. Recall that the matrix multiplication exponent $\omega$ is defined as the infimum over $\sigma$ such that the product of two $n \times n$ matrices can be computed with $O(n^\sigma)$ arithmetic operations; it is known that $2 \leq \omega < 2.3728639$ [LG14], and it is conjectured, though not unanimously, that $\omega = 2$.

Our main technical result is proving that the product of two $n \times n$ matrices can be computed in the CLIQUE in $O(n^{1/3})$ rounds over semirings, and in $O(n^{1-2/\omega+\epsilon})$ rounds over rings (for any constant $\epsilon > 0$). This result is proven by adapting known parallel matrix multiplication algorithms for semirings [ABG+95, McC95] and rings [LD95, McC96, Tis99] to the CLIQUE model, via the routing technique of Lenzen [Len13]. In fact, with little extra work one can show that the resulting algorithm is also oblivious, that is, the communication pattern is predefined and does not depend on the input matrices. Hence, the oblivious routing technique of Dolev et al. [DLP12] suffices for implementing these algorithms.

Lower Bounds for Matrix Multiplication Our results are optimal in the sense that for any sequential matrix multiplication implementation, no scheme simulating it on the CLIQUE can give a faster algorithms than those mentioned above; this follows from known results for parallel matrix multiplication [BDHS12, ITT04, ACS90, Tis98]. Moreover, we note that for the CLIQUE-Bcast model, recent lower bounds [HP13] imply that matrix multiplication requires $\tilde{\Omega}(n)$ rounds.

1.1.5 Approximate Proof-Labeling Schemes

Verification of a given property in decentralized systems finds applications in various domains, such as, checking the result obtained from the execution of a distributed program [APV91, FRT13], establishing lower bounds on the time required for distributed approximation [DSHK+12], estimating the complexity of logic required for distributed run-time verification [FRT14], general distributed complexity theory [FKPI3], and the construction of self stabilizing algorithms [BFP14, KKM11].
In the distributed setting, a network configuration $G_s$ is represented by an underlying graph and a state assignment. The state assignment is the state of each node, which can contain a unique identifier, edge weights, a specified subset of incident edges, an output of a distributed algorithm and more. In order to verify that a network configuration has a specified property, nodes exchange messages along the edges and output either $\text{true}$ or $\text{false}$ depending on whether the local configuration is consistent with a legal state of the network. The distributed verification process is correct if all nodes return $\text{true}$ on legal configurations, and at least one node returns $\text{false}$ on every illegal configuration. Some properties are local by nature and easy to verify, for example, whether a specified subset of edges is a matching in the graph. However, many other properties cannot be verified in less than diameter time, even if message size and local computational power are unbounded, for example, whether a specified matching is of maximum cardinality.

In order to cope with strong time lower bounds, Korman, Kutten, and Peleg [KKP10] have introduced a computational model, called proof-labeling schemes (PLSs), where nodes are given auxiliary global information in the form of labels. A proof-labeling scheme for a predicate $\mathcal{P}$ consists of a prover and a verifier. For every legal state of the network, the prover assigns a label to every node. The verifier is a distributed algorithm, which uses one round of communication to exchange labels between neighbors and output $\text{true}$ or $\text{false}$ at each node, as a function of the state and label of the node and the labels it receives from its neighbors. A PLS satisfies completeness if for every legal configuration, with the labels assigned by the prover, all nodes output $\text{true}$, and it satisfies soundness if for every illegal configuration and every label assignment, some node outputs $\text{false}$.

The complexity measure used in evaluating the quality of a PLS is the maximum size of a label, which is called the proof size. It is known that, for every sequentially decidable graph property, there exists a PLS with proof size $O(m \log n)$ [KKP10, GST16, BFP15]. For some properties, lower bounds on the proof size have been proven in this model, for example $\Omega(\log n)$ for verification of a spanning-tree [KKP10] and bi-connectivity [BFP15], $\Omega(n^2/\log n)$ for verifying that the graph is not 3-colorable [GST16], and $\Omega(\log n \log W)$ for verification of a minimum-weight spanning-tree [KK07], assuming that the maximal edge-weight $W$ satisfies $\log^{1+\epsilon} n < W$ for some constant $\epsilon > 0$.

As in the computational framework, variations of the model may allow us to break known lower bounds. It has been suggested to use super-constant number of rounds in verification [BOR16, KKM11]. In the former, a linear reduction of proof size is proven for acyclicity and the universal scheme. In the latter, they present a scheme for minimum-weight spanning-tree with $O(\log n)$ proof size and $O(\log^2 n)$ rounds. In [BFP15] it was suggested to distinguish between labels and communication in the verification process, and to use randomization in order to reduce the communication complexity of verification. They show an exponential reduction in the communication complexity of every scheme at the cost of increasing the proof size by a factor of the maximum degree.
PLSs for Diameter  Yet, some properties are still harder. In Section 7.3 we show that any PLS for $D \leq k$ must have labels of $\Omega(n)$ bits where $k \in \mathbb{N}$ is a constant. A natural way to circumvent this lower bound is through approximation, e.g., by defining a 2-approximation for the problem by the predicate $D \leq 2k$, but the lower bound obviously applies to this predicate as well.

Approximate PLS  Inspired by the above example, we present and investigate a new concept of approximate proof-labeling schemes (APLSs for short) for optimization problems. Let $\psi, \varphi : \mathcal{F} \to \mathbb{N}$ be two functions from a family of configurations to the natural numbers. Assume that we are interested in verifying for every $G_s \in \mathcal{F}$ whether $\psi(G_s) \leq \varphi(G_s)$, and let $\alpha > 1$ be the approximation ratio. If $\psi(G_s) \leq \varphi(G_s)$ then there is an assignment of labels such that all nodes output true, and if $\psi(G_s) > \alpha \varphi(G_s)$ then for every label assignment at least one node outputs false; if $\varphi(G_s) < \psi(G_s) \leq \alpha \varphi(G_s)$, we do not have any promise. Put differently, we are promised that if all nodes output true, then $\psi(G_s) \leq \alpha \varphi(G_s)$, i.e., the approximation holds.

The concept of APLS indeed allows us to find schemes with shorter labels: we show a 2-APLS for $D \leq k$ with proof size of only $O(\log n)$ bits, and a $(3/2)$-APLS for $D \leq k$ with proof size of $O(\sqrt{n \log^2 n})$ bits. In addition, we consider APLS for the maximum weight matching problem, and show a 2-APLS for the problem, improving upon prior work [KKP10, GS16].

1.1.6 Maximum Weight Matching in the Semi-Streaming Model

In the last chapter of this thesis we take a detour and consider a different model of computation, the semi-streaming model. This model helps us to study large-scale systems, where the complete information about the system is too large to fit in the memory, and thus some of the processing of the input must be made while reading it. This fact creates uncertainty regarding the input, which is a similarity point between the semi-streaming model and distributed models.

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, GZ16], 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 [Edm65], 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 this model was first presented [FKM+05], and probably the most studied problem in this model since.
In the first algorithms for the MWM problem in the semi-streaming model, 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 \cite{BES85, BBF01} 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 work on the semi-streaming model \cite{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.
Chapter 2

Preliminaries

2.1 Graph Theoretic Concepts

All graphs in this work are simple, connected and undirected. A graph can be unweighted, \( G = (V, E) \), or weighted \( G = (V, E, w) \) with \( w : E \rightarrow \{0, \ldots, W\} \), in which case we assume \( W \in \text{poly}(n) \). Given a path \( \rho \) in a weighted graph \( G \), we use \( |\rho| \) to denote the length of \( \rho \), which is the number of edges in it, and \( w(\rho) \) to denote the weight of the path, which is the sum of its edge-weights.

The distance between two nodes \( u, v \) in a graph \( G \), denoted \( \delta_G(u, v) \), is the minimum length of a path in \( G \) connecting \( u \) and \( v \). A BFS tree rooted at a node \( r \) is a tree composed of shortest paths from \( r \) to each other node in the graph [CSRL01].

2.2 Models of Computation

The Congest Model The distributed model we assume is the well-known CONGEST model [Pel00]. This model is composed of a system that consists of a set of \( n \) computational units, who exchange messages according to an undirected communication graph \( G = (V, E) \), \( |V| = n \), where nodes represent the computational units and edges the communication links. Each node has a unique identifier which can be encoded using \( O(\log n) \) bits. We use \( D \) to denote the diameter of \( G \), i.e. the maximal distance between two nodes in it.

When the computation starts, each node knows its own identifier and the identifiers of its neighbors; when there are other graph parameters involved in the computation, such as edge weights, a set \( S \) of nodes or a set \( \mathcal{P} \) of node-pairs, it also knows the weights of the edges it is adjacent on, if it belongs to \( S \), or all the pairs in \( \mathcal{P} \) it belongs to. The computation proceeds in rounds, where in each round each node sends an \( O(\log n) \)-bit message to each of its neighbors, receives a message from each neighbor, and locally performs a computation. We use the number of rounds as our complexity measure, while ignoring the local computation time; however, in our algorithms all local computations take polynomial time. When the computation ends, each node knows its own local view.
of the output; this could be its distance to a given source node or distances to all other graph nodes, or, which of its neighbors is also its neighbor in the spanner \( H = (V, E') \) generated by the algorithm.

We assume the nodes have some bounds on \( n \), the total number of nodes on the graph. Most of the problems we discuss are global, in the sense that solving them takes \( \Omega(D) \) rounds; there are standard techniques for computing \( n \) and finding a 2-approximation for \( D \) in \( O(D) \) rounds \([\text{Pel00}]\), so we further assume these values are known to all nodes.

We use standard asymptotic notation, \( O, o, \Theta, \omega \) and \( \Omega \) (see, e.g. \([\text{CSRL01}]\)) and use \( \tilde{O} \) and \( \tilde{\Omega} \) to hide polylog factors. We say that an event occurs with high probability (w.h.p) if it occurs with probability \( \frac{1}{n^c} \), for some constant \( c > 0 \).

**The Clique Model**  The *congested clique* model, denoted CLIQUE henceforth, is a variant of the CONGEST model, where the input graph \( G \) is no longer the communication graph: while each node still represents a computational unit, a pair of nodes can communicate directly even if they are not adjacent in graph \( G \).

**The Clique-Bcast model**  The *broadcast congested-clique* model, denoted CLIQUE-Bcast \([\text{DKO14}, \text{HP15}]\), is a variant of the CLIQUE model where in each round each node must send the same \( O(\log n) \)-bit message to all nodes. This model is not one of our main topics of research, but we sometimes mention results the concerns it as well.

**The Local model**  Another classical model of distributed graph algorithms is the Local model \([\text{Lin92}, \text{Pel00}]\). This model resembles the CONGEST model, but it does not have bandwidth limitations. Under these assumptions, any graph problem is solvable in \( O(D) \) rounds, as in this this time all nodes can learn all the graph structure, and no uncertainty is left. In this thesis, we mainly treat problems that take \( \Omega(D) \) rounds to solve in the Local model, and study the extra time needed to solve them in the CONGEST and CLIQUE models due to bandwidth limitations.

### 2.3 Communication Complexity

#### 2.3.1 Protocols and Definitions

In a two-party communication complexity problem \([\text{KN97}]\), there is a function \( f : \{0, 1\}^K \times \{0, 1\}^K \rightarrow \{\text{true, false}\} \), and two players, Alice and Bob, who are given two input strings, \( x, y \in \{0, 1\}^K \), respectively, that need to compute \( f(x, y) \). The communication complexity of a protocol \( \pi \) for computing \( f \), denoted \( CC(\pi) \), is the maximal number of bits Alice and Bob exchange in \( \pi \), taken over all values of the pair \((x, y)\). The deterministic communication complexity of \( f \), denoted \( CC(f) \), is the minimum over \( CC(\pi) \), taken over all deterministic protocols \( \pi \) that compute \( f \).
In a randomized protocol $\pi$, Alice and Bob may each use a random bit string. A randomized protocol $\pi$ computes $f$ if the probability, over all possible bit strings, that $\pi$ outputs $f(x,y)$ is at least $2/3$. The randomized communication complexity of $f$, $CC^R(f)$, is the minimum over $CC(\pi)$, taken over all randomized protocols $\pi$ that compute $f$.

In a nondeterministic protocol $\pi$, Alice and Bob may each use an auxiliary bit string, and run a deterministic protocol that uses their inputs and auxiliary strings. We say a nondeterministic protocol $\pi$ computes the $\text{true}$ case of $f$ if for every input pair $(x,y)$ such that $f(x,y) = \text{true}$ there is an assignment of auxiliary strings to the players such that both output $\text{true}$, while for every input pair such that $f(x,y) = \text{false}$ and any assignment of auxiliary strings to the players, at least one player outputs $\text{false}$. The nondeterministic communication complexity of the $\text{true}$ case of $f$ is the minimum over $CC(\pi)$, taken over all nondeterministic protocols $\pi$ that compute the $\text{true}$ case of $f$. A protocol that computes the $\text{false}$ case of $f$ and the nondeterministic communication complexity of the $\text{false}$ case of $f$ are analogously defined.

### 2.3.2 Typical Problems

In the Set Disjointness problem ($\text{DISJ}_K$), the function $f$ is $\text{DISJ}_K(x,y)$, whose output is $\text{false}$ if there is an index $i \in \{0,\ldots,K-1\}$ such that $x_i = y_i = 1$, and $\text{true}$ otherwise. In the Equality problem ($\text{EQ}_K$), the function $f$ is $\text{EQ}_K(x,y)$, whose output is $\text{true}$ if $x = y$, and $\text{false}$ otherwise.

Both the deterministic and randomized communication complexities of the $\text{DISJ}_K$ problem are known to be $\Omega(K)$ [KN97, Example 3.22]. The deterministic communication complexity of $\text{EQ}_K$ is in $\Omega(K)$ [KN97, Example 1.21], while its randomized communication complexity is in $\Theta(\log K)$ [KN97, Example 3.9]. (See also [Yao79] for the first definitions and results regarding communication-complexity, and [Raz92] for the lower bound for randomized algorithms for the DISJ function.)

As per the nondeterministic communication complexities of the $\text{DISJ}_K$ problem we note that if $\text{DISJ}(x,y) = \text{false}$ and Alice and Bob both use the index $i$ such that $x_i = y_i = 1$ as an auxiliary string, then they only need to exchange $O(\log K)$, to verify they have the same index, so the nondeterministic communication complexity of the $\text{false}$ case of the problem is $O(\log K)$. On the other hand, a celebrated result asserts that when $\text{DISJ}(x,y) = \text{true}$, Alice and Bob must communicate $\Omega(K)$ bits even with an optimal assignment of auxiliary strings, i.e. nondeterminism cannot help Alice and Bob in asymptotically minimizing the communication, and the nondeterministic communication complexity of the $\text{true}$ case is in $\Omega(K)$ (this follows, e.g., from [KN97, Example 1.23 and Definition 2.3]). We do not address the nondeterministic communication complexities of equality in this work.
Chapter 3
Parallel Construction of Weighted BFS Trees

Given a source node $s$ in a weighted graph $G$, we define a *weighted breadth first search tree (WBFS)* as a tree which consists of the lightest among all shortest paths from $s$ to $v$, for every node $v$. We present a distributed algorithm for the construction of multiple WBFS trees from a set $S$ of sources in the CONGEST model, which completes in $|S| + D$ rounds.

At a high level, our approach is to generalize the algorithm of Lenzen and Peleg [LP13b] in order to handle weights. In [LP13b], the messages are pairs that consist of a source node and a distance, which are prioritized by the distance traversed so far. When incorporating weights into this framework it makes sense to use triplets instead of pairs, where each triplet also contains the weight of the respective path. However, it may be that a node $v$ needs to send multiple messages that correspond to the same source and the same distance but contain different weights, since congestion over edges may cause the respective messages to arrive at $v$ in different rounds and, in the worst case, in a decreasing order of weights. The challenge in generalizing this framework therefore lies in guaranteeing that despite the need to consider weights, we can carefully choose a total order to prioritize triplets, such that not too many messages need to be sent, allowing us to handle congestion. Our construction and its proof appear in Section 3.1, giving the following.

**Theorem 3.1.** Given a weighted graph $G = (V, E, w)$ and a set of nodes $S \subseteq V$, there exists an algorithm for the CONGEST model, which constructs a WBFS tree rooted at $s$, for every $s \in S$, in $|S| + D$ rounds.

**Related work** Algorithms for constructing multiple (unweighted) BFS trees in the CONGEST model were suggested in [LP13b] and in [HW12], requiring $O(|S| + D)$ rounds for a set of sources $S$. Both algorithms start the construction of all the BFS trees simultaneously, and proceed by transferring messages containing the source of a BFS tree and the distance the message has traversed so far. The algorithms differ in how they order message deliveries when several messages need to be sent over an edge at the same round.
We base our multiple WBFS construction on the \([\text{LP13b}]\) algorithm, in which messages sent by a node are prioritized by the distance they traversed so far, with a preference to messages that traversed smaller distance. The \([\text{HW12}]\) algorithm, which we cannot use for our construction, prioritizes messages by the identity of the root, and an edge only delivers a message in a single direction in each round.

### 3.1 Multiple Weighted BFS Trees

In this chapter, we are interested in a weighted BFS tree, which consists of all lightest shortest paths from the root, formally defined as follows.

**Definition 3.2.** Given a connected, weighted graph \(G = (V, E, w)\) and a node \(r \in V\), a weighted BFS tree (WBFS) for \(G\) rooted at \(r\), is a spanning tree \(T\) of \(G\) satisfying the following properties:

1. For each \(v \in V\), the path from \(r\) to \(v\) in \(T\) is a shortest path in \(G\) between \(r\) and \(v\).

2. For each \(v \in V\), no shortest path from \(r\) to \(v\) is lighter than the path from \(r\) to \(v\) in \(T\).

We emphasize that this is different than requiring a subgraph containing all lightest paths from the root. It is easy to see that a WBFS tree always exists, by going over the nodes in an order of non-decreasing distances from the source \(r\), which is initialized with \(w(r) = 0\), and for each node \(v\) adding to the WBFS tree the edge \(\{v, u\} \in E\) that minimizes \(w(u) + w(v, u)\).

In the CONGEST model, the problem of finding a WBFS tree requires each node to know its parent in the WBFS tree. This allows the node to send messages to the source node through the lightest among all shortest paths. When there are multiple sources, each node should know the parent leading to each of the sources in \(S\).

We define the data structures used for representing multiple WBFS trees. Given a node \(v \in V\), the S-proximity-list (or proximity list for short) of \(v\), noted \(\text{PL}^*_v\), is an ascending lexicographically ordered list of triples \((d(s, v), s, w(s, v))\), where \(d(s, v)\) and \(w(s, v)\) are the length and weight of the path from \(s\) to \(v\) in \(T_s\). Two triples are ordered such that \((d(s, v), s, w(s, v)) < (d(t, v), s, w(t, v))\) if \(d(s, v) < d(t, v)\), or \(d(s, v) = d(t, v)\) and \(s < t\). Comparisons between \(s\) and \(t\) are decided by an arbitrary predefined order over the node identifiers.

The S-path-map (or path-map for short) of \(v\) is a mapping from each source \(s \in S\) to the parent of \(v\) in \(T_s\), noted by \(\text{PM}^*_v\). The list \(\text{PM}^*_v\) is sorted with respect to the order of \(\text{PL}^*_v\), such that the top records of \(\text{PM}^*_v\) belong to sources closest to \(v\).

Our algorithm for constructing multiple WBFS trees in the CONGEST model is based on carefully extending the distributed Bellman-Ford-based algorithm of Lenzen and Peleg \([\text{LP13b}]\). Algorithm 1 constructs a WBFS tree from each node \(s\) in a set \(S\). The
The algorithm is designed to run in the CONGEST model of computation in $|S| + D$ rounds, where $D$ is the unweighted diameter of the input graph.

The algorithm builds the WBFS trees by gradually updating the proximity list and the path map of each node. Each round is composed of two phases: updating the neighbors about changes in the proximity list, and receiving updates from other nodes. The path map is only used by the current node, and therefore any changes to it are not sent.

Ideally, each node would update its neighbors of all changes made to its proximity list. However, due to bandwidth restrictions, a node can not send the entire list in each round. Therefore, at each round each node sends to all of its neighbors the lexicographically smallest triplet in its proximity list that it has not yet sent, while maintaining a record noting which triplets have been sent and which are waiting. Each triplet is only sent once, though a node can send multiple updates regarding its proximity to a given source.

The node uses the messages received in the current round in order to update its proximity list and path map for the next round. A triplet $(d_s, s, w_s)$ received by a node $v$ from a neighbor $u$ represents the length $d_s$ and weight $w_s$ of some path $\rho$ from $s$ to $u$ in the graph. The node $v$ then considers the extended path $\rho' = \rho \cup v$ from $s$ to $v$, compares it to its currently known best path from $s$ to $v$, and updates the proximity list and path map in case a shorter path has been found, or a lighter path with the same length.

To prove correctness, we generalize the proof of [LP13b] to handle weights, and show that our algorithm solves the weighted $(s, d, k)$-detection problem, defined as follows.

**Definition 3.3.** Given a weighted graph $G = (V, E, w)$, a subset $S \subseteq V$ of source nodes, and a node $v \in V$, let $PL_v^*$ denote the $S$-proximity-list and let $PM_v^*$ denote the path map of the node $v$. The weighted $(S, d, k)$-detection problem requires that each node $v \in V$ learns the first $\min(k, \lambda_d^v) \leq d$ entries of $PL_v^*$ and $PM_v^*$, where $\lambda_d^v$ is the number of sources $s \in S$ such that $d_s \leq d$.

Given a node $v$, let $PL_v$ and $PM_v$ be the proximity list and path map of $v$, as obtained by the algorithm. We prove that our algorithm obtains the correct values of $PL_v$, i.e. $PL_v^{(\infty)} = PL_v^*$, where $PL_v^{(\infty)}$ is the value of $PL_v$ at the end of our algorithm. Since the records of $PM_v$ are updated under the same conditions as the records of $PL_v$, the correctness of $PM_v$ at the end of the algorithm with respect to $PM_v^*$, immediately follows.

We start by showing that if there was no bound on the number of rounds, then the values of $PL_v$ would eventually converge to the true values of $PL_v^*$. In order to formalize this statement, we denote $PL_v^{(r)}$ the state of the list $PL_v$ at the beginning of round $r \in \mathbb{N}$ of the algorithm.

**Lemma 3.4.** For any weighted graph $G = (V, E, w)$ and any set $S \subseteq V$, there exists a round $r_0 \in \mathbb{N}$ such that no node $v \in V$ sends messages or modifies $PL_v$ after round $r_0$. Moreover, $PL_v^{(r_0)} = PL_v^*$, i.e., for every $(d_s, s, w_s) \in PL_v^{(r_0)}$, it holds that $d_s = d(v, s)$ and $w_s = \min \{w(\rho) \mid \rho \text{ connects } v \text{ with } s, \text{ and } |\rho| = d_s\}$.
there exists a path $\rho$ and eventually no node adds further triplets to its lists or sends additional messages. As the graph is finite, the number of paths is bounded, and hence does not insert a message into $PL_s$ bounded by the number of paths from $d_u$ node $v$ for any round $r$. Therefore, if we show the existence of a round $r_0$ such that $PL_v(r_0)$ contains all paths from $s$ to $v$, we can conclude that no message is sent in round $r_0$, and hence $PL_v(r_0) = PL_v(r_0)$. This claim is applied inductively, concluding that for any round $r > r_0$, it holds that $PL_v(r) = PL_v(r_0)$. We prove the existence of the round $r_0$ by showing that the number of messages that are sent by each node $v \in V$ is finite.

Each triple $(d_s, s, w_s)$ is inserted into $PL_v$ when a message is received from another node $u \in V$, and such a message implies that a path from $s$ to $v$ through $u$ of length $d_s$ and weight $w_s$ exists. Thus, the number of messages $(d_s, s, w_s)$ sent by a node $v$ is bounded by the number of paths from $s$ to $v$ of length $d_s$. Furthermore, the algorithm does not insert a message into $PL_v$ if it has already inserted a lexicographically smaller message from the same source. As the graph is finite, the number of paths is bounded, and eventually no node adds further triplets to its lists or sends additional messages.

It remains to show that for all $s \in S$ and $v \in V$ it holds that $(d(s, v), s, w(s, v)) \in PL_v(r_0)$.

First, we show that if a triplet $(d_s, s, w_s)$ is added to $PL_v$ in some round $r$, then there exists a path $\rho$ from $s$ to $v$ such that $d_s = |\rho|$ and $w_s = w(\rho)$. For each source

---

Algorithm 3.1: Weighted distributed Bellman-Ford algorithm for node $v$

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$L_v \leftarrow ()$</td>
</tr>
<tr>
<td>2</td>
<td>for $s \in S$ do</td>
</tr>
<tr>
<td>3</td>
<td>$PM_v(s) \leftarrow \bot$</td>
</tr>
<tr>
<td>4</td>
<td>if $v \in S$ then</td>
</tr>
<tr>
<td>5</td>
<td>$PL_v \leftarrow ((0, v, 0))$</td>
</tr>
<tr>
<td>6</td>
<td>$sent_v(0, v, 0) \leftarrow \text{FALSE}$</td>
</tr>
<tr>
<td>7</td>
<td>for $</td>
</tr>
<tr>
<td>8</td>
<td>if $\exists (d_s, s, w_s) \in PL_v$ such that $sent_v(d_s, s, w_s) = \text{FALSE}$ then</td>
</tr>
<tr>
<td>9</td>
<td>$(d_s, s, w_s) \leftarrow \min {(d_t, t, w_t) \in PL_v \text{ such that } sent_v(d_t, t, w_t) = \text{FALSE}}$</td>
</tr>
<tr>
<td>10</td>
<td>send $(d_s, s, w_s)$ to all neighbors</td>
</tr>
<tr>
<td>11</td>
<td>$sent_v(d_s, s, w_s) \leftarrow \text{TRUE}$</td>
</tr>
<tr>
<td>12</td>
<td>for received $(d_s, s, w_s)$ from $u \in V$ do</td>
</tr>
<tr>
<td>13</td>
<td>$d_s \leftarrow d_s + 1$</td>
</tr>
<tr>
<td>14</td>
<td>$w_s \leftarrow w_s + w(u, v)$</td>
</tr>
<tr>
<td>15</td>
<td>if $\exists (d'_s, s, w'_s) \in PL_v$ such that $(d'_s &lt; d_s$ or $(d'_s = d_s$ and $w'_s &lt; w_s))$ then</td>
</tr>
<tr>
<td>16</td>
<td>$PL_v \leftarrow PL_v \setminus {(s, s, s)}$</td>
</tr>
<tr>
<td>17</td>
<td>$PL_v \leftarrow PL_v \cup {(d_s, s, w_s)}$</td>
</tr>
<tr>
<td>18</td>
<td>$PM_v(s) \leftarrow u$</td>
</tr>
<tr>
<td>19</td>
<td>$sent_v(d_s, s, w_s) \leftarrow \text{FALSE}$</td>
</tr>
</tbody>
</table>

Proof. In each iteration of the algorithm, each node $v \in V$ sends to all of its neighbors the first triplet $(d_s, s, w_s)$ such that $sent_v(d_s, s, w_s) = \text{FALSE}$. Each triplet received is sent at most once. Therefore, if we show the existence of a round $r_0$ where for each $v \in V$, all messages in $PL_v(r_0)$ have been sent in previous rounds, it implies that no message is sent in round $r_0$, and hence $PL_v(r_0) = PL_v(r_0)$. This claim is applied inductively, concluding that for any round $r > r_0$, it holds that $PL_v(r) = PL_v(r_0)$. We prove the existence of the round $r_0$ by showing that the number of messages that are sent by each node $v \in V$ is finite.
when a lexicographically smaller triplet \((d, s, w)\) is inserted into \(PL_v\). The crux of the correctness of part (i) is in the method used by our algorithm for managing the list \(PL_v\). According to our algorithm, triplets are not removed from \(PL_v\) when they are sent. The only case in which a triplet \((d, s, w)\) is removed from \(PL_v\) is when a lexicographically smaller triplet \((d', s, w')\) is added to the list instead. When this occurs in round \(r\), it holds that \(t_v^{(r)}(d, s, w) \geq t_v^{(r+1)}(d', s, w')\), since the new triplet is
Lemma 3.5 implies that as the algorithm progresses, messages at higher indexes of the proximity list are sent and updated. This can be used to obtain an upper bound on the round in which a triplet at a certain index of the proximity list can be sent or received, as formalized by the next lemma.

**Lemma 3.6.** For every node \( v \in V \) and round \( r \in \mathbb{N} \) of Algorithm I,

(i) \( v \) does not send a message \( (d_s, s, w_s) \) with

\[
d_s + \ell_v^{(r)}(d_s, s, w_s) < r
\]

in round \( r \), and

(ii) \( v \) does not add to \( PL_v \) a triplet \( (d_s, s, w_s) \) with

\[
d_s + \ell_v^{(r+1)}(d_s, s, w_s) \leq r
\]

in round \( r \), where \( \ell_v^{(r+1)}(d_s, s, w_s) \) is the index of the new entry \( (d_s, s, w_s) \) after insertion, in the beginning of the next round.

**Proof.** We start by showing that, for a given round \( r \in \mathbb{N} \), if Lemma 3.6(i) holds for all nodes then Lemma 3.6(ii) holds as well. Consider a triplet \( (d_s, s, w_s) \) sent in round \( r \) from a node \( u \in V \) to its neighbor \( v \in V \), resulting in the addition of the triplet \( (d'_s, s, w'_s) \) to \( PL_v \) at the end of the round, where \( d'_s = d_s + 1 \) and \( w'_s = w_s + w(u, v) \). Lemma 3.6(i) implies that \( d_s + \ell_u^{(r)}(d_s, s, w_s) \geq r \), and by Lemma 3.5(ii) we have that \( \ell_v^{(r+1)}(d'_s, s, w'_s) \geq \ell_u^{(r)}(d_s, s, w_s) \). As \( d'_s > d_s \), we have

\[
d'_s + \ell_v^{(r+1)}(d'_s, s, w'_s) > d_s + \ell_u^{(r)}(d_s, s, w_s) \geq r,
\]

which implies Lemma 3.6(ii).

Next, we prove by induction that both parts of the lemma hold. In round 1, Lemma 3.6(i) holds trivially, since by definition \( \ell_v^{(1)}(d_s, s, w_s) \geq 1 \) for all \( v, s, d_s \) and \( w_s \). Assume that Lemma 3.6 holds for round \( r - 1 \). We show the lemma holds for round
r. Since Lemma 3.6(i) implies Lemma 3.6(ii), it is sufficient to show that every message \((d_s, s, w_s)\) sent by some node \(v \in V\) in round \(r\) satisfies \(d_s + \ell_v^{(r)}(d_s, s, w_s) \geq r\).

Observe that if \((d_s, s, w_s)\) is sent by a node \(v\) in round \(r\), then the triplet must have been added to \(\text{PL}_v\) in some round \(r' \leq r - 1\). If \(r' = r - 1\), according to the induction hypothesis, Lemma 3.6(ii) holds and \(d_s + \ell_v^{(r)}(d_s, s, w_s) > r - 1\), implying \(d_s + \ell_v^{(r)}(d_s, s, w_s) \geq r\) since all the terms of the inequality are integers.

Otherwise \(r' < r - 1\). In this case, in round \(r - 1\) the triplet \((d_s, s, w_s)\) appeared in \(\text{PL}_v\) and was not yet sent. Since \((d_s, s, w_s)\) is sent in round \(r\), a different triplet \((d_t, t, w_t)\) such that \(t \neq s\) must have been sent in round \(r - 1\), implying:

\[
d_s + \ell_v^{(r-1)}(d_s, s, w_s) > d_t + \ell_v^{(r-1)}(d_t, t, w_t)
\]

By Lemma 3.5(ii), we have that \(\ell_v^{(r)}(d_s, s, w_s) \geq \ell_v^{(r-1)}(d_s, s, w_s)\), and combined with the induction hypothesis for Lemma 3.6(i) in round \(r - 1\) we conclude:

\[
d_s + \ell_v^{(r)}(d_s, s, w_s) \geq d_s + \ell_v^{(r-1)}(d_s, s, w_s) > d_t + \ell_v^{(r-1)}(d_t, t, w_t) \geq r - 1.
\]

This gives that \(d_s + \ell_v^{(r)}(d_s, s, w_s) > r\), since all the terms of the inequality are integers.

Lemma 3.4 implies that eventually, the lists \(\text{PL}_v\) converge to contain the correct values, and Lemma 3.6 restricts the number of rounds in which specific list entries may change. From this, we can conclude that the algorithm solves the weighted \((S, d, k)\)-detection problem, as follows.

**Lemma 3.7.** Given an instance of the weighted \((S, d, k)\)-detection problem, for every \(v \in V\) and for every round \(r\) of an execution of Algorithm 1 with

\[
r \geq r(S, d, k) = \min \{d, D\} + \min \{k, |S|\},
\]

the truncation of \(\text{PL}_v^{(r)}\) to the first \(\min\{k, \lambda_v^d\}\) entries, where \(\lambda_v^d\) is the number of sources \(s \in S\) such that \(d_s \leq d\), solves weighted \((S, d, k)\)-detection.

**Proof.** Assume w.l.o.g. that \(d \leq D\), as \(D\) bounds the distance to any source, and \(k \leq |S|\), as otherwise \(v\) needs to learn about all sources.

In round \(r\), consider the first (up to) \(k\) triplets \((d_s, s, w_s) \in \text{PL}_v^{(r)}\) with \(d_s \leq d\). By Lemma 3.6, no such triplet \((d_s, s, w_s)\) changes when \(r \geq d_s + k \geq d + k\). By Lemma 3.4, in any round \(r' \geq r_0\), for some \(r_0 \in \mathbb{N}\), we have \(\text{PL}_v^{(r')} = \text{PL}_v^{(r_0)} = \text{PL}_v^*\), which means any of the first \(d + k\) entries of \(\text{PL}_v^{(r)}\) are correct.

Lemma 3.7 implies our main result for this section.

**Theorem 3.1.** Given a weighted graph \(G = (V, E, w)\) and a set of nodes \(S \subseteq V\), there exists an algorithm for the CONGEST model, which constructs a WBFS tree rooted at \(s\), for every \(s \in S\), in \(|S| + D\) rounds.
3.2 Discussion and Open Questions

While we present an application of WBFS trees, our algorithm also solves the weighted 
\((S, d, k)\)-detection problem, a result the could be of independent interest.

The question of finding the lightest paths between all pairs of nodes in a graph is a fundamental question in many computational models. In the CONGEST model, Nanongkai [Nan14] gives a \((1 + \epsilon)\)-approximation algorithm for the related problem of finding the lightest paths from a set \(S\) of sources, when considering only paths of at most \(d\) hops, in \(\tilde{O}(|S| + d)\) rounds. He uses this subroutine to devise a \((1 + o(1))\)-approximation weighted APSP algorithm, running in \(\tilde{O}(n)\) rounds. However, no exact APSP algorithm running in \(O(n)\) rounds is known, and even the question of sub-quadratic time APSP algorithm was open until very recently, when Elkin [Elk17] presented an \(O(n^{5/3} \log^{2/3} n)\)-time algorithm for it. This major question is left open here, but we hope our study of lightest shortest paths could facilitate future research on the question of finding lightest paths.
Chapter 4

Distributed Construction of Purely Additive Spanners

4.1 Introduction

This thesis studies the complexity of distributed construction of purely additive spanners in the \textsc{Congest} model. We describe algorithms for building such spanners in several cases. Because of the need to simultaneously make decisions at far apart locations, the algorithms use additional mechanisms compared to their sequential counterparts.

We complement our algorithms with a lower bound on the number of rounds required for computing pairwise spanners. The standard reductions from set-disjointness and equality seem unsuitable for this task because no specific edge needs to be removed from the graph. Instead, to obtain our lower bound, we define a new communication complexity problem that reduces to computing a sparse spanner, and prove a lower bound on its communication complexity. This technique significantly extends the current toolbox used for obtaining lower bounds for the \textsc{Congest} model, and we believe it may find additional applications.

4.1.1 The Challenge

A subgraph $H$ of an undirected unweighted graph $G = (V, E)$ is called a purely additive spanner with stretch $\beta$ if for every every pair $(u, v) \in V \times V$, we have $\text{dist}_H(u, v) \leq \text{dist}_G(u, v) + \beta$, where $\text{dist}_H(u, v)$ is the $u$-$v$ distance in $H$ and $\text{dist}_G(u, v)$ is the $u$-$v$ distance in $G$. The goal in spanner problems is to construct a subgraph $H$ that is as sparse as possible with $\beta$ as small as possible, i.e., we seek a sparse subgraph of $G$ which approximates all distances with a small stretch.

The problem of computing sparse spanners with small stretch $\beta$ is well-studied and we know how to construct sparse purely additive spanners for $\beta = 2, 4, 6$. These have sizes $O(n^{3/2})$ \cite{ACIM99}, $\tilde{O}(n^{7/5})$ \cite{Che13b}, and $O(n^{4/3})$ \cite{BKMP10}, respectively. In a very recent breakthrough, it was shown that there is no purely additive spanner of size at most
In a bid to get sparser subgraphs than all-pairs spanners with the same stretch, the following relaxation of pairwise spanners has attracted recent interest. Here we are given $P \subseteq V \times V$: these are our “relevant pairs” and we seek a sparse subgraph which approximates distances between all pairs in $P$ with a small stretch. That is, for every pair $(u,v) \in P$, the graph $H$ should satisfy $\text{dist}_H(u,v) \leq \text{dist}_G(u,v) + \beta$ and for pairs $(u,v)$ outside $P$, the value $\text{dist}_H(u,v)$ could be arbitrarily large. Such a subgraph $H$ is called a $(+\beta)$-pairwise spanner. We use $\tau(P)$ to denote the number of nodes appearing in $P$, i.e. $\tau(P) = |\{u \mid \exists v : \{u,v\} \in P\}|$.

The problem of constructing sparse pairwise spanners was first studied by Coppersmith and Elkin [CE06] who showed sparse subgraphs where distances for pairs in $P$ were exactly preserved; these subgraphs were called pairwise preservers. A natural case for $P$ is when $P = S \times V$, where $S \subseteq V$ is a set of source nodes — here we seek a sparse subgraph that well-approximates $s$-$v$ distances for all $(s,v) \in S \times V$. Such pairwise spanners are called sourcewise spanners. Another natural setting is when $P = S \times S$ and such pairwise spanners are called subsetwise spanners.

Purely additive spanners are usually built in three steps: first, building clusters which contain all high-degree nodes and adding all the edges of the unclustered nodes; second, building BFS trees which $(+2)$-approximate all the paths with many missing edges; and third, adding more edges to approximate the other paths.

While our constructions follow the general outline of known sequential constructions of pairwise additive spanners [Kav15, KV13], their techniques cannot be directly implemented in a distributed setting. In the sequential setting, the clustering phase is implemented by repeatedly choosing a high-degree node and adding some of its edges to the spanner; these neighbors are marked and ignored in the rest of the phase. In the distributed setting, going over high degree nodes one by one, creating clusters and updating the degrees is too costly. Instead, we choose the cluster centers at random, as done by Cohen [Coh93], Thorup and Zwick [TZ05], Baswana and Sen [BS07], and Chechik [Che13b].

Sources for BFS trees are carefully chosen in the sequential setting by approximately solving a set-cover problem, in order to cover all paths with many missing edges. Once again, this cannot be directly implemented in the distributed setting, as the knowledge of all paths cannot be quickly gathered in one location, so we choose the BFS sources randomly [Che13b]. In both the clustering and BFS phases, the number of edges increases by a multiplicative $\log^c n$ factor, for $c < 1$.

The main challenge left is to choose additional edges to add to the spanner in order to approximate the remaining paths well. To this end, we make heavy use of the parallel-BFS technique of Lenzen and Peleg [LP13b], which allows to construct BFS trees rooted at $s$ different nodes in $O(s + D)$ rounds. We use this technique to count edges in a path, to count missing edges in it, and to choose which edges to add to the spanner. Yet, these techniques were not sufficient in order to match the guarantee on the number of edges.

$O(n^{4/3-\epsilon})$ [AB16a].
<table>
<thead>
<tr>
<th>Spanner Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+2)-sourcewise</td>
</tr>
<tr>
<td>(+2)-pairwise</td>
</tr>
<tr>
<td>(+4)-pairwise</td>
</tr>
<tr>
<td>(+2)-all-pairs</td>
</tr>
<tr>
<td>(+4)-all-pairs</td>
</tr>
<tr>
<td>(+6)-all-pairs</td>
</tr>
<tr>
<td>(+2)-subsetwise</td>
</tr>
<tr>
<td>(+4)-subsetwise</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Edges — Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O \left( n^{5/4}</td>
</tr>
<tr>
<td>( O \left( n</td>
</tr>
<tr>
<td>( O \left( n</td>
</tr>
<tr>
<td>( O \left( n^{3/2} \log^{1/2} n \right) )</td>
</tr>
<tr>
<td>( O \left( n^{7/5} \log^{4/5} n \right) )</td>
</tr>
<tr>
<td>( O \left( n^{4/3} \log^{4/3} n \right) )</td>
</tr>
<tr>
<td>( O \left( n</td>
</tr>
<tr>
<td>( O \left( n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Edges — Sequential</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O \left( n^{5/4}</td>
</tr>
<tr>
<td>( O \left( n</td>
</tr>
<tr>
<td>( O \left( n</td>
</tr>
<tr>
<td>( O \left( n^{3/2} / 2^{3/2} \right) )</td>
</tr>
<tr>
<td>( O \left( n^{7/5} \log^{4/5} n \right) )</td>
</tr>
<tr>
<td>( O \left( n^{4/3} \right) )</td>
</tr>
<tr>
<td>( O \left( n</td>
</tr>
<tr>
<td>( O \left( n</td>
</tr>
</tbody>
</table>

Table 4.1: The number of edges in our new constructions versus prior, sequential work.

The relevant running times appear in Table 4.2. We compare our (+4)-subsetwise with a sequential construction of a (+2)-subsetwise spanner.

Our Contribution

We provide various spanner constructions in the Congest model, as summarized in Tables 4.1 and 4.2.

The distributed spanner construction algorithms we present have three main properties: stretch, number of edges, and running time. All three properties hold w.h.p. (with high probability). That is, the algorithm stops in the desired time, with the desired number of edges and the spanner produced has the desired stretch with probability \( 1 - n^{-c} \), where \( c \) is constant of choice. However, we can trade the properties and guarantee two of the three to always hold: if the spanner is too dense or the stretch is too large, we can repeat the algorithm; if the running time exceeds some threshold, we can stop the execution and output the whole graph to get 0 stretch, or output an empty graph to get the desired number of edges. The edges of the constructed spanner can be counted over a BFS tree in \( G \) within \( O(D) \) rounds. In sourcewise, subsetwise and pairwise spanners, the stretch is measured by running BFS from the relevant nodes (nodes of \( S \), or nodes appearing in \( P \)) for \( O(D) \) rounds in \( G \) and again in \( H \); in all-pairs spanners, the stretch is measured by
measuring the stretch of the underlying sourcewise or subsetwise spanner.

In the case of all-pairs spanners, we also present a tradeoff between the running time and number of edges: if one is willing to compromise on a denser spanner, the running time can be decreased. As an example, consider the construction of a (+2)-all-pairs spanner: for any parameter \( h \geq 1 \), we show how to build such a spanner with \( O \left( nh + \frac{n^2 \log n}{h} \right) \) edges in \( O \left( \frac{n \log n}{h} + D \right) \) rounds (see Theorem 4.25). The sparsest spanner is achieved for \( h = (n \log n)^{1/2} \), it has \( O \left( n^{3/2} \log^{1/2} n \right) \) edges and can be constructed in \( O \left( (n \log n)^{1/2} + D \right) \) rounds; this yields an algorithm that is very similar to the algorithm of Lenzen and Peleg \[LP13b\], though slightly faster and with a slightly stronger sparsity guarantee. A more interesting contribution is the aforementioned tradeoff: in only \( O(D) \) rounds, we can construct a (+2)-all-pairs spanner with \( O \left( \frac{n^2 \log n}{D} + nD \right) \) edges, by choosing \( h = \frac{n \log n}{D} \). The number of edges can be further reduced to \( O \left( \frac{n^2 \log n}{D} + n^{3/2} \log^{1/2} n \right) \), by choosing \( h = \max \left\{ \frac{n \log n}{D}, (n \log n)^{1/2} \right\} \).

A (+6)-All-Pairs Spanner Designing a distributed algorithm for the construction of (+6)-all-pairs spanners was a challenging task. None of the known, sequential algorithms for the task seem to be suitable for a distributed implementation, and so we had to devise a new sequential algorithm, and only then design a distributed implementation of it.

There are three known approaches for the design of sequential algorithms for the construction of (+6)-spanners. The first, presented by Baswana et al. \[BKMP10\], is based on measuring the quality of paths in terms of cost and value, and adding to the spanner only paths which are “affordable”. This approach was later extended by Kavitha \[Kav15\] to other families of additive spanners. The second approach, presented

<table>
<thead>
<tr>
<th>Spanner Type</th>
<th>Number of Rounds</th>
<th>Lower Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+2)-sourcewise ( O(</td>
<td>S</td>
<td>+ D) )</td>
</tr>
<tr>
<td>(+2)-pairwise ( O(\tau(P) + D) )</td>
<td>(Thm 4.10) ( \min \left{ \hat{\Omega} \left( \frac{n^{1/2}}{</td>
<td>P</td>
</tr>
<tr>
<td>(+4)-pairwise ( O(\tau(P) + D) )</td>
<td>(Thm 4.20) ( \min \left{ \hat{\Omega} \left( \frac{n^{1/2}}{</td>
<td>P</td>
</tr>
<tr>
<td>(+2)-all-pairs ( O(n^{1/2} \log^{1/2} n + D) )</td>
<td>(Cor 4.26) ( \min \left{ \hat{\Omega} \left( n^{1/4} \right), \Omega(D) \right} )</td>
<td>[Pet10]</td>
</tr>
<tr>
<td>(+4)-all-pairs ( O(n^{3/5} \log^{1/5} n + D) )</td>
<td>(Cor 4.12) ( \min \left{ \hat{\Omega} \left( n^{3/10} \right), \Omega(D) \right} )</td>
<td>[Pet10]</td>
</tr>
<tr>
<td>(+6)-all-pairs ( O\left( \frac{n^{5/3}}{\log^{1/3} n} + D \right) )</td>
<td>(Cor 4.31) ( \min \left{ \hat{\Omega} \left( n^{1/3} \right), \Omega(D) \right} )</td>
<td>[Pet10]</td>
</tr>
<tr>
<td>(+2)-subsetwise ( O(</td>
<td>S</td>
<td>+ D) )</td>
</tr>
<tr>
<td>(+4)-subsetwise ( O(</td>
<td>S</td>
<td>+ D) )</td>
</tr>
</tbody>
</table>

Table 4.2: Running time: algorithms versus lower bounds, for number of edges as in Table 4.1 We use \( \tau(P) \) to denote the number of nodes appearing in a set \( P \) of pairs.
by Woodruff [Woo10], uses a subroutine that finds almost-shortest paths between pairs of nodes, and obtains a faster algorithm at the expense of a slightly worst sparsity guarantee.

The third approach, presented by Knudsen [Knu14], is based on repeatedly going over pairs of nodes, and adding a shortest path between a pair of nodes if their current distance in the spanner is too long.

Unfortunately, direct implementation in the CONGEST model of the known sequential algorithms is highly inefficient. We are not aware of fast distributed algorithms that allow the computation of the cost and value of paths needed for the algorithm of [BKMP10]. Similarly, for [Woo10], the almost-shortest-paths subroutine seems too costly for the CONGEST model. The algorithm of [Knu14] needs repeated updates of the distances in the spanner between pairs of nodes after every addition of a path to it, which is a sequential process in essence, and thus we do not find it suitable for an efficient distributed implementation.

We remedy this state of affairs by presenting a new algorithm for the construction of (+6)-all-pairs spanners, an algorithm that is more suitable for a distributed implementation. Our construction picks some edges to the spanner and then adds additional paths which minimize the number of additional edges that need to be added to the spanner. To implement our construction in the CONGEST model, we assign weights to the edges and use our WBFS algorithm to find shortest paths with as few edges as possible that are not yet in the spanner. Note that although the graph and the spanner we construct for it are unweighted, the ability of our multiple WBFS algorithm to handle weights is crucial for our solution.

Our algorithm constructs a (+6)-all-pairs spanner with \(O(n^{4/3} \log^{4/3} n)\) edges, in \(\tilde{O}(n^{2/3} + D)\) rounds. This number of edges is larger than the best sequential algorithms [Knu14, BKMP10] by only a \(O(\log^{4/3} n)\) factor, which is essentially tight [AB16a]. Compared with the sequential algorithm of [Woo10], the spanner in our construction is sparser by a factor of \(O(\log^{2/3} n)\).

**Lower Bounds** We complement our algorithms with some lower bounds for the CONGEST model. We show that any algorithm that constructs an additive (+2)-pairwise spanner with \(m'\) edges on \(p \leq m'\) pairs must have at least \(\Omega(p/n)\) rounds, as long as \(m' \in O\left(n^{3/2-\epsilon}\right)\) for a constant \(\epsilon > 0\). For example, a CONGEST construction of a (+2)-pairwise spanner must take \(\Omega\left(n^{1/2-\epsilon}\right)\) rounds. We also prove lower bounds for \((\alpha, \beta)\)-pairwise spanners (i.e., for which \(\text{dist}_H(u, v) \leq \alpha \text{dist}_G(u, v) + \beta\)). We show that any algorithm that constructs an \((\alpha, \beta)\)-pairwise spanner with \(m'\) edges on \(p \leq m'\) pairs must have at least \(\Omega(p/n)\) rounds, as long as \(m' \in O\left(n^{1+\frac{4}{3\alpha+3\beta+10}-\epsilon}\right)\), where the constant in the \(\Omega\) notation depends on \(\alpha, \beta\).

We believe the difficulty in obtaining this lower bound arises from the fact that standard reductions from set-disjointness and equality are unsuitable for this task. At a high level, in most standard reductions the problem boils down to deciding the existence
of an edge (which can represent, e.g., the intersecting element between the inputs); when constructing spanners, no specific edge needs to be added to the spanner or omitted from it, so the solution is allowed a considerable amount of slack that is not affected by any particular edge alone.

Instead, to obtain our lower bound, we define a new communication complexity problem that reduces to computing a sparse spanner, and prove a lower bound on its communication complexity using a counting argument. In this new problem, which we call $\text{SUPERSET}_{m,m',p}$, Alice has a set $x \subseteq \{1, \ldots, m\}$ of size $|x| = p$, and Bob has to output a set $y \subseteq \{1, \ldots, m\}$ of size $|y| = m'$ so that $x \subseteq y$. We show that any protocol that solves $\text{SUPERSET}_{m,m',p}$ must convey $\Omega(p \log n)$ bits of information about the set $x$. This technique significantly extends the current toolbox used for obtaining lower bounds for the CONGEST model. As such, we believe it may find additional applications, especially in obtaining lower bounds for computing in this model.

Roadmap: We conclude this section with a further discussion of related work. In Section 4.2 we present distributed algorithms for computing the various types of spanners discussed above. In Section 4.3 we present our new lower bounds, and we conclude with a short discussion in Section 4.4.

4.1.3 Related Work

Sparse spanners with a small multiplicative stretch are well-understood: Althöfer et al. [ADD+93] in 1993 showed that any weighted graph $G$ on $n$ nodes has a spanner of size $O(n^{1+1/k})$ with multiplicative stretch $2k - 1$, for every integer $k \geq 1$. Since then, several works [BS07, DHZ00, EP04, Elk05, Knu14, Pet09, RTZ05, RZ11, TZ06] have considered the problem of efficiently constructing sparse spanners with small stretch and have used spanners in the applications of computing approximate distances and approximate shortest paths efficiently.

For unweighted graphs, one seeks spanners where the stretch is purely additive and as mentioned earlier, an almost tight bound of $n^{4/3}$ is known for how sparse a purely additive spanner can be. Bollobás et al. [BCE05] were the first to study a variant of pairwise preservers called distance preservers, where the set of relevant pairs is $P = \{(u, v) : \text{dist}_G(u, v) \geq d\}$, for a given parameter $d$. Coppersmith and Elkin [CE06] showed pairwise preservers of size $O(n \sqrt{|P|})$ and $O(n + |P| \sqrt{n})$ for any $P \subseteq V \times V$, which are also applicable for weighted graphs. For unweighted graphs with $|P| = \omega(n^{3/4})$, the bound of $O(n \sqrt{|P|})$ for pairwise preservers has very recently been improved to $O(n^{2/3}|P|^{2/3} + n|P|^{1/3})$ by Bodwin and Vassilevska-Williams [BW16].

The problem of designing sparse pairwise preservers with stretch was first considered by Cygan et al. [CGK13] who showed a tradeoff between the additive stretch and size of the preservers. The current sparsest pairwise spanner with purely additive stretch has size $O(n|P|^{1/4})$ and additive stretch 6 [Kav15]. Woodruff [Woo10] and Abboud and
Bodwin \[AB16b, AB16a\] showed lower bounds for additive spanners and pairwise spanners. Parter \[Par14\] showed sparse multiplicative sourcewise spanners and a lower bound of $\Omega(n|S|^{1/k}/k)$ on the size of a sourcewise spanner with additive stretch $2(k-1)$, for any integer $k \geq 1$.

Distributed construction of sparse spanners with multiplicative stretch was addressed in several studies \[BKMP10, BS07, DG08, DGP07, DGVP08, DMP+05, Pet10\]. Constructions of $(\alpha, \beta)$-spanners were addressed in \[BKMP10, DGVP09, Pet10\]. Towards the goal of obtaining purely additive spanners, for which $\alpha = 1$, Elkin and Peleg \[EP04\] introduced nearly-additive spanners, for which $\alpha = 1 + \epsilon$. Additional distributed constructions of nearly-additive spanners are given in \[Elk05, EZ06, DGVP09, Pet10\]. Finally, somewhat related, are constructions of various spanners in the streaming model, and in dynamic settings, both centralized and distributed \[BKST12, Bas08, BS08, Elko7a, Elko7b\].

The only distributed algorithm for constructing purely additive spanners is by Lenzen and Peleg \[LP13b\], who showed how to construct a $(+2)$-all-pairs spanner with $O(n^{3/2} \log n)$ edges in $O(\sqrt{n} \log n + D)$ rounds. This construction is a simple distributed implementation of a prior, sequential algorithm \[ACIM99, DHZ00\]. The main technical tool used by Lenzen and Peleg is a subroutine they present, which constructs BFS trees from a set $S$ of source nodes in $O(|S| + D)$ rounds; a different subroutine for the multiple-BFS problem was presented by Holzer and Wattenhofer \[HW12\]. The same idea of running multiple BFS searches in parallel was independently presented by Peleg et al. \[PRT12\], but their algorithm constructs BFS trees from all graph nodes, which takes $O(n)$ rounds.

Derbel et al. \[DGVP08\] and Pettie \[Pet10\] present lower bounds for the number of rounds needed by distributed algorithms in order to construct several families of spanners in the LOCAL model. Both bounds are achieved using an indistinguishability argument, a technique that cannot give bounds higher than $\Omega(D)$; indeed, the bounds we present in this work supersede these bounds in the case of low-diameter graphs. We focus on the bounds by Pettie, as the other bounds do not apply to purely additive spanners.

Pettie shows that computing an all-pair additive $\beta$-spanner with size $nh$ in expectation, for a constant $\beta$, requires $\Omega\left(\left(\frac{n}{h}\right)^{1/2}\right)$ rounds of communication. This lower bound is obtained by showing an $n$-node graph with diameter $D = \Theta\left(\left(\frac{n}{h}\right)^{1/2}\right)$ where, roughly speaking, removing wrong edges induces a stretch that is too large, and identifying these wrong edges takes $\Omega(D)$ rounds. This gives a lower bound of $\Omega\left(\min\left\{\left(\frac{n}{h}\right)^{1/2}, D\right\}\right)$ rounds. By examining the construction in detail, it is not hard to show that it works for other types of spanners as well: even for a single pair of nodes, or a set $S$ composed of a single source, at least $\Omega(D)$ rounds are necessary in order to avoid removing wrong edges.

### 4.2 Building Spanners

In this section we present distributed algorithms for building several types of additive spanners. For each spanner, we first describe a template for constructing it independently
of a computational model and analyze its stretch and number of edges. Then, we provide a distributed implementation of the algorithm in the CONGEST model and analyze its running time. For all-pair spanners, we present a tradeoff between the number of edges and running time. This is done by using a parameter \( h \), which is always assumed to satisfy \( 1 \leq h \leq n \). For these spanners, we also present the optimal choice of \( h \), in terms of number of edges, to allow an easy comparison of our results to prior, sequential algorithms.

In a nutshell, our algorithms have three steps: first, each node tosses a coin to decide if it will serve as a cluster center; second, each cluster center tosses another coin to decide if it will serve as a root of a BFS tree; third, add to the current graph edges that are part of certain short paths. The parameters of the coins and the meaning of “short” are carefully chosen, depending on the input to the problem and the desired stretch.

Proving that the algorithms perform well is about analyzing the probability of failure. This analysis uses the graph structure as well as standard concentration bounds. In all of our algorithms, \( c \) is a constant that can be chosen according to the desired exponent of \( 1/n \) in the failure probability.

### 4.2.1 Basic Building Blocks

**Clustering and BFS:** The first building block in all of our algorithms is *clustering*. A cluster \( C_i \) around a cluster center \( c_i \) is a set of neighbors of \( c_i \) (not including \( c_i \) itself). A node belonging to a cluster is said to be *clustered*, while the other nodes of \( G \) are *unclustered*. We use \( \mathcal{C} \) to denote the set of cluster centers and \( \hat{\mathcal{C}} \) to denote the set of clusters.

In the *clustering* phase of our algorithms we divide some of the nodes into clusters. We create a new graph containing all the edges connecting a clustered node to its cluster center, and all the edges incident on unclustered nodes.

Another building block is *BFS trees*. A BFS tree in a graph \( G \), rooted at a node \( r \), consists of shortest paths from \( r \) to all other nodes in \( G \). In the distributed setting, a single BFS tree can be easily constructed by a technique called *flooding* (see, e.g., [Peleg 2000, §3]) and then used in order to compute global parameters such as \( n \) or the number \( |\mathcal{P}| \) of pairs in a pairwise spanner, and to disseminate these values to all graph nodes, in \( O(D) \) rounds. Since the running time of our algorithms is at least \( O(D) \) in all cases, we assume such a pre-processing is always done, and all nodes know \( n \), and \( |\mathcal{P}| \) or \( |S| \) when relevant.

The fast construction of several BFS trees in parallel is a harder task. Lenzen and Peleg [LP13] showed that multiple BFS trees, rooted at a set \( S \) of nodes, can be constructed in \( O(|S| + D) \) rounds. We use this technique in order to add BFS trees to the spanner we construct, and to measure distances in the original graph.

For example, a sourcewise spanner for a set \( S \) of sources can be constructed by running \( |S| \) BFS searches, one from each node of \( S \), and adding all the trees to the spanner \( H \). This constructs a (+0)-sourcewise spanner with \( O(|S| n) \) edges in \( O(|S| + D) \) rounds. This
solves the problem of constructing pairwise spanners with a few sources, which allows us to focus only on the case of many sources in the rest of this thesis.

**Theorem 4.1.** Given a graph $G$ on $n$ nodes and a set of sources $S$, a $(+0)$-sourcewise spanner with $O(|S| n)$ edges can be deterministically constructed in $O(|S| + D)$ rounds in the CONGEST model.

**Chernoff Bounds** We use Chernoff bounds to bound the number of edges in the spanners we construct. If $X$ is a set of nodes defined by adding each node of $V$ to $X$ with an independent probability, such that $E[|X|] = \mu$, then for any $\delta \geq 0$ it holds that $\Pr[|X| \geq (1 + \delta)\mu] < \left(\frac{\exp(\delta)}{(1+\delta)^{(1+\delta)}}\right)^\mu$ (see [MU05 Chapter 4]). For simplicity, we always use the case $\delta = 3$, i.e. $\Pr[|X| \geq 4\mu] < \exp(-\mu)$.

We now turn to describe the subroutines for clustering and constructing BFS trees.

**Algorithm Clustering**

Input: a graph $G = (V, E)$, a parameter $h$

Output: a subgraph $H$, a set $C$ of cluster centers, a set $\hat{C}$ of clusters

Initialization: $n = |V|$, $H = (V, \emptyset)$

Pick each node as a cluster center w.p. $\min\{\frac{c\log n}{h}, 1\}$. Denote the set of selected nodes by $C = \{c_1, c_2, \ldots\}$, and initiate a set of empty clusters $\hat{C} = \{C_i = \emptyset \mid c_i \in C\}$. For each node $v \in V$, choose a neighbor $c_i$ of $v$ which is a cluster center if such a neighbor exists, add the edge $(v, c_i)$ to $H$, and add $v$ to $C_i$; if none of the neighbors of $v$ is a cluster center, add to $H$ all the edges $v$ belongs to.

**Analysis of Clustering**

We now study the properties of the post-clustering graph $H$ created by the algorithm, and then describe its implementation in the CONGEST model and analyze the running time of the algorithm.

**Lemma 4.2.** Given a graph $G = (V, E)$ with $|V| = n$ and a parameter $h \leq n$, Clustering outputs a graph $H = (V, E')$, $E' \subseteq E$, with $|E'| = O(nh)$ edges and a set $\hat{C}$ of at most $\frac{4cn\log n}{h}$ clusters, w.p. at least $1 - O(n^{-c+2})$.

**Proof.** The algorithm starts with $H = (V, \emptyset)$ and adds only edges from $G$, so $E' \subseteq E$. Each node adds to $H$ at most one edge, connecting it to a single cluster center, for a total of $O(n)$ edges. Then, the probability that a node of degree at least $h$ is left unclustered is at most $(1 - \frac{c\log n}{h})^h$ which is $O(n^{-c})$. A union bound implies that all nodes of degree at least $h$ are clustered w.p. $1 - O(n^{-c+2})$, and thus the total number of edges added to $H$ by unclustered nodes in the second part of the clustering phase is $O(nh)$ w.p. $1 - O(n^{-c+2})$.

Each of the $n$ nodes is chosen as a cluster center with probability $\frac{c\log n}{h}$, so the expected number of clusters is $E[|\hat{C}|] = \frac{cn\log n}{h}$. A Chernoff bound implies $\Pr[|\hat{C}| \geq \frac{4cn\log n}{h}] <$
exp\((-\frac{cn \log n}{h})\). Since \(h \leq n\), we have that \(\exp\left(-\frac{cn \log n}{h}\right) \leq \exp(-cn^{1/4}) = O(n^{-c})\), as claimed.

**Lemma 4.3.** Clustering can be implemented in the CONGEST model, so that it always finishes in a constant number of rounds.

**Proof.** Clustering is implemented as follows: first, each node becomes a cluster center w.p. \(\frac{c \log n}{h}\) and sends a message to all its neighbors; then, each node that gets at least one message joins a cluster of one of its neighbors, by sending a message to that neighbor and adding their connecting edge to the graph; finally, nodes that are not neighbors of any cluster center send a message to all their neighbors and add all their incident edges to the graph.

**Algorithm BFS-Trees**

Input: a graph \(G = (V,E)\), a parameter \(h\), the output of Clustering: \(H, C, \hat{C}\)

Output: a new subgraph \(H\) of \(G\)

Initialization: \(n = |V|\)

Pick each cluster center as a root of a BFS tree w.p. \(\min\left\{ \frac{h^2}{c n \log n}, 1 \right\}\), and add to \(H\) a BFS tree rooted at each chosen root.

**Analysis of BFS-Trees**

**Lemma 4.4.** Given a parameter \(h\), \(c \log n \leq h \leq n\) and the output of Clustering executed with this parameter, BFS-Trees adds at most \(4h\) BFS trees to the graph \(H\) and outputs a new graph \(H = (V,E')\), \(E' \subseteq E\), with \(|E'| = O(nh)\) edges w.p. at least \(1 - O(n^{-c+2})\).

**Proof.** A node becomes a root of a BFS tree if it is chosen as a cluster center in Clustering, and then as a root in BFS-Trees, which happens with probability \(\frac{c \log n}{h} \cdot \frac{h^2}{c n \log n} = \frac{h}{n}\). Let \(R\) denote the set of roots, so \(\mathbb{E}[|R|] = \frac{h}{n} \cdot n = h\), and a Chernoff bound implies that \(\Pr[|R| > 4h] \leq \exp(-h)\). As \(h \geq c \log n\) we have \(\exp(-h) = O(n^{-c})\), and BFS-Trees adds at most \(4h\) trees, which are \(O(nh)\) edges. By Lemma 4.2, the output of Clustering is a graph with \(O(nh)\) edges w.p. at least \(1 - O(n^{-c+2})\), and the lemma follows.

**Lemma 4.5.** Let \(\rho\) be a shortest path in \(G\) between two nodes \(u,v \in V\), such that \(\rho\) has more than \(\frac{3c^2 n \log^2 n}{h^2}\) missing edges in \(H\). Then after running BFS-Trees, \(\delta_H(u,v) \leq \delta_G(u,v) + 2\) w.p. at least \(1 - O(n^{-c})\).

**Proof.** Consider a shortest path \(\rho\) between \(u,v \in V\) in \(G\), as in the statement of the lemma (see Figure 4.1). Since \(\rho\) has more than \(\frac{3c^2 n \log^2 n}{h^2}\) missing edges in \(H\) after running Clustering, the nodes of \(\rho\) that belong to clusters must belong to at least \(\frac{c^2 n \log^2 n}{h^2}\) different clusters: otherwise, there are 4 nodes in the same cluster and there is a shorter path between \(u\) and \(v\) in \(G\). The probability that none of the centers of these clusters is chosen as a root in
Figure 4.1: **BFS-Trees** guarantees a (+2) stretch for paths with many missing edges (Lemma 4.5)

**BFS-Trees** is at most \( \left( 1 - \frac{h^2}{cn \log n} \right) c^2 n \log^2 n / h^2 = O(\exp(-c \log n)) = O(n^{-c}) \). Let \( C_i \) be a cluster that \( \rho \) traverses, and let \( w \) be a node in \( \rho \cap C_i \). Adding a BFS tree rooted at \( c_i \) ensures that \( \delta_H(u, c_i) = \delta_G(u, c_i) \leq \delta_G(u, w) + 1 \) and similarly \( \delta_H(c_i, v) = \delta_G(c_i, v) \leq \delta_G(w, v) + 1 \). By the triangle inequality

\[
\delta_H(u, v) \leq \delta_H(u, c_i) + \delta_H(c_i, v) \leq \delta_G(u, w) + \delta_G(w, v) + 2
\]

which equals \( \delta_G(u, v) + 2 \) since \( w \) is on \( \rho \).

**Lemma 4.6.** **BFS-Trees** can be implemented in the **CONGEST** model so that it halts within \( O(h + D) \) rounds w.p. at least \( 1 - o(n^{-c+2}) \).

*Proof.* Each cluster center becomes a root of a BFS tree w.p. \( \frac{eh^2}{n \log n} \), which is done without communication. Then, all BFS roots run BFS searches in parallel. By Lemma 4.4, the number of BFS trees is \( O(h) \) w.p. \( O(n^{-c+2}) \), and this number of BFS searches can be run in parallel in \( O(h + D) \) rounds, using the algorithm of Lenzen and Peleg [LP13b]. While their algorithm outputs the distances along the BFS trees, we wish to mark the BFS tree edges and add them to the graph; this requires a simple change to the algorithm, which does not affect its correctness or asymptotic running time. \( \square \)

### 4.2.2 A (+2)-Sourcewise Spanner

Our first algorithm constructs a (+2)-sourcewise spanner. Given a set \( S \subseteq V \), the algorithm returns a subgraph \( H \) of \( G \) satisfying \( \delta_H(s, v) \leq \delta_G(s, v) + 2 \) for all \( (s, v) \in S \times V \), with guarantees as given in the following theorem.

**Theorem 4.7.** Given a graph \( G \) on \( n \) nodes and a set of sources, a (+2)-sourcewise spanner with \( O(n^{5/4} |S|^{1/4} \log^{3/4} n) \) edges can be constructed in \( O(|S| + D) \) rounds in the **CONGEST** model w.h.p.

This is only a factor \( O(\log^{1/2} n) \) more than the number of edges given by the best sequential algorithm known for this type of spanners [KV13]. Lemmas 4.8 and 4.9 below analyze the size and stretch of Algorithm 2S. The number of rounds of its distributed implementation is analyzed in Lemma 4.10 giving Theorem 4.7.
Algorithm 2S

Input: a graph $G = (V, E)$, a set of source nodes $S \subseteq V$
Output: a subgraph $H$
Initialization: $n = |V|$, $h = (n |S|)^{1/4} \log^{3/4} n$, and $H = (V, \emptyset)$
If $|S| \leq h$, return the union of BFS trees rooted at all nodes of $S$. Otherwise continue as follows.

Clustering and BFS  Run Clustering and then BFS-Trees

Path Buying  For each source-cluster pair $(s, C_i) \in S \times \hat{C}$: build a temporary set of paths, containing a single, arbitrary shortest path from $s$ to each $x \in C_i$; omit from the set all paths with more than $\frac{3c^2 n \log^2 n}{h^3}$ missing edges (i.e. edges in $G$ but not in $H$); if any paths are left, add to $H$ the shortest among them.

Analysis of Algorithm 2S

We now study the properties of the spanner $H$ created by the algorithm; in the following section, we describe the implementation of the different phases in the CONGEST model and analyze the running time of the algorithm. For the case $|S| \leq h$, Theorem 4.7 implies Theorem 4.1 and the three following lemmas, so from here on we assume $|S| > h$.

Lemma 4.8. Given a graph $G = (V, E)$ with $|V| = n$ and a set $S \subseteq V$, Algorithm 2S outputs a graph $H = (V, E')$, $E' \subseteq E$, with $|E'| = O(n^{5/4} |S|^{1/4} \log^{3/4} n)$ edges w.p. at least $1 - O(n^{-c+2})$.

Proof. The algorithm starts with $H = (V, \emptyset)$, and adds to it only edges from $G$. We analyze the number of edges added in each phase.

By Lemma 4.2 Clustering add $O(nh)$ edges to the spanner, and by Lemma 4.4 BFS-Trees adds another $O(nh)$ edges, both w.p. at least $1 - O(n^{-c+2})$. Moreover, by Lemma 4.2 the number of clusters is at most $\frac{4cn \log n}{h}$ with at most the same probability.

For each pair in $S \times \hat{C}$, at most $\frac{3c^2 n \log^2 n}{h^3}$ edges are added in the path buying phase, for a total of $O\left(|S| \cdot \frac{n \log n}{h} \cdot \frac{n \log^2 n}{h^3}\right) = O\left(|S| n \log^3 n\right)$ edges.

Substituting $h = (n |S|)^{1/4} \log^{3/4} n$ gives a total of $O(n^{5/4} |S|^{1/4} \log^{3/4} n)$ edges, as claimed.

Lemma 4.9. Given a graph $G = (V, E)$ with $|V| = n$ and a set $S \subseteq V$, the graph $H$ constructed by Algorithm 2S satisfies $\delta_H(s, v) \leq \delta_G(s, v) + 2$ for each pair of nodes $(s, v) \in S \times V$ w.p. at least $1 - O(n^{-c})$.

Proof. Consider a shortest path $\rho$ between $s \in S$ and $v \in V$ in $G$ (see Figure 4.2).

If $\rho$ has more than $\frac{3c^2 n \log^2 n}{h^3}$ missing edges in $H$ after the clustering phase, then by Lemma 4.5 after running BFS-Trees we have that $\delta_h(s, v) \leq \delta_G(s, v) + 2$ w.p. at least $1 - O(n^{-c})$.  

38

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-16 - 2017
Consider the complementary case, where $\rho$ has at most $\frac{3c^2 n \log^2 n}{h^2}$ missing edges in $H$ after running Clustering, and note that BFS-Trees can only decrease the number of missing edges in $\rho$. If $\rho$ traverses no clusters, then it is contained in $H$, and $\delta_H(s, v) = \delta_G(s, v)$.

Otherwise, let $v'$ be the first clustered node on $\rho$ when traversed from $v$ to $s$ (possibly $v$ itself). The sub-path from $v$ to $v'$ is contained in $H$, so $\delta_H(v, v') = \delta_G(v, v')$. Let $C_i$ be the cluster containing $v'$, then there is a node $x \in C_i$ (possibly $v'$ itself) such that the shortest path between $s$ and $x$ is added to the graph $H$ in the path buying phase, so $\delta_H(s, x) = \delta_G(s, x) \leq \delta_G(s, v')$. The triangle inequality implies
\[
\delta_H(s, v) \leq \delta_H(s, x) + \delta_H(x, v') + \delta_H(v', v) \\
\leq \delta_G(s, v') + 2 + \delta_G(v', v)
\]
where the last inequality uses the previous equality and inequality, together with the fact that $v'$ and $x$ belong to the same cluster. The fact that $v'$ is on $\rho$ implies $\delta_G(s, v') + \delta_G(v', v) = \delta_G(s, v)$ and the claim follows.

\[\square\]

### Implementing Algorithm 2S in the Congest Model

We now discuss the implementation of Algorithm 2S in the Congest model.

**Lemma 4.10.** Algorithm 2S can be implemented in $O(|S| + D)$ rounds in the Congest model, w.p. at least $1 - O(n^{-c+2})$.

**Proof.** Clustering can be implemented in a constant number of rounds, by Lemma 4.3 and BFS-Trees can be implemented in $O(h + D)$ by Lemma 4.6.

The path buying phase starts by measuring all the distances between pairs of nodes in $S \times V$, and counting the number of missing edges in each shortest path measured. To find all distances from a node $s \in S$ to all other nodes, we run a BFS search from $s$; moreover, we augment each BFS procedure with a counter that counts the missing edges in each path from the root to a node on the BFS tree. Running BFS searches from all the nodes of $S$ is done in $O(|S| + D)$ rounds, as before, and adding a counter does not change the time complexity. When a node $v \in V$ receives a message of a BFS initiated by some $s \in S$, it learns its distance from $s$ and the number of missing edges on one shortest path from $s$ to $v$, which lies within the BFS tree; we refer to this path as the shortest path from $s$ to $v$. 

---

Figure 4.2: Algorithm 2S guarantees a (+2) stretch for each pair $(s, v) \in S \times V$ (Lemma 4.9).
After all the BFS searches complete, each clustered node $x$ sends to its cluster center the distance from each $s \in S$ to $x$, and the number of missing edges on the corresponding path. This sub-phase takes $O(|S|)$ rounds to complete.

Each cluster center $c_i$ now knows, for each $s \in S$, the length of the shortest path from $s$ to each $x \in C_i$, and the number of missing edges in each such path; it then locally chooses the shortest among all paths with at most $\frac{2^2n\log^2n}{hn}$ missing edges. Finally, for each chosen $(x, s)$ path, $c_i$ sends a message to $x$ containing the identifier of $s$. All BFS searches are now executed backwards, by sending all the messages in opposite direction and order; when $x$ runs backwards the BFS search initiated by $s$, it marks the message to his parent with a “buy” bit, which is passed up the tree and makes each of its receivers add the appropriate edge to the graph. This sub-phase requires $O(|S| + D)$ rounds as well.

In total, the running time of the algorithm is $O(h + |S| + D)$, w.p. at least $1 - O(n^{-c})$. Since the case $|S| \leq h$ is already handled by Theorem 4.1 we assume here that $|S| > h$, which completes the proof.

4.2.3 A ($+4$)-All-Pairs Spanner

Recall that a subgraph $H$ of $G$ is a ($+4$)-all-pairs spanner if $\delta_H(u, v) \leq \delta_G(u, v) + 4$ for all pairs $(u, v) \in V \times V$. We now use Algorithm 2S in order to design an algorithm that builds a ($+4$)-all-pairs spanner that has the properties guaranteed by the following theorem.

**Theorem 4.11.** Given a graph $G$ on $n$ nodes and a parameter $h$, a ($+4$)-all-pairs spanner with $O\left(nh + \frac{n^3\log^4n}{h^4}\right)$ edges can be constructed in $O\left(\frac{n\log n}{h} + D\right)$ rounds in the Congest model w.h.p.

The main idea is that cluster centers are now sources for a ($+2$)-sourcewise spanner, which, as we show, promises a ($+4$)-stretch to all pairs. Lemmas 4.13 and 4.14 analyze the size and the stretch of Algorithm 4AP below. Lemma 4.15 analyzes the running time of its distributed implementation, completing the proof of Theorem 4.11. Choosing $h = n^{2/5} \log^{4/5} n$ yields a spanner with a minimal number of edges, described in the following corollary.

**Corollary 4.12.** Given a graph $G$ on $n$ nodes, a ($+4$)-all-pairs spanner with $O\left(n^{7/5} \log^{4/5} n\right)$ edges can be constructed in $O(n^{3/5} \log^{1/5} n + D)$ rounds in the Congest model w.h.p.

**Algorithm 4AP**

Input: $G = (V, E)$, $h$
Output: a subgraph $H$
Initialization: $n = |V|$, $H = (V, \emptyset)$

Clustering and BFS  Run Clustering and then BFS-Trees.
Figure 4.3: Algorithm 4AP guarantees a (+4) stretch for each pair $(u, v) \in V \times V$ (Lemma 4.14). The unlabeled nodes and cluster are part of the proof of Lemma 4.9 which guarantees a (+2) stretch between $c_i$ and $v$.

**Path Buying**  Run the path buying phase from Algorithm 2S, with cluster centers as sources, i.e. $S = C$.

**Analysis of Algorithm 4AP**

**Lemma 4.13.** Given a graph $G = (V, E)$ with $|V| = n$, Algorithm 4AP outputs a graph $H = (V, E')$, $E' \subseteq E$, with $|E'| = O\left(nh + \frac{n^3 \log^4 n}{h^4}\right)$ edges w.p. at least $1 - O(n^{-c+2})$.

*Proof.* The proof is analogous to the proof of Lemma 4.8. By Lemmas 4.2 and 4.4, Clustering and BFS-Trees add $O(nh)$ edges to the spanner, w.p. at least $1 - O(n^{-c+2})$.

By Lemma 4.2, the number of clusters is at most $4cn\log n$, so in the execution of Algorithm 4AP, $|C| = \frac{4cn\log n}{h}$ w.p. at least $1 - O(n^{-c+2})$. For each pair in $C \times C$, at most $\frac{3c^2n \log^2 n}{h^2}$ edges are added in the path buying phase, for a total of $O\left(|C|^2 \cdot \frac{n \log^2 n}{h^4}\right) = O\left(\frac{n^4 \log^4 n}{h^4}\right)$ edges. \hfill $\square$

**Lemma 4.14.** Given a graph $G = (V, E)$ with $|V| = n$, Algorithm 4AP outputs a graph $H$ satisfying $\delta_H(u, v) \leq \delta_G(u, v) + 4$ for each pair of nodes $(u, v) \in V \times V$ w.p. at least $1 - O(n^{-c})$.

*Proof.* Let $(u, v) \in V \times V$ be an arbitrary pair of nodes, and set a shortest path $\rho$ in $G$ between them (see Figure 4.3).

If $\rho$ is not incident on any clustered node, then all its nodes are unclustered and all its edges are present in $H$. Otherwise, let $u'$ be the first clustered node on $\rho$, when traversing it from $u$ to $v$, and let $C_i$ be the cluster containing $u'$. The sub-path of $\rho$ from $u$ to $u'$ exists in $H$, as all nodes on this sub-path except for $u'$ are unclustered. It can be easily verified that Lemma 4.9 from the proof of Algorithm 2S holds here as well, with $C$ instead of $S$. Thus, the distance from $c_i$ to $v$ satisfies $\delta_H(c_i, v) \leq \delta_G(c_i, v) + 2$, w.p. at least $1 - O(n^{-c})$. The triangle inequality completes the proof:

$$
\delta_H(u, v) \leq \delta_H(u, u') + \delta_H(u', c_i) + \delta_H(c_i, v) \\
\leq \delta_G(u, u') + 1 + \delta_G(c_i, v) + 2 \\
\leq \delta_G(u, u') + \delta_G(c_i, u') + \delta_G(u', v) + 3 \\
= \delta_G(u, v) + 4
$$
Implementing Algorithm 4AP

Running Algorithm 4AP is done by executing Algorithm 2S with $S$ being the set $C$ of cluster centers; thus, their running times are identical, as stated in the following lemma.

Lemma 4.15. Algorithm 4AP can be implemented in the CONGEST model in $O\left(\frac{n \log n}{h} + D\right)$ rounds w.p. at least $1 - O(n^{-c})$.

Proof. The lemma follows from Lemmas 4.2 and 4.10: in Algorithm 4AP, $S$ is the set of cluster centers, whose size is $|C|$, and by Lemma 4.2, $|C| \leq \frac{4cn \log n}{h}$ with the desired probability. Substituting $|S| = O\left(\frac{n \log n}{h}\right)$ in Lemma 4.10, we get that the algorithm completes in $O\left(\frac{n \log n}{h} + D\right)$ rounds with the claimed probability.

4.2.4 A (+2)-Pairwise Spanner

Recall that a (+2)-pairwise spanner, for a set of pairs $P \subseteq V \times V$, is subgraph $H$ of $G$ satisfying $\delta_H(u, v) \leq \delta_G(u, v) + 2$ for all pairs $(u, v) \in P$. Recall that $\tau(P)$ denotes the number of nodes appearing in $P$, i.e. $\tau(P) = |\{u \mid \exists v : \{u, v\} \in P\}|$.

We present a distributed algorithm, Algorithm 2P, which returns a (+2)-pairwise spanner with the properties described in the following theorem.

Theorem 4.16. Given a graph $G$ on $n$ nodes and a set $P$ of pairs of nodes in $G$, a (+2)-pairwise spanner with $O(n |P|^{1/3} \log^{2/3} n)$ edges can be constructed in $O(\tau(P) + D)$ rounds in the CONGEST model w.h.p.

For the case $\tau(P) \geq 3c^2 |P|^{1/3} \log^{2/3} n$, Lemmas 4.17 and 4.18 below prove the claimed size and stretch of Algorithm 2P, and Lemma 4.19 proves the running time of its distributed implementation, together giving Theorem 4.16. The complementary case is handled by Theorem 4.1.

Algorithm 2P

Input: $G = (V, E)$, $P \subseteq V \times V$
Output: a subgraph $H$
Initialization: $n = |V|$, $h = |P|^{1/3} \log^{2/3} n$, and $H = (V, \emptyset)$
If $\tau(P) < 3c^2 |P|^{1/3} \log^{2/3} n$, return the union of BFS trees rooted at all nodes appearing in $P$. Otherwise continue as follows.

Clustering and BFS Run Clustering and then BFS-Trees.

Path Buying For each pair $(u, v) \in P$, if the shortest path between $u$ and $v$ in $G$ has at most $\frac{3c^2 n \log^2 n}{h^2}$ missing edges in $H$, add it to $H$. 

as desired.
Analysis of Algorithm 2P

**Lemma 4.17.** Given a graph $G = (V, E)$ with $|V| = n$ and a set $\mathcal{P} \subseteq V \times V$, Algorithm 2P outputs a graph $H = (V, E')$, $E' \subseteq E$, with $|E'| = O(n^{1/3} \log^{2/3} n)$ edges, w.p. at least $1 - O(n^{-c+2})$.

*Proof.* The proof is analogous to the proof of Lemma 4.8. By Lemmas 4.2 and 4.4, Clustering and BFS-Trees add $O(nh)$ edges to the spanner, w.p. at least $1 - O(n^{-c+2})$, as long as $c \log n \leq h \leq n$. The first inequality holds due to the condition at the beginning of Algorithm 2P and the fact that $\tau(\mathcal{P}) \leq 2|\mathcal{P}|$, and the second from $|\mathcal{P}| \leq n^2$.

In the path buying phase, at most $\frac{3c^2n \log^2 n}{h^2}$ edges are added for each pair in $\mathcal{P}$, for a total of $O\left(\frac{|\mathcal{P}|n \log^2 n}{h^2}\right)$ edges. Substituting $h = |\mathcal{P}|^{1/3} \log^{2/3} n$, we get a total of $\frac{3c^2n \log^2 n}{h^2}$ edges in $H$, with the desired probability. \hfill $\square$

**Lemma 4.18.** Given a graph $G = (V, E)$ with $|V| = n$, Algorithm 2P outputs a graph $H$ satisfying $\delta_H(u, v) \leq \delta_G(u, v) + 2$ for each pair of nodes $(u, v) \in \mathcal{P}$, w.p. at least $1 - O(n^{-c})$.

*Proof.* Let $(u, v) \in \mathcal{P}$ be an arbitrary pair of nodes, and fix a shortest path $\rho$ in $G$ between them.

If $\rho$ has at most $\frac{3c^2n \log^2 n}{h^2}$ missing edges in $H$ before the path buying phase, it is added to $H$, and $\delta_G(u, v) = \delta_H(u, v)$. Otherwise, note that the number of missing edges can only be decreased by BFS-Trees, so $\rho$ has more than $\frac{3c^2n \log^2 n}{h^2}$ missing edges before running BFS-Trees, and by Lemma 4.5 we have that $\delta_H(u, v) \leq \delta_G(u, v) + 2$, w.p. at least $1 - O(n^{-c})$, as claimed. \hfill $\square$

### Implementing Algorithm 2P

**Lemma 4.19.** Algorithm 2P can be implemented in $O(\tau(\mathcal{P}) + |\mathcal{P}|^{1/3} \log^{2/3} n + D)$ rounds in the Congest model w.p. at least $1 - O(n^{-c+2})$.

*Proof.* By Lemmas 4.3 and 4.6, Clustering and BFS-Trees can be implemented in $O(h + D) = O(\tau(\mathcal{P}) + D)$ rounds with success probability $1 - O(n^{-c})$.

In order to count missing edges in paths, we run a BFS search in $G$ from each node appearing in $\mathcal{P}$. Then, the BFS search is run backwards, and is used to add the “cheap” paths: for a pair $(u, v) \in \mathcal{P}$, if the BFS from $v$ arrives at $u$ traversing at most $\frac{3c^2n \log^2 n}{h^2}$ missing edges, then $u$ sends back a “buy” message up the tree, and the path is added. We may end up adding two shortest paths for a pair $(u, v) \in \mathcal{P}$, but this does not affect the asymptotic number of edges or the time complexity. This phase is implemented in $O(\tau(\mathcal{P}) + D)$ rounds, by running the $\tau(\mathcal{P})$ BFS searches in parallel. \hfill $\square$

### 4.2.5 A (+4)-Pairwise Spanner

We present an algorithm for constructing a (+4)-pairwise spanner, with the parameters described by the following theorem.
Theorem 4.20. Given a graph $G$ on $n$ nodes and a set $P$ of pairs, a $(+4)$-pairwise spanner with $O\left(n |P|^{2/7} \log^{6/7} n \right)$ edges can be constructed in $O(\tau(P) + D)$ rounds in the CONGEST model w.h.p.

Lemmas [4.21] and [4.22] show the claimed size and stretch of Algorithm 4P below, which together with Lemma 4.23, which analyzes the running time of its distributed implementation, prove Theorem 4.20.

Algorithm 4P

Input: a graph $G = (V, E)$, a set of pairs $P \subseteq V \times V$
Output: a subgraph $H$
Initialization: $n = |V|$, $h = |P|^{2/7} \log^{6/7} n$, $\ell = \frac{n \log^3 n}{h^{5/2}}$, $H = (V, \emptyset)$
If $|P| < c \log^4 n$, run Algorithm 2P. Otherwise continue as follows.

Clustering and BFS
Run Clustering and then BFS-Trees

Prefix-Suffix Buying
For each pair $(u, v) \in P$, let $\rho$ be a shortest path from $u$ to $v$. Add to $H$ the first $\ell$ missing edges and the last $\ell$ missing edges in $\rho$.

Choosing Cluster Centers
Construct a set $A$ of cluster centers by adding to it each cluster center independently w.p. $\frac{3c \log n}{\ell}$.

Path Buying
For each pair $(c_1, c_2) \in A \times A$: fix a set of paths containing a single shortest path from $c_1$ to each $x \in C_2$; omit all paths with more than $\frac{3c^2 n \log^2 n}{h^2 n}$ missing edges in $H$; if any paths are left, add to $H$ the shortest among them.

Analysis of Algorithm 4P

If $|P| < c \log^4 n$, Algorithm 2P returns a spanner with the desired number of edges and smaller stretch, as follows from Theorem 4.16. We analyze the complementary case.

Lemma 4.21. Given a graph $G = (V, E)$ with $|V| = n$ and a set $P \subseteq V \times V$, Algorithm 4P outputs a graph $H = (V, E')$, $E' \subseteq E$, with $|E'| = O\left(n |P|^{2/7} \log^{6/7} n \right)$ edges w.p. at least $1 - O(n^{-c+2})$.

Proof. By Lemmas 4.2 and 4.4 Clustering and BFS-Trees add $O(nh)$ edges to the spanner, w.p. at least $1 - O(n^{-c+2})$, as long as $c \log n \leq h \leq n$. The first inequality holds due to the condition on $|P|$ at the beginning of the algorithm, and the second from $|P| \leq n^2$.

In the prefix-suffix buying phase, at most $O(\ell)$ edges are bought for each pair in $P$, for a total of $O\left(|P| \cdot \frac{n \log^3 n}{h^{5/2}}\right) = O\left(n |P|^{2/7} \log^{6/7} n \right)$ edges.
(a) If σ has many missing edges, it is stretched by at most 2, by Lemma 4.5.

(b) If σ has a few missing edges, there is a path through x ∈ C2 that guarantees a small stretch.

Figure 4.4: Algorithm 4P guarantees a (+4) stretch for each pair (u, v) ∈ P (Lemma 4.22). The prefix and suffix on ρ with ℓ missing edges each are marked blue.

Finally, in the path buying phase we add to H at most |A|^2 paths, each with O\left(\frac{n \log^2 n}{h^2}\right) missing edges. Each node is chosen to be a cluster center w.p. \(\frac{c \log n}{h}\), and is added to A w.p. \(\frac{3c \log n}{h}\), so \(\mathbb{E}[|A|] = n \cdot \frac{c \log n}{h} \cdot \frac{3c \log n}{h} = 3c^2 |\mathcal{P}|^{3/7} \log^{2/7} n\). A Chernoff bound implies \(\Pr\left(|A| > 12c^2 |\mathcal{P}|^{3/7} \log^{2/7} n\right) \leq \exp\left(-3c^2 |\mathcal{P}|^{3/7} \log^{2/7} n\right) = O(n^{-c})\); the last equality holds under the assumption |\mathcal{P}| > c \log^2 n, which follows from the condition in the algorithm. Hence, the number of edges added in the path buying step is \(O\left(|\mathcal{P}|^{3/7} \log^{2/7} n \cdot \frac{n \log^2 n}{h}\right) = O\left(n |\mathcal{P}|^{2/7} \log^{6/7} n\right)\) edges. In total, H has \(O\left(n |\mathcal{P}|^{2/7} \log^{6/7} n\right)\) edges w.p. at least \(1 - O(n^{-c+2})\).

**Lemma 4.22.** Given a graph \(G = (V, E)\) with |V| = n, Algorithm 4P outputs a graph H satisfying \(\delta_H(u, v) \leq \delta_G(u, v) + 4\) for each pair of nodes \((u, v) \in \mathcal{P}\) w.p. at least \(1 - O(n^{-c+2})\).

**Proof.** Let \((u, v) \in \mathcal{P}\) be an arbitrary pair of nodes, and fix a shortest path ρ between them. If ρ has at most 2ℓ missing edges in H after Clustering, it is added to H in the prefix-suffix buying phase and \(\delta_G(u, v) = \delta_H(u, v)\).

Otherwise, there are at least \(\ell/3\) clusters incident on the prefix of ρ with ℓ missing edges. Each cluster center is added to A independently w.p. \(\frac{3c \log n}{h}\), so the probability that none of the centers of these clusters is chosen to A is at most \((1 - \frac{3c \log n}{h})^{\ell/3} = O(n^{-c})\). The same argument shows that there exists a cluster center in A whose cluster is incident on the suffix of ρ, and a union bound implies that a similar claim holds for all prefixes and suffixes of shortest paths in G w.p. at least \(1 - O(n^{-c+2})\).
Let $c_1$ be a center of a cluster in $A$ which is incident on the prefix of $\rho$, and $c_2$ a center of a cluster incident on the suffix of $\rho$. Let $u'$ and $v'$ be nodes in $\rho \cap C_1$ and $\rho \cap C_2$ respectively, and let $\sigma$ be a shortest path between $c_1$ and $v'$ in $G$.

If the number of edges of $\sigma$ missing in $H$ after running Clustering is more than $\frac{3\varepsilon \log^2 n}{\varepsilon^2}$, then by Lemma 4.5 we have $\delta_H(c_1, v') \leq \delta_G(c_1, v') + 2$ (see Figure 4.4(a)). Thus, $\delta_H(u, v) \leq \delta_H(u, u') + \delta_H(u', c_1) + \delta_H(c_1, v') + \delta_H(v, v) \leq \delta_G(u, u') + 1 + \delta_G(c_1, v') + 2 + \delta_G(v', v) \leq \delta_G(u, u') + \delta_G(u', v') + 1 + \delta_G(v', v) + 3 = \delta_G(u, v) + 4$.

If $\sigma$ has less than $\frac{3\varepsilon \log^2 n}{\varepsilon^2}$ missing edges then a path between $c_1$ and some $x \in C_2$ is added to $H$ in the path buying phase, satisfying $\delta_H(c_1, x) \leq \delta_G(c_1, v') \leq \delta_G(u', v') + 1$ (see Figure 4.4(b)). Hence, $\delta_H(u, v) \leq \delta_H(u, u') + \delta_H(u', c_1) + \delta_H(c_1, x) + \delta_H(x, v') + \delta_H(v', v) \leq \delta_G(u, u') + 1 + \delta_G(u', v') + 1 + 2 + \delta_G(v', v) = \delta_G(u, v) + 4$, as required.

### Implementing Algorithm 4P

**Lemma 4.23.** Algorithm 4P can be implemented in $O(\tau(\mathcal{P}) + D)$ rounds in the CONGEST model w.p. at least $1 - O(n^{-c})$.

**Proof.** Clustering and BFS-Trees can be implement in $O(h + D)$ rounds with success probability $1 - O(n^{-c+2})$, as stated in Lemmas 4.3 and 4.6.

The prefix-suffix buying phase is implemented as follows: run a BFS from each $u$ appearing in $\mathcal{P}$, counting missing edges on each path. Roll back the BFS, and when passing a node $v$ such that $(u, v) \in \mathcal{P}$, it sends a “buy-suffix” message with a counter initiated to $\ell$; when a node receives such a counter it adds the edge to the parent to $H$ and decreases the counter by 1, unless the edge is already on $H$; if the counter reaches another node $v'$ satisfying $(u, v') \in \mathcal{P}$, it is set to $\ell$ again. When the counter hits 0, no more edges are bought but the counter is passed up the tree, until it arrives at a node whose count of missing edges in the original BFS was $\ell$. This node replaces the counter with a simple “buy” message, adds the edge to its parent to $H$ and sends it to the parent; each receiver of the “buy” message does the same, all the way to the tree root. In total, we construct $\tau(\mathcal{P})$ BFS trees in parallel, which takes $O(\tau(\mathcal{P}) + D)$ rounds, and then send messages in the same paths but opposite direction, which takes the same number of rounds.

The choice of $A$ is simple, requiring no communication.

The path buying phase is implemented similarly to its implementation in Algorithm 2S, in the proof of Lemma 4.10, measuring distances and counting missing edges on a path from each $c_1 \in A$ to each other node is done by running a BFS from each cluster center with the appropriate counters; each clustered node reports its cluster’s center the above parameters in $O(|A|)$ rounds; each $c_2 \in A$ then chooses which paths to buy, and reports the appropriate node in its cluster; running the BFS searches backwards, each node may send “buy” messages up the trees, as decided by $c_2$. This phase takes $O(|A| + D)$ rounds.

The proof of Lemma 4.21 implies $|A| = O(|\mathcal{P}|^{3/7} \log^{2/7} n)$ w.p. at least $1 - O(n^{-c})$. This, together with the assumption $|\mathcal{P}| \geq c \log^4(n)$ and with the choice of $h$, im-
ply \( O(h + |A| + \tau(P) + D) = O(\tau(P) + D) \); hence, the above implementation takes \( O(\tau(P) + D) \) rounds with the desired probability.

### 4.2.6 Subsetwise Spanners

Recall that a \((+\beta)\)-subsetwise spanner for a set \( S \subseteq V \) is a subgraph \( H \) of \( G \) satisfying \( \delta_H(u, v) \leq \delta_G(u, v) + \beta \) for all pairs \( (u, v) \in S \times S \). We show how to build such spanners for \( \beta = 2 \) and \( \beta = 4 \), with \( O(n |S|^{2/3} \log^{2/3} n) \) edges and \( O(n |S|^{1/7} \log^{6/7} n) \) edges respectively, in \( O(|S| + D) \) rounds w.h.p.

The algorithms follow immediately from Algorithm 2P and Algorithm 4P: set \( P = S \times S \) and run Algorithm 2P or Algorithm 4P. The number of edges follows from the fact \( |P| = |S|^2 \), and the running time from \( \tau(P) = |S| \).

Finally, note that in the case \( |S| > n^{3/5} \log^{1/5} n \), Algorithm 2S gives a sparser spanner than Algorithm 2P in the same running time, and with a stretch of +2 for all \( S \times V \) pairs. Similarly, when \( |S| > n^{7/10} \log^{-1/10} n \), Algorithm 4AP gives a sparser spanner than Algorithm 4P in a shorter running time, with a stretch of +4 on all pairs of nodes in the graph.

**Corollary 4.24.** Given a graph \( G \) on \( n \) nodes and a set \( S \) of nodes, a \((+2)\)-subsetwise spanner with \( O \left( \min \left\{ n |S|^{2/3} \log^{2/3} n, n^{5/4} |S|^{1/4} \log^{3/4} n \right\} \right) \) edges and a \((+4)\)-subsetwise spanner with \( O \left( \min \left\{ n |S|^{1/7} \log^{6/7} n, n^{7/5} \log^{4/5} n \right\} \right) \) edges, can both be constructed in at most \( O(|S| + D) \) rounds in the CONGEST model w.h.p.

### 4.2.7 A \((+2)\)-All-Pairs Spanner

In this section we present an algorithm that constructs a \((+2)\)-all-pairs spanner. While the algorithm resembles a prior result \([\text{LPT13b}]\), we presents a new tradeoff between the running time and sparsity of the constructed spanner.

**Theorem 4.25.** Given a graph \( G \) on \( n \) nodes and a parameter \( h \), a \((+2)\)-all-pairs spanner with \( O \left( nh + \frac{n^2 \log n}{h} \right) \) edges can be constructed in \( O \left( \frac{n \log n}{h} + D \right) \) rounds in the CONGEST model w.h.p.

Lemmas 4.27 and 4.28 provide the required size and stretch of Algorithm 2AP below, and Lemma 4.29 gives the running time of its distributed implementation, proving Theorem 4.25. Substituting \( h = (n \log n)^{1/2} \) in the theorem gives a minimal number of edges, described in the following corollary.

**Corollary 4.26.** Given a graph \( G \) on \( n \) nodes, a \((+2)\)-all-pairs spanner with \( O \left( n^{3/2} \log^{1/2} n \right) \) edges can be constructed in \( O \left( (n \log n)^{1/2} + D \right) \) rounds in the CONGEST model w.h.p.
Figure 4.5: Algorithm 2AP guarantees a (+2) stretch for each pair \((u, v) \in V \times V\) (Lemma 4.28).

**Algorithm 2AP**

Input: \( G = (V, E), h \)
Output: a subgraph \( H \)
Initialization: \( n = |V|, H = (V, \emptyset), A = \emptyset \)

**Low-degree** For each node \( v \in V \), if \( \text{deg}(v) \leq h \) then add to \( H \) all the edges incident on \( v \).

**BFS** Add each node \( v \in V \) to \( A \) w.p. \( \frac{c \log n}{h} \). For each \( v \in A \), add to \( H \) a BFS tree rooted at \( v \).

**Analysis of Algorithm 2AP**

**Lemma 4.27.** Given a graph \( G = (V, E) \) with \( |V| = n \) and a parameter \( h \), Algorithm 2AP outputs a graph \( H = (V, E'), E' \subseteq E \), with \( |E'| = O \left( nh + \frac{n^2 \log n}{h} \right) \) edges w.p. at least \( 1 - O(n^{-c}) \).

**Proof.** In the Low-degree phase, at most \( nh \) edges are added to \( H \).

In the BFS phase, the expected number of nodes in \( A \) is \( \mathbb{E}[|A|] = \frac{cn \log n}{h} \). A Chernoff bound implies \( \Pr \left[ |A| \geq \frac{4cn \log n}{h} \right] < \exp \left( -\frac{c \log n}{h} \right) \). Since \( h \leq n \), we have that \( \exp \left( -\frac{c \log n}{h} \right) \leq \exp \left( -cn^{1/4} \right) = O(n^{-c}) \). Thus, at most \( O \left( \frac{n \log n}{h} \right) \) BFS trees with at most \( O \left( \frac{n^2 \log n}{h} \right) \) edges are added to \( H \), w.p. at least \( 1 - O(n^{-c}) \). \( \square \)

**Lemma 4.28.** Given a graph \( G = (V, E) \) with \( |V| = n \), Algorithm 2AP outputs a graph \( H \) satisfying \( \delta_H(u, v) \leq \delta_G(u, v) + 2 \) for each pair of nodes \((u, v) \in V \times V\) w.p. at least \( 1 - O(n^{-c}) \).

**Proof.** Let \((u, v) \in V \times V\) be an arbitrary pair of nodes, and fix a shortest path \( \rho \) in \( G \) between them (see Figure 4.5).

Let \( u' \) be the first node satisfying \( \text{deg}(u') > h \) when traversing \( \rho \) from \( u \) to \( v \), possibly \( u \) itself. The prefix of \( \rho \) from \( u \) to \( u' \) consists of low-degree nodes, so the Low-degree phase adds all of it to \( H \). If no such node exists, then all the edges of \( \rho \) are added to \( H \) and we are done.
The probability that none of the neighbors of \( u' \) enters \( A \) is at most \( \left( 1 - \frac{c \log n}{h} \right)^h = O(n^{-\epsilon}) \). Let \( x \) be such a neighbor, if exists, so \( H \) contains a BFS tree rooted at \( x \) and thus a shortest path from \( x \) to \( v \). The triangle inequality completes the proof:

\[
\delta_H(u,v) \leq \delta_H(u,u') + 1 + \delta_H(x,v) \\
\leq \delta_H(u,u') + 1 + \delta_G(x,v) \\
\leq \delta_H(u,u') + 2 + \delta_G(u',v) = \delta_G(u,v) + 2
\]

as desired.

**Implementing Algorithm 2AP**

**Lemma 4.29.** Algorithm 2AP can be implemented in the CONGEST model in \( O \left( \frac{n \log n}{h} + D \right) \) rounds w.p. at least \( 1 - O(n^{-\epsilon}) \).

**Proof.** The Low-degree phase can be implemented in a single round, by having each low-degree node send a message to all its neighbors.

The choice of \( A \) is done without communication, and the construction of BFS trees is done in parallel, in \( O(|A| + D) \) rounds. By the proof of Lemma 4.27, we have that \( |A| = O \left( \frac{n \log n}{h} \right) \) w.p. at least \( 1 - O(n^{-\epsilon}) \), which completes the proof.

**4.2.8 A (+6)-All-Pairs Spanner**

In this section we discuss the distributed construction of (+6)-all-pairs spanners. Like previous algorithms, our algorithms starts with clustering, and then shortest paths between clusters are added to the spanner. In order to find these shortest paths in the CONGEST model, we use the WBFS construction algorithm. At the heart of our algorithm stands the path-hitting framework of Woodruff [Woo10]: a shortest path in the graph which has many edges between clustered nodes, must go through many clusters. This fact is used in order to show that a path with many missing edges is more likely to have an adjacent source of a WBFS tree, and thus it is well approximated by a path within the spanner.

Woodruff’s algorithm starts with a similar clustering step. However, in order to add path between clusters, it uses an involved subroutine that finds light almost-shortest paths between pairs of nodes. This subroutine seems too global to be implemented efficiently in the CONGEST model, so in our construction it is replaced by only considering shortest paths, which we do using the WBFS trees defined earlier.

Our algorithm constructs a (+6)-all-pairs spanner, and presents a tradeoff between the running time and number of edges, as stated in the following theorem.

**Theorem 4.30.** Given a graph \( G \) on \( n \) nodes and a parameter \( h \), a (+6)-all-pairs spanner with \( O \left( nh + \frac{n^2 \log^4 n}{h^2} \right) \) edges can be constructed in \( O \left( \frac{n \log n}{h} + D \right) \) rounds in the CONGEST model w.h.p.
Lemmas 4.32 and 4.33 analyze the size and stretch of Algorithm 6AP given below. The number of rounds of its distributed implementation is analyzed in Lemma 4.34, giving Theorem 4.30. Substituting \( h = n^{1/3}n \log^{4/3} n \) in the theorem gives a minimal number of edges, described in the following corollary.

**Corollary 4.31.** Given a graph \( G \) on \( n \) nodes, a \((+6)\)-all-pairs spanner with \( O \left( \left( n \log n \right)^{4/3} \right) \) edges can be constructed in \( O \left( \frac{n^{2/3}}{\log^{1/3} n} + D \right) \) rounds in the CONGEST model w.h.p.

**Algorithm 6AP**

Input: \( G = (V, E), h \)

Output: a subgraph \( H \)

Initialization: \( n = |V|, H = (V, \emptyset), k = 1 \)

**Clustering** Run Clustering and mark the returned graph as \( H_0 \)

**Path Buying.**

While \( k \leq \frac{12cn \log n}{h} \) do:

1. \( S_k \leftarrow \emptyset \)

2. Add each cluster center \( c_i \in C \) to \( S_k \) w.p. \( \frac{3c \log n}{k} \), independently of other cluster centers

3. For each pair \((c_i, c_j) \in C \times S_k\):
   
   (a) \( A \leftarrow \emptyset \) /* \( A \) is a set of paths */
   
   (b) For each \( v \in C_j \):
     
     i. Let \( P_v \) be a shortest path from \( c_i \) to \( v \) such that \( |P_v \setminus H_0| \) is minimal
     
     ii. If \( |P_v \setminus H_0| < 2k \), add \( P_v \) to \( A \)

   (c) If \( A \neq \emptyset \), add to \( H \) the shortest among all paths in \( A \)

4. \( k \leftarrow 2k \)

**Analysis of Algorithm 6AP**

**Lemma 4.32.** Algorithm 6AP outputs a graph \( H = (V, E') \), \( E' \subseteq E \), with \( |E'| = O \left( nh + \frac{n^2 \log^4 n}{h^2} \right) \) edges, with probability at least \( 1 - O(n^{-5c/4+3}) \).

**Proof.** The algorithm starts with \( H = (V, \emptyset) \) and only adds edges from \( G \), so \( H \) is indeed a subgraph of \( G \) over the same nodes.

By Lemma 4.2, the clustering phase adds \( O(nh) \) edges to \( H_0 \) w.p. at least \( 1 - O(n^{-c+2}) \). Also, \( |\hat{C}| \leq \frac{4cn \log n}{h} \) with the same probability.
Lemma 4.33. The graph $H$ constructed by Algorithm 6AP satisfies $\delta_H(x, y) \leq \delta_G(x, y) + 6$ for each pair $(x, y) \in V \times V$, with probability at least $1 - O(n^{-c/4+2})$.

Proof. Consider a shortest path $\rho$ in $G$ between two nodes $x, y \in V$ (See Figure 4.6). Let $x'$ and $y'$ be the first and last clustered nodes on $\rho$ respectively. If all nodes of $\rho$ are unclustered, then $\rho$ is fully contained in $H_0$ and we are done.

Let $c_1$ and $c_3$ be the centers of the clusters containing $x'$ and $y'$, respectively. Let $\sigma$ be a shortest path in $G$ between $c_1$ and $c_3$, denote by $k'$ the number of edges of $\sigma \setminus H_0$, and by $k$ the largest power of 2 such that $k \leq k'$. As $\sigma$ is a shortest path, the number of clusters adjacent to $\sigma$ is at least $k'/3$, and $k'/3 \geq k/3$. Lemma 4.2 implies that, with the desired probability, the total number of clusters $|C|$ is at most $\frac{4cn \log n}{k}$ and thus $k \leq k' \leq \frac{12cn \log n}{k}$. The value of $k$ is processed in the Path Buying phase. In addition, when $k$ is processed and $S_k$ is constructed, the probability that none of the centers of the clusters adjacent to $\sigma$ is chosen to $S_k$ is upper bounded by $(1 - \frac{3c \log n}{k})^{k/3} = O(n^{-c})$. A union bound implies that the same argument holds for all pairs in $V \times V$, w.p. at least $1 - O(n^{-c/2})$. 

To analyze the path-buying phase, we bound the size of $S_k$ in a way similar to the one used in the proof of Lemma 4.2 for bounding the number of clusters. By the choice of cluster centers into $S_k$, we have $\mathbb{E}[|S_k|] = \frac{3c^2n \log^2 n}{hk}$, and

$$\Pr \left[ |S_k| > \frac{12c^2n \log^2 n}{hk} \right] \leq \exp \left( -\frac{3c^2n \log^2 n}{hk} \right) = O(n^{-c/4}),$$

where the last equality follows since $k \leq \frac{12cn \log n}{h}$. A union bound implies that $|C| = O\left(\frac{n \log n}{h}\right)$ and $|S_k| = O\left(\frac{n \log^2 n}{hk}\right)$ for all $k$, w.p. at least $1 - O(n^{-5c/4+3})$.

Finally, for each $k$, for each $(c_i, c_j) \in C \times S_k$ we add at most one path with less than $2k$ missing edges to $H$. Thus, for each value of $k$ we add less than $|C| \cdot |S_k| \cdot 2k = O\left(\frac{n^2 \log^3 n}{h^2}\right)$ edges to $H$, with the above probability. Summing over all $O(\log n)$ values of $k$, and adding the number of edges contributed by the clustering phase, we conclude that $H$ has at most $O\left(nh + \frac{n^2 \log^4 n}{h^2}\right)$ edges, w.p. at least $1 - O(n^{-5c/4+3})$. 

Figure 4.6: Algorithm 6AP guarantees a (+6) stretch for each pair $(u, v) \in V \times V$ (Lemma 4.33)
Let $w$ be a node on $\sigma$ in a cluster $C_2$ such that $c_2 \in S_k$, if such a cluster exists. Denote by $\sigma[c_1, w]$ the sub-path of $\sigma$ from $c_1$ to $w$. As there are $k' < 2k$ edges in $\sigma \setminus H_0$, there are at most $2k$ edges in $\sigma[c_1, w] \setminus H_0$ as well. Thus, in step 3(b) of the path-buying phase for $k$, either the path $\sigma[c_1, w]$ or some other path between $c_1$ and $w$ of length at most $\delta_G(c_1, w)$ is added to $A$. In step 3(c), a path from $c_1$ to some node $w_1 \in C_2$ is added to $H$, and this is a shortest path in $A$, so $\delta_H(c_1, w_1) \leq \delta_G(c_1, w)$. Similarly, a shortest path from $c_3$ to some $w_3 \in C_2$ is added to $H$, and $\delta_H(c_3, w_3) \leq \delta_G(c_3, w)$.

The path $\sigma$ is a shortest path from $c_1$ to $c_3$ in $G$, so $|\sigma| \leq \delta_G(x', y') + 2$. As $\delta_G(c_1, w) + \delta_G(c_3, w) = |\sigma|$, we conclude $\delta_H(c_1, w_1) + \delta_H(w_3, c_3) + 1 + \delta_H(y', y) \leq \delta_G(x', y') + 2 + \delta_G(y', y) = \delta_G(x, y) + 6$,

as desired. \hfill \Box

### Implementing Algorithm 6AP

We now discuss the implementation of Algorithm 6AP in the CONGEST model.

**Lemma 4.34.** Algorithm 6AP can be implemented in $O\left(\frac{n \log n}{k} + D\right)$ rounds in the CONGEST model with probability at least $1 - O\left(n^{-c+2}\right)$.

**Proof.** Clustering can be implemented in $O(h + D)$ rounds with success probability $1 - O\left(n^{-c+2}\right)$, as stated in Lemma 4.3.

The nodes mark the edges of $H_0$ with weight 0 and the other edges with weight 1. Then, they construct a WBFS tree rooted at each cluster center by executing Algorithm 1 for $|C| + D'$ many rounds. By Lemma 4.2, $|C| \leq \frac{4cn \log n}{h}$ with the desired probability, and thus the construction of the WBFS trees takes $O\left(\frac{n \log n}{k} + D\right)$ rounds with the same probability.

Each node $v$ now knows about a “good” path to each cluster center $c_i$, i.e., a shortest path from $c_i$ to $v$, with a minimal number of edges not in $H$ after the clustering phase. A node $v$ in a cluster $C_j$ notifies its neighbor $c_j$ about all the distances to other cluster centers in $C$ and the number of missing edges in each such path. That is, each $v \in C_j$ sends $|C|$ messages to $c_j$, which takes $O\left(\frac{n \log n}{k} + D\right)$ rounds.

Each cluster center $c_j$ decides locally to join each set $S_k$ w.p. $\frac{3c \log n}{k}$. For each other center $c_i \in C$, $c_j$ locally constructs the list $A$: for each $v \in C_j$, $A$ contains the shortest path from $c_i$ to $v \in C_j$ found by the WBFS algorithm from $c_i$, and the number of missing edges in it. Then, $c_j$ chooses from $A$ a path from $c_i$ to some $v \in C_j$ with a minimal number of missing edges, and if it has at most $2k$ missing edges, $c_j$ sends a “buy $c_i$” message to $v$. 

52
Finally, all nodes simultaneously execute a “buy” phase, where “buy $c_i$” messages are sent up the WBFS tree. To avoid congestion, we assume that during the execution of Algorithm 1, each node keeps a record of the messages it got in each round and the WBFS source each message referred to. Each node $v$ then sends messages in reversed order: if $v$ has a message “buy $c_i$”, and it got a message from $u$ regarding $c_i$ in the $r$-before-last round of Algorithm 1, then it sends the message “buy $c_i$” to $u$ in round $r$ of the “buy” phase. Then, $u$ adds “buy $c_i$” to its list of messages, and adds the edge $(u,v)$ to the spanner. This part takes $O\left(\frac{n \log n}{n} + D\right)$ rounds w.p. at least $1 - O(n^{-c+2})$, as in the execution of Algorithm 1 from all cluster centers.

4.3 Lower Bounds

In this section we prove lower bounds on the number of rounds that are needed for constructing spanners in the CONGEST model. All previous lower bounds for the distributed construction of spanners [Pet10] use an indistinguishability argument: while many edges should be omitted from the graph in order to create a sparse spanner, there are a few edges that must not be omitted. However, in order to distinguish these few edges from the rest, some nodes must learn a considerable part of the graph. In a nutshell, the heart of the proof is that information must travel a constant portion of the diameter $D$, and thus the lower bound is $\Omega(D)$.

The lower bounds from [Pet10] apply also to the LOCAL model, where the message sizes are unbounded. Here, we present the first lower bound that is specific for the CONGEST model. As in previous lower bounds for the CONGEST model, our proof uses a reduction from a communication complexity problem. However, previous lower bounds used reductions either from the equality problem [PR99] or from set-disjointness, e.g., [DSHK+12, DKO14, FHW12, HP15, CGK14, GK13]. These seem unsuitable for our purposes, thus we diverge from this approach and define a new communication complexity problem we call superset. We bound the communication complexity of this problem from below.

We first prove a lower bound for the construction of a $(+2)$-pairwise spanner. Then, we generalize the bound for the construction of an $(\alpha, \beta)$-pairwise spanner, for any $\alpha \geq 1, \beta \geq 0$.

4.3.1 A Communication Complexity Problem

Let $m, m', p$ be three positive integers so that $p \leq m' \leq m^{1-\epsilon}$ for some $\epsilon > 0$. The superset communication problem, denoted $\text{SUPERSET}_{m,m',p}$, is defined as follows: Alice has a set $x \subseteq \{1, \ldots, m\}$ of size $|x| = p$, and Bob has to output a set $y \subseteq \{1, \ldots, m\}$ of size $|y| = m'$ so that $x \subseteq y$. Note that the goal of this communication problem is to compute a relation, not a function. In this section we prove that the randomized communication complexity
of the superset problem is high (for formal definitions in communication complexity see the textbook [KN97]).

**Theorem 4.35.** If \( \pi \) is a \((1/3)\)-error randomized protocol computing \( \text{SUPERSET}_{m,m',p} \) then the number of communicated bits in \( \pi \) is more than \( \epsilon p \log(m) - 1 \).

**Proof.** Let \( \pi \) be a randomized protocol computing \( \text{SUPERSET}_{m,m',p} \) with \( c \) bits of communication and error probability 1/3. We can assume without loss of generality that only Alice speaks in \( \pi \), since Bob does not receive an input. The protocol \( \pi \) thus has the following form: There is public randomness \( r \) that is visible to both Alice and Bob. Alice gets input \( x \) and sends to Bob a message \( M \in \{0,1\}^c \) that is a function of both \( x \) and \( r \). Bob decodes \((M,r)\) to name a set \( y \). The protocol has the property that for all \( x \), the probability over \( r \) that \( x \subseteq y \) is at least \( 2/3 \).

The lower bound we prove is in the distributional setting. The input \( x \) given to Alice is distributed uniformly at random from all possible subsets of \( \{1,\ldots,m\} \) of size \( p \). Thus, the probability over \( x \) and \( r \) that \( x \subseteq y \) is at least \( 2/3 \). We can thus fix some \( r = r_0 \), and get that the probability over \( x \) that \( x \subseteq y \) is at least \( 2/3 \).

Now, the output of Bob is completely determined by \( M \), since \( r = r_0 \) is fixed. Moreover, each of his outputs is correct for exactly \( \binom{m'}{p} \) input sets. Thus, Bob is correct on at most \( 2^c \binom{m'}{p} \) of the inputs. The probability that the output is correct is therefore at most

\[
2^c \binom{m'}{p} / \binom{m}{p} = 2^c \frac{(m')!}{(m'-p)!} \cdot \frac{(m-p)!}{m!} = 2^c \prod_{i=0}^{p-1} \frac{m' - i}{m - i} \leq 2^c \left( \frac{m'}{m} \right)^p \leq 2^c m^{-\epsilon p}.
\]

Since \( \pi \) is correct w.p. at least \( 2/3 \), we know \( 2/3 \leq 2^c m^{-\epsilon p} \) and so \( \epsilon p \log m + \log(2/3) \leq c \). \( \square \)

### 4.3.2 A Lower Bound for Constructing a \((+2)\)-Pairwise Spanner

In this section we prove a lower bound on the number of rounds needed for constructing a sparse \((+2)\)-pairwise spanner. This bound holds even for graphs of constant diameter, as it relies only on the amount of communication needed and not on the distances in the graph.

**Theorem 4.36.** For any constant \( \epsilon > 0 \) the following holds. Any distributed protocol for the Congest model with success probability at least \( 2/3 \) which, given a graph with \( n \) nodes and a set of \( p \in O(n^{3/2-\epsilon}) \) pairs of nodes, outputs a \((+2)\)-pairwise spanner with \( O(n^{3/2-\epsilon}) \) edges, must take \( \Omega(p/n) \) rounds to complete.
For any constant \( \epsilon > 0 \), the theorem implies a lower bound of \( \Omega \left( n^{1/2-\epsilon} \right) \) on the number of rounds needed for an algorithm in the CONGEST model to output a \((+2)\)-pairwise spanner with \( O(n^{3/2-\epsilon}) \) edges, when \( |P| = \Theta(n^{3/2-\epsilon}) \). For comparison, for a set \( \mathcal{P} \) of that size, Algorithm 2P can construct a \((+2)\)-pairwise spanner in time that vary between \( O \left( n^{3/4-\epsilon} \right) \) and \( O(n) \), depending on the structure of \( \mathcal{P} \).

The girth of a graph is the length of the shortest simple cycle in it. There is a known upper bound on the asymptotic number of edges in a graph of a given girth \( g \), but constructions of graph that match this bound are scarce. One of the cases where a graph \( G' \) with an optimal number of edges can be constructed is that of girth \( g = 6 \), as follows Theorem 4.40 bellow.

**Lemma 4.37.** For every \( n \geq 2 \) there is a graph \( G' \) on \( n/2 \) nodes with girth 6, \( \Theta(n^{3/2}) \) edges, and a constant diameter.

The graph \( G \) for which the lower bound is proved is defined as follows. We denote the nodes of \( G' \) by \( V_0 = \{v'_1, \ldots, v'_{n/2}\} \). The graph \( G \) consists of \( G' \), an additional \( n/2 \) nodes denoted \( V_A = \{v_1, \ldots, v_{n/2}\} \), and an additional \( n/2 \) edges of the form \( (v_i, v'_j) \).

In the pairwise spanner we construct, we wish to approximately preserve distances between pairs of nodes in \( V_A \), i.e. \( \mathcal{P} \subseteq V_A \times V_A \). The main observation is that, since the girth of \( G' \) is 6, if \( e' = \{v'_i, v'_j\} \) is an edge of \( G' \) then the following holds. If \( (v_i, v_j) \in \mathcal{P} \) then any \((+2)\)-pairwise spanner must contain the edge \( e' \), as otherwise the distance is stretched from 3 to 7, which exceeds the required +2 stretch. On the other hand, if \( (v_i, v_j) \notin \mathcal{P} \) then the edge \( e' \) can be safely omitted from the spanner.

**Proof of Theorem 4.36** Fix a distributed protocol \( \sigma \) for constructing a \((+2)\)-pairwise spanner with \( m' \in O \left( n^{3/2-\epsilon} \right) \) edges. Let \( G \) be the graph described above, and denote the edges of \( G' \) by \( e_1, \ldots, e_m \).

We describe a reduction from \textsc{Superset} \(_{m,m',p}\) to \( \sigma \). Assume Alice has a set \( x \subseteq \{1, \ldots, m\} \) of size \( p \), and Bob has to output a set \( y \subseteq \{1, \ldots, m\} \) of size \( m' \) satisfying \( x \subseteq y \). Alice and Bob simulate \( \sigma \) on the graph \( G \) with the set of pairs

\[
\mathcal{P} = \{(v_i, v_j) : \exists k \in x \text{ } e_k = \{v'_i, v'_j\}\}.
\]

That is, a pair \((v_i, v_j)\) is in \( \mathcal{P} \) if the corresponding pair \((v'_i, v'_j)\) is an edge \( e_k \) whose index \( k \) is in \( x \). Alice simulates the nodes in \( V_A \), and Bob simulates the nodes \( V_B \) and the edges among them. To simulate communication on edges of the form \((v_i, v'_j)\), Alice and Bob communicate. Note that \( \mathcal{P} \) contains only pairs of nodes that are simulated by Alice.

The spanner constructed is a subgraph \( H \) of \( G \) with at most \( m' \) edges, satisfying \( \delta_H(v_i, v_j) \leq \delta_G(v_i, v_j) + 2 \) for all \((v_i, v_j) \in \mathcal{P} \). For each such pair, by definition of \( \mathcal{P} \), we have \( \delta_G(v_i, v_j) = 3 \), which implies \( \delta_H(v_i, v_j) \leq 5 \) and \( \delta_H(v'_i, v'_j) \leq 3 \). The fact that \( G' \) has girth 6 implies that the edge \( \{v'_i, v'_j\} \) must be in \( H \). Let

\[
y = \{k : e_k \in E_H\}.
\]
The spanner size implies $|y| \leq m'$, while the above discussion implies $x \subseteq y$. Thus, Bob can output a subset of $y$ of size $m'$, solving the communication complexity problem.

By the communication complexity lower bound (Theorem 4.35), Alice and Bob must communicate $\Omega(p \log n)$ bits during the simulation. The number of edges they simulate together is $n/2$, and $O(\log n)$ bits are sent over each edge at each round. Thus, the protocol must take $\Omega(|P|/n)$ rounds to complete.

Theorem 4.36 applies for any algorithm that constructs a $(+2)$-pairwise spanner with $O\left(\frac{n^{3/2}}{2^{1-\epsilon}}\right)$ edges. For algorithms that construct a spanner with $o\left(\frac{n^{3/2}}{2^{1-\epsilon}}\right)$ edges but not $O\left(\frac{n^{3/2}}{2^{1-\epsilon}}\right)$ edges, we can get a slightly weaker result.

Fix a distributed protocol $\sigma$ for constructing a $(+2)$-pairwise spanner with $o\left(\frac{n^{3/2}}{2^{1-\epsilon}}\right)$ edges. Let $m$ be the number of edges in the graph $G$ described above, so for large enough value of $n$ the algorithm constructs a spanner with at most $m' \leq m/2$ edges. Apply Theorem 4.35 with $\epsilon = \frac{1}{\log m}$ to conclude that for $m' \leq m/2$, the number of communicated bits in $\pi$ is more than $p - 1$. The same proof as in Theorem 4.36 implies that Alice and Bob must communicate $\Omega(p)$ bits in order to construct a spanner with at most $m'/2$ edges, which gives the following lower bound.

**Theorem 4.38.** There is a constant $c > 0$ so that the following holds. Any distributed protocol for the CONGEST model with success probability at least $2/3$ which, given a graph with $n$ nodes and a set of $p \leq cn^{3/2}$ pairs of nodes, outputs a $(+2)$-pairwise spanner with at most $cn^{3/2}$ edges, must take $\Omega\left(\frac{p}{n \log n}\right)$ rounds to complete. The lower bound holds even for graphs with constant diameter.

### 4.3.3 Generalization: A Lower Bound for Constructing an $(\alpha, \beta)$-Pairwise Spanner

Recall that an $(\alpha, \beta)$-pairwise spanner for a graph $G$ and a set $P$ of pairs of nodes is a subgraph $H$ of $G$ satisfying $\delta_H(u, v) \leq \alpha \delta_G(u, v) + \beta$ for every $(u, v) \in P$.

To obtain our lower bound for any $\alpha \geq 1, \beta \geq 0$, we first study the tradeoff between the girth and number of edges in a graph. The most relevant claim for this question is Erdős’ girth conjecture:

**Conjecture 4.39 (Erdős’ Girth Conjecture [Erd64]).** For every $g$ there is a constant $c$ such that there exists a graph on $n$ nodes with girth $g$ and $cn^{1+\frac{1}{2g^{1/2}}} - 1$ edges.

For example, for $g = 3$, the complete graph on $n$ nodes has roughly $n^2/2$ edges, and for $g = 4$ the full bipartite graph has $n^2/4$ edges. For $g = 5$ and $g = 6$ there exist a graph with $n^{3/2}$ edges, which we used in the last section.

The conjecture is known to be true for a few values of $g$, while for the other values there are constructions with slightly less edges:
Theorem 4.40 ([LUW95], [LUW96], or see [Mat02, §15.3]). For every \( g \geq 3 \) and \( n \geq 2 \) there is a graph on \( n \) nodes with girth \( g \) and \( \Omega \left( n^{1+\frac{4}{9g-10}} \right) \) edges. For \( g \in \{3,4,5,6,9,10\} \) there is a constant \( c \) such that for every \( n \geq 2 \) there is a graph on \( n \) nodes with girth \( g \) and \( cn^{1+\frac{4}{9g-10}} \) edges.

Theorem 4.41. Let \( \alpha \geq 1, \beta \geq 0, \epsilon > 0 \) be constants, and \( g = 3\alpha + \beta \). Any distributed protocol for the Congest model with success probability at least 2/3 which, given a graph with \( n \) nodes and a set of \( p \in O \left( n^{1+\frac{4}{9g-10}-\epsilon} \right) \) pairs of nodes, outputs an \((\alpha, \beta)\)-pairwise spanner with \( m' \in O \left( n^{1+\frac{4}{9g-10}-\epsilon} \right) \) edges, must take \( \Omega \left( \frac{p}{n} \right) \) rounds to complete.

The bound on \( p \) and on the number of edges can be replaced by \( O \left( n^{1+\frac{4}{9g/2}-\epsilon} \right) \) for \( g \in \{3,4,5,6,9,10\} \), and for any constant \( g \) if Erdős’ girth conjecture is true.

Unfortunately, pairwise-spanners are not very well understood. Thus, it is not currently clear for which values of \((\alpha, \beta)\) other than \( \alpha = 1 \) spanners such as in the theorem even exist. Yet, for the spanners that exist, our theorem implies a bound on the time it takes to construct them.

The graph \( G \) for which the lower bound is proved is defined similarly to the graph in the proof of Theorem 4.36 with an extra construction that ensures that the diameter of \( G \) is constant. Let \( G' \) be a graph on \( n' \) nodes with girth \( g = 3\alpha + \beta \) and \( m \) edges, where \( m = m(n', g) \) is the maximal possible number of edges for these \( n' \) and \( g \). Let \( v'_1, \ldots, v'_{n'} \) be the nodes of \( G' \). Add to \( G' \) a new node \( u \) and connect each node \( v'_i \) to \( u \) by a disjoint path of \( \lceil g/2 \rceil \) new nodes. This increases the number of nodes in \( G' \) and the number of edges by a multiplicative \( g \) factor, does not decrease the girth, and ensures the diameter is \( O(g) \).

The lower bound graph \( G \) consists of \( G' \), the node \( u \), the nodes and paths connecting each \( v'_i \) to \( u \), and for each \( v'_i \) another node \( v_i \) connected to it by an edge \((v_i, v'_i)\). Let \( V_A = \{v_1, \ldots, v_{n'}\} \) and \( V_B \) be the set of all other nodes.

The pairwise spanner we construct approximately preserves distances between pairs of nodes in \( V_A \), i.e. \( \mathcal{P} \subseteq V_A \times V_A \). Since the girth of \( G' \) is \( 3\alpha + \beta \), if \( e' = \{v'_i, v'_j\} \) is an edge in \( G' \) and \((v_i, v_j) \in \mathcal{P} \), then any \((\alpha, \beta)\)-pairwise spanner must contain the edge \( e' \) in \( G \), the distance between \( v_i \) and \( v_j \) is 3; and if there is a path in \( H \) connecting \( v_i \) and \( v_j \) with length at most \( 3\alpha + \beta \), then this path connects \( v'_i \) and \( v'_j \) in \( G' \) with \( 3\alpha + \beta - 2 \) edges, which together with \( e' \) closes a cycle of length \( 3\alpha + \beta - 1 < g \) in \( G' \). On the other hand, if \((v_i, v_j) \notin \mathcal{P} \) then the edge \( e' \) can be omitted from the spanner.
Proof of Theorem 4.41. Fix a distributed protocol $\sigma$ for constructing an $(\alpha, \beta)$-pairwise spanner. Let $G$ be the graph described above, and denote the edges of $G'$ by $e_1, \ldots, e_m$. Theorem 4.40 gives the bound $m \geq 2cn^{1+\frac{1}{10g}}$ for some constant $c > 0$.

We use the same reduction from SUPERSET$_{m,m',p}$ to $\sigma$: Alice has a set $x \subseteq \{1, \ldots, m\}$ of size $p$, and Bob has to output a set $y \subseteq \{1, \ldots, m\}$ of size $m'$ satisfying $x \subseteq y$. They simulate $\sigma$ on the graph $G$ and the set of pairs $P = \{(v_i, v_j) : \exists k \in x \ e_k = \{v'_i, v'_j\}\}$; Alice simulates the nodes in $V_A$ and Bob simulates the nodes $V_B$ and the edges among them. To simulate communication on edges of the form $(v_i, v'_i)$, Alice and Bob communicate.

The spanner constructed is a subgraph $H$ of $G$ with at most $m'$ edges, satisfying $\delta_H(v_i, v_j) \leq \alpha \cdot \delta_G(v_i, v_j) + \beta$ for all $(v_i, v_j) \in P$. For each such pair, the edge $\{v'_i, v'_j\}$ must be in $H$, as explained above. Let $y = \{k : e_k \in E_H\}$. The spanner size implies $|y| \leq m'$, while the above discussion implies $x \subseteq y$. Thus, Bob can output a subset of $y$ of size $m'$, solving the communication complexity problem.

By the communication complexity lower bound, Alice and Bob must communicate $\Omega(p \log n)$ bits during the simulation. The number of edges they simulate together is at most $n/2$, and $O(\log n)$ bits are transmitted over each edge at each round. Thus, the protocol must take $\Omega(|P|/n)$ rounds to complete. 

\[ \square \]

4.4 Discussion

This thesis presents various algorithms for computing sparse purely additive spanners in the CONGEST model. For all-pairs spanners, our algorithms exhibit tradeoffs between the running time and the sparsity of the constructed spanners. By choosing different values for the parameter $h$, one can obtain a spanner with the same stretch in a smaller number of rounds but at the expense of increasing the density. This tradeoff is an important direction for future work.

Our lower bound uses a new communication complexity problem, and leverages the distributed nature of the system by using the fact that each node initially only knows the pairs in $P$ to which it belongs. That is, the topology of the graph used for the lower bound reduction is known completely to both Alice and Bob, regardless of their inputs to the SUPERSET$_{m,m',p}$ instance, while the uncertainty about the identity of the pairs in $P$ is what makes the problem hard. Our lower bound holds even if any node in $P$ knows all of $P$, but it fails if the set $P$ is given to all nodes. This raises the question of whether one can design a faster distributed constructions given this information, or whether there is a lower bound construction for this case as well.

Finally, we believe that our new lower bound technique can be useful for proving additional lower bounds in the CONGEST model, as it diverges from reducing to the set-disjointness problem.
Chapter 5

Quadratic and Near-Quadratic Lower Bounds for the CONGEST Model

5.1 Introduction

In this chapter, we present the first super-linear lower bounds for natural graph problems in the CONGEST model, answering a long-standing open question.

Specifically, we show that any exact computation of a minimum vertex cover or a maximum independent set requires $\Omega(n^2 / \log^2 n)$ rounds in the worst case in the CONGEST model, as well as any algorithm for $\chi$-coloring a graph, where $\chi$ is the chromatic number of the graph. We further show that such strong lower bounds are not limited to NP-hard problems, by showing two simple graph problems in P which require a quadratic and near-quadratic number of rounds.

Finally, we address the problem of computing an exact solution to weighted all-pairs-shortest-paths (APSP), which arguably may be considered as a candidate for having a super-linear lower bound. We show a simple $\Omega(n)$ lower bound for this problem, which implies a separation between the weighted and unweighted cases, since the latter is known to have a complexity of $\Theta(n / \log n)$. We also formally prove that the standard Alice-Bob framework is incapable of providing a super-linear lower bound for exact weighted APSP, whose complexity remains an intriguing open question.

5.1.1 The Challenge

Many lower bounds for the CONGEST model rely on reductions from 2-party communication problems (see, e.g., ACK16, PHW12, HW12, Nan14, PR00, DSHK12, DKO14, Elk06, NSP11, CKPY17). In this setting, two players, Alice and Bob, are given inputs of $K$ bits and need to a single output a bit according to some given function of their inputs. One of the most common problem for reduction is Set Disjointness, in which the players need
to decide whether there is an index for which both inputs are 1. That is, if the inputs represent subsets of \(\{0, \ldots, K-1\}\), the output bit of the players needs to indicate whether their input sets are disjoint. The communication complexity of 2-party Set Disjointness is known to be \(\Theta(K)\) \[KN97\].

In a nutshell, there are roughly two standard frameworks for reducing the 2-party communication problem of computing a function \(f\) to a problem \(P\) in the CONGEST model. One of these frameworks works as follows. A graph construction is given, which consists of some fixed edges and some edges whose existence depends on the inputs of Alice and Bob. This graph should have the property that a solution to \(P\) over it determines the solution to \(f\). Then, given an algorithm \(ALG\) for solving \(P\) in the CONGEST model, the vertices of the graph are split into two disjoint sets, \(V_A\) and \(V_B\), and Alice simulates \(ALG\) over \(V_A\) while Bob simulates \(ALG\) over \(V_B\). The only communication required between Alice and Bob in order to carry out this simulation is the content of messages sent in each direction over the edges of the cut \(C = E(V_A, V_B)\). Therefore, given a graph construction with a cut of size \(|C|\) and inputs of size \(K\) for a function \(f\) whose communication complexity on \(K\) bits is at least \(\text{CC}(f)\), the round complexity of \(ALG\) is at least \(\Omega(\text{CC}(f)/|C| \log n)\).

The challenge in obtaining super-linear lower bounds was that the cuts in the graph constructions were large compared with the input size \(K\). For example, the graph construction for the lower bound for computing the diameter in \[FHW12\] has \(K = \Theta(n^2)\) and \(|C| = \Theta(n)\), which gives an almost linear lower bound. The graph construction in \[FHW12\] for the lower bound for computing a \((3/2 - \epsilon)\)-approximation to the diameter has a smaller cut of \(|C| = \Theta(\sqrt{n})\), but this comes at the price of supporting a smaller input size \(K = \Theta(n)\), which gives a lower bound that is roughly a square-root of \(n\).

To overcome this difficulty, we leverage the recent framework of \[ACK16\], which provides a bit-gadget whose power is in allowing a logarithmic-size cut. We manage to provide a graph construction that supports inputs of size \(K = \Theta(n^2)\) in order to obtain our lower bounds for minimum vertex cover, maximum independent set and 3-coloring\(^1\). The latter is also inspired by, and is a simplification of, a lower bound construction for the size of proof-labelling schemes \[GS16\].

Further, for the problems in P that we address, the cut is as small as \(|C| = O(1)\). For one of the problems, the size of the input is such that it allows us to obtain the highest possible lower bound of \(\Omega(n^2)\) rounds.

With respect to the complexity of the weighted APSP problem, we show an embarrassingly simple graph construction that extends a construction of \[Nan14\], which leads to an \(\Omega(n)\) lower bound. However, we argue that a new technique must be developed in order to obtain any super-linear lower bound for weighted APSP. Roughly speaking, this is because given a construction with a set \(S\) of nodes that touch the cut, Alice and Bob can exchange \(O(|S| n \log n)\) bits which encode the weights of all lightest paths from any

\(^1\)It can also be shown, by simple modifications to our constructions, that these problems require \(\Omega(m)\) rounds, for graphs with \(m\) edges.
node in their set to a node in $S$. Since the cut has $\Omega(|S|)$ edges, and the bandwidth is $\Theta(\log n)$, this cannot give a lower bound of more than $\Omega(n)$ rounds. With some additional work, our proof can be carried over to a larger number of players at the price of a small logarithmic factor, as well as to the second Alice-Bob framework used in previous work (e.g. [DSHK+12]), in which Alice and Bob do not simulate nodes in a fixed partition, but rather in decreasing sets that partially overlap. Thus, determining the complexity of weighted APSP requires new tools, which we leave as a major open problem.

5.1.2 Additional Related Work

**Vertex Coloring, Minimum Vertex Cover, and Maximum Independent Set:** One of the most central problems in graph theory is vertex coloring, which has been extensively studied in the context of distributed computing (see, e.g., [BEPS16, Bar16, BE11, BE14, BEK14, Lin92, EPSW14, FGIP09, FHK16, HSS16, MW08, SW11, PS15, CPS14, CKP16, CV86, Bar12] and references therein). The special case of finding a $(\Delta + 1)$-coloring, where $\Delta$ is the maximum degree of a node in the network, has been the focus of many of these studies, but is a local problem, which can be solved in much less than a sublinear number of rounds.

Another classical problem in graph theory is finding a minimum vertex cover (MVC). In distributed computing, the time complexity of approximating MVC has been addressed in several cornerstone studies [˚AFP+09, BCS16, ˚AS10, GKP08, GKPS08, KYY94, KY09, KMW16, PS09, BEPS16, HKP01, PR01, KMW06].

Observe that finding a minimum size vertex cover is equivalent to finding a maximum size independent set. However, these problems are not equivalent in an approximation-preserving way. Distributed approximations for maximum independent set has been studied in [LW08, CHW08, BHKK16, BCGS17].

**Distance Computations:** Distance computation problems have been widely studied in the CONGEST model for both weighted and unweighted networks [ACK16, FHW12, HW12, HPRW14, PRTT2, HP15, LP15, LP13, Nan14, HFQ+16, HKN16]. One of the most fundamental problems of distance computations is computing all pairs shortest paths. For unweighted networks, an upper bound of $O(n/\log n)$ was recently shown by [HFQ+16], matching the lower bound of [FHW12]. Moreover, the possibility of bypassing this near-linear barrier for any constant approximation factor was ruled out by [Nan14]. For the weighted case, however, we are still very far from understanding the complexity of APSP, as there is still a huge gap between the upper and lower bounds. Recently, Elkin [Elk17] showed an $O(n^{5/3} \log^{2/3} n)$ upper bound for weighted APSP, while the previously highest lower bound was the near-linear lower bound of [Nan14] (which holds also for any (poly $n$)-approximation factor in the weighted case).

Distance computation problems have also been considered in the CLIQUE model [HKN16, CKK+15, HP15], in which the underlying communication network forms a clique. In
this model \cite{CKK+15} showed that unweighted APSP, and a \((1 + o(1))\)-approximation for weighted APSP, can be computed in \(O(n^{0.158})\) rounds.

**Subgraph Detection:** The problem of finding subgraphs of a certain topology has received a lot of attention in both the sequential and the distributed settings (see, e.g., \cite{AL13,WW13,DKS17,JP14,MP14,ALW14,DKO14,DLP12,AYZ97,CKK+15} and references therein). The problems of finding paths of length 4 or 5 with zero weight are also related to other fundamental problems, notable in our context is APSP \cite{AL13}.

### 5.2 Lower Bound Graphs

To prove lower bounds on the number of rounds necessary in order to solve a distributed problem in the \textsc{Congest} model, we use reductions from two-party communication complexity problems. To formalize them we use the following definition.

**Definition 5.1. (Family of Lower Bound Graphs)**

Fix an integer \(K\), a function \(f : \{0, 1\}^K \times \{0, 1\}^K \rightarrow \{\text{true}, \text{false}\}\) and a predicate \(P\) for graphs. The family of graphs \(\{G_{x,y} = (V, E_{x,y}) \mid x, y \in \{0, 1\}^K\}\), is said to be a family of lower bound graphs w.r.t. \(f\) and \(P\) if the following properties hold:

1. The set of nodes \(V\) is the same for all graphs, and we denote by \(V = V_A \cup V_B\) a fixed partition of it;
2. Only the existence or the weight of edges in \(V_A \times V_A\) may depend on \(x\);
3. Only the existence or the weight of edges in \(V_B \times V_B\) may depend on \(y\);
4. \(G_{x,y}\) satisfies the predicate \(P\) iff \(f(x, y) = \text{true}\).

We use the following theorem, which is standard in the context of communication complexity-based lower bounds for the \textsc{Congest} model (see, e.g. \cite{ACK16,FHW12,DKO14,HP15}). Its proof is by a standard simulation argument.

**Theorem 5.2.** Fix a function \(f : \{0, 1\}^K \times \{0, 1\}^K \rightarrow \{\text{true}, \text{false}\}\) and a predicate \(P\). If there is a family \(\{G_{x,y}\}\) of lower bound graphs with \(C = E(V_A, V_B)\) then any deterministic algorithm for deciding \(P\) in the \textsc{Congest} model requires \(\Omega(\frac{CC(f)}{|C| \log n})\) rounds, and any randomized algorithm for deciding \(P\) in the \textsc{Congest} model requires \(\Omega(\frac{CC_R(f)}{|C| \log n})\) rounds.

**Proof.** Let \(ALG\) be a distributed algorithm in the \textsc{Congest} model that decides \(P\) in \(T\) rounds. Given inputs \(x, y \in \{0, 1\}^K\) to Alice and Bob, respectively, Alice constructs the part of \(G_{x,y}\) for the nodes in \(V_A\) and Bob does so for the nodes in \(V_B\). This can be done by items (1),(2) and (3) in Definition 5.1 and since \(G_{x,y}\) satisfies this definition. Alice and Bob simulate \(ALG\) by exchanging the messages that are sent during the algorithm.
between nodes of $V_A$ and nodes of $V_B$ in either direction. (The messages within each set of nodes are simulated locally by the corresponding player without any communication). Since item (4) in Definition 5.1 also holds, we have that Alice and Bob correctly output $f(x, y)$ based on the output of $ALG$. For each edge in the cut, Alice and Bob exchange $O(\log n)$ bits per round. Since there are $T$ rounds and $|C|$ edges in the cut, the number of bits exchanged in this protocol for computing $f$ is $O(T|C|\log n)$. The lower bounds for $T$ now follows directly from the lower bounds for $CC(f)$ and $CC^R(f)$.

In what follows, for each decision problem addressed, we describe a fixed graph construction $G = (V, E)$, which we then generalize to a family of graphs $\{G_{x,y} = (V_x,y, E_{x,y}) | x, y \in \{0, 1\}^K\}$, which we show to be a family lower bound graphs w.r.t. to some function $f$ and the required predicate $P$. By Theorem 5.2 and the known lower bounds for the 2-party communication problem, we deduce a lower bound for any algorithm for deciding $P$ in the CONGEST model.

**Remark:** For our constructions which use the Set Disjointness function as $f$, we need to exclude the possibilities of all-1 input vectors. This is for the sake of guaranteeing that the graphs are connected, in order to avoid trivial impossibilities. However, this restriction does not change the asymptotic bounds for Set Disjointness, since computing this function while excluding all-1 input vectors can be reduced to computing this function for inputs that are shorter by one bit (by having the last bit be fixed to 0).

### 5.3 Near-Quadratic Lower Bounds for NP-Hard Problems

#### 5.3.1 Minimum Vertex Cover

The first near-quadratic lower bound we present is for computing a minimum vertex cover, as stated in the following theorem.

**Theorem 5.3.** Any distributed algorithm in the CONGEST model for computing a minimum vertex cover or for deciding whether there is a vertex cover of a given size $M$ requires $\Omega(n^2/\log^2 n)$ rounds.

Finding the minimum size of a vertex cover is equivalent to finding the maximum size of a maximum independent set, because a set of nodes is a vertex cover if and only if its complement is an independent set. Thus, Theorem 5.4 is a direct corollary of Theorem 5.3

**Theorem 5.4.** Any distributed algorithm in the CONGEST model for computing a maximum independent set or for deciding whether there is an independent set of a given size requires $\Omega(n^2/\log^2 n)$ rounds.
Figure 5.1: The family of lower bound graphs for deciding the size of a vertex cover, with many edges omitted for clarity. The node $a_{k-1}^1$ is connected to all the nodes in $T_{A_1}$, and $a_1^2$ is connected to $t_0^{A_2}$ and to all the nodes in $F_{A_2}\setminus \{f_0^{A_2}\}$. Examples of edges from $b_0^1$ and $b_0^2$ to the bit-gadgets are also given. An additional edge, which is among the edges corresponding to the strings $x$ and $y$, is $\{b_0^1, b_0^2\}$, while the edge $\{a_0^1, a_0^2\}$ does not exist. Here, $x_{0,0} = 1$ and $y_{0,1} = 0$.

Observe that a lower bound for deciding whether there is a vertex cover of some given size $M$ or not implies a lower bound for computing a minimum vertex cover. This is because computing the size of a given subset of nodes can be easily done in $O(D)$ rounds using standard tools. Therefore, to prove Theorem 5.3 it is sufficient to prove its second part. We do so by describing a family of lower bound graphs with respect to the Set Disjointness function and the predicate $P$ that says that the graph has a vertex cover of size $M$. We begin with describing the fixed graph construction $G = (V, E)$ and then define the family of lower bound graphs and analyze its relevant properties.

**The fixed graph construction:** Let $k$ be a power of 2. The fixed graph (Figure 5.1) consists of four cliques of size $k$: $A_1 = \{a_i^1 \mid 0 \leq i \leq k - 1\}$, $A_2 = \{a_i^2 \mid 0 \leq i \leq k - 1\}$, $B_1 = \{b_i^1 \mid 0 \leq i \leq k - 1\}$ and $B_2 = \{b_i^2 \mid 0 \leq i \leq k - 1\}$. In addition, for each set $S \in \{A_1, A_2, B_1, B_2\}$, there are two corresponding sets of nodes of size $\log k$, denoted $F_S = \{f_h^S \mid 0 \leq h \leq \log k - 1\}$ and $T_S = \{t_h^S \mid 0 \leq h \leq \log k - 1\}$. The latter are called bit-gadgets and their nodes are bit-nodes.

The bit-nodes are partitioned into $2\log k$ 4-cycles: for each $h \in \{0, \ldots, \log k - 1\}$ and $\ell \in \{1, 2\}$, we connect the 4-cycle $(f_h^{A_1}, t_h^{A_1}, f_h^{B_2}, t_h^{B_2})$. Note that there are no edges between pairs of nodes denoted $f_h^S$, or between pairs of nodes denoted $t_h^S$.

The nodes of each set $S \in \{A_1, A_2, B_1, B_2\}$ are connected to nodes in the corresponding set of bit-nodes, according to their binary representation, as follows. Let $s_i^\ell$ be a node
in a set \( S \in \{A_1, A_2, B_1, B_2\} \), i.e. \( s \in \{a, b\} \), \( \ell \in \{1, 2\} \) and \( i \in \{0, \ldots, k - 1\} \), and let \( i_h \) denote the bit number \( h \) in the binary representation of \( i \). For such a node \( s_i^h \) define \( \text{bin}(s_i^h) = \{ f_S^h | i_h = 0 \} \cup \{ t_S^h | i_h = 1 \} \), and connect \( s_i^h \) by an edge to each of the nodes in \( \text{bin}(s_i^h) \). The next two claims address the basic properties of vertex covers of \( G \).

**Claim 5.5.** Any vertex cover of \( G \) must contain at least \( k - 1 \) nodes from each of the clique \( A_1, A_2, B_1 \) and \( B_2 \), and at least \( 4 \log k \) bit-nodes.

**Proof.** In order to cover all the edges of each if the cliques on \( A_1, A_2, B_1 \) and \( B_2 \), any vertex cover must contain at least \( k - 1 \) nodes of the clique. For each \( h \in \{0, \ldots, \log k - 1\} \) and \( \ell \in \{1, 2\} \), in order to cover the edges of the 4-cycle \((f_{A_1}^h, h_{A_1}^h, f_{B_2}^h, h_{B_2}^h)\), any vertex cover must contain at least two of the cycle nodes. □

**Claim 5.6.** If \( U \subseteq V \) is a vertex cover of \( G \) of size \( 4(k - 1) + 4 \log k \), then there are two indices \( i, j \in \{0, \ldots, k - 1\} \) such that \( a_i^1, a_j^2, b_i^1, b_j^2 \) are not in \( U \).

**Proof.** By Claim [5.5], \( U \) must contain \( k - 1 \) nodes from each clique \( A_1, A_2, B_1 \) and \( B_2 \), and \( 4 \log k \) bit-nodes, so it must not contain one node from each clique. Let \( a_i^1, a_j^2, b_i^1, b_j^2 \) be the nodes in \( A_1, A_2, B_1 \) and \( B_2 \) which are not in \( U \), respectively. To cover the edges connecting \( a_i^1 \) to \( \text{bin}(a_i^1) \), \( U \) must contain all the nodes of \( \text{bin}(a_i^1) \), and similarly, \( U \) must contain all the nodes of \( \text{bin}(b_j^2) \). If \( i \neq i' \) then there is an index \( h \in \{0, \ldots, \log k - 1\} \) such that \( i_h \neq i_h' \), so one of the edges \((f_{A_1}^h, h_{B_2}^h)\) or \((h_{A_1}^h, f_{B_2}^h)\) is not covered by \( U \). Thus, it must hold that \( i = i' \). A similar argument shows \( j = j' \). □

**Adding edges corresponding to the strings \( x \) and \( y \):** Given two binary strings \( x, y \in \{0, 1\}^{k^2} \), we augment the graph \( G \) defined above with additional edges, which defines \( G_{x,y} \). Assume that \( x \) and \( y \) are indexed by pairs of the form \((i,j) \in \{0, \ldots, k - 1\}^2 \). For each such pair \((i,j)\) we add to \( G_{x,y} \) the following edges. If \( x_{i,j} = 0 \), then we add an edge between the nodes \( a_i^1 \) and \( a_j^2 \), and if \( y_{i,j} = 0 \) then we add an edge between the nodes \( b_i^1 \) and \( b_j^2 \). To prove that \( \{G_{x,y}\} \) is a family of lower bound graphs, it remains to prove the next lemma.

**Lemma 5.7.** The graph \( G_{x,y} \) has a vertex cover of cardinality \( M = 4(k - 1) + 4 \log k \) iff \( \text{DISJ}(x, y) = \text{false} \).

**Proof.** For the first implication, assume that \( \text{DISJ}(x, y) = \text{false} \) and let \( i, j \in \{0, \ldots, k - 1\} \) be such that \( x_{i,j} = y_{i,j} = 1 \). Note that in this case \( a_i^1 \) is not connected to \( a_j^2 \), and \( b_i^1 \) is not connected to \( b_j^2 \). We define a set \( U \subseteq V \) as the union of the two sets of nodes \((A_1 \setminus \{a_i^1\}) \cup (A_2 \setminus \{a_j^2\}) \cup (B_1 \setminus \{b_i^1\}) \cup (B_2 \setminus \{b_j^2\}) \) and \( \text{bin}(a_i^1) \cup \text{bin}(a_j^2) \cup \text{bin}(b_i^1) \cup \text{bin}(b_j^2) \), and show that \( U \) is a vertex cover of \( G_{x,y} \).

First, \( U \) covers all the edges inside the cliques \( A_1, A_2, B_1 \) and \( B_2 \), as it contains \( k - 1 \) nodes from each clique. These nodes also cover all the edges connecting nodes in \( A_1 \) to nodes in \( A_2 \) and all the edges connecting nodes in \( B_1 \) to nodes in \( B_2 \). Furthermore, \( U \)
covers any edge connecting some node \( u \in (A_1 \setminus \{a'_1\}) \cup (A_2 \setminus \{a'_2\}) \cup (B_1 \setminus \{b'_1\}) \cup (B_2 \setminus \{b'_2\}) \) with the bit-gadgets. For each node \( s \in a'_1, a'_2, b'_1, b'_2 \), the nodes bin\( (s) \) are in \( U \), so \( U \) also cover the edges connecting \( s \) to the bit gadget. Finally, \( U \) covers all the edges inside the bit gadgets, as from each 4-cycle \((f^h_{A_1}, t^h_{A_1}, f^h_{B_1}, t^h_{B_1})\) it contains two non-adjacent nodes: if \( i_h = 0 \) then \( f^h_{A_1}, t^h_{B_1} \in U \) and otherwise \( t^h_{A_1}, t^h_{B_1} \in U \), and if \( j_h = 0 \) then \( f^h_{A_2}, f^h_{B_2} \in U \) and otherwise \( t^h_{A_2}, t^h_{B_2} \in U \). We thus have that \( U \) is a vertex cover of size \( 4(k - 1) + 4 \log k \), as needed.

For the other implication, let \( U \subseteq V \) be a vertex cover of \( G_{x,y} \) of size \( 4(k - 1) + 4 \log k \). As the set of edges of \( G \) is contained in the set of edges of \( G_{x,y} \), \( U \) is also a cover of \( G \), and by Claim 5.6 there are indices \( i, j \in \{0, \ldots, k - 1\} \) such that \( a'_1, a'_2, b'_1, b'_2 \) are not in \( U \). Since \( U \) is a cover, the graph does not contain the edges \((a'_1, a'_2)\) and \((b'_1, b'_2)\), so we conclude that \( x_{i,j} = y_{i,j} = 1 \), which implies that \( \text{DISJ}(x, y) = \text{false} \).

Having constructed the family of lower bound graphs, we are now ready to prove Theorem 5.3.

**Proof of Theorem 5.3:** To complete the proof of Theorem 5.3, we divide the nodes of \( G \) (which are also the nodes of \( G_{x,y} \)) into two sets. Let \( V_A = A_1 \cup A_2 \cup F_{A_1} \cup T_{A_1} \cup F_{A_2} \cup T_{A_2} \) and \( V_B = V \setminus V_A \). Note that \( n \in \Theta(k) \), and thus \( K = |x| = |y| = \Theta(n^2) \). Furthermore, note that the only edges in the cut \( E(V_A, V_B) \) are the edges between nodes in \( \{F_{A_1} \cup T_{A_1} \cup F_{A_2} \cup T_{A_2}\} \) and nodes in \( \{F_{B_1} \cup T_{B_1} \cup F_{B_2} \cup T_{B_2}\} \), which are in total \( \Theta(\log n) \) edges. Since Lemma 5.7 shows that \( \{G_{x,y}\} \) is a family of lower bound graphs, we can apply Theorem 5.2 on the above partition to deduce that because of the lower bound for Set Disjointness, any algorithm in the CONGEST model for deciding whether a given graph has a cover of cardinality \( M = 4(k - 1) + 4 \log k \) requires at least \( \Omega(K/\log^2(n)) = \Omega(n^2/\log^2(n)) \) rounds.

### 5.3.2 Graph Coloring

Given a graph \( G \), we denote by \( \chi(G) \) the minimal number of colors in a proper vertex-coloring of \( G \). In this section we consider the problems of coloring a graph in \( \chi \) colors, computing \( \chi \) and approximating it. We prove the next theorem.

**Theorem 5.8.** Any distributed algorithm in the CONGEST model that colors a \( \chi \)-colorable graph \( G \) in \( \chi \) colors or compute \( \chi(G) \) requires \( \Omega(n^2/\log^2 n) \) rounds.

Any distributed algorithm in the CONGEST model that decides if \( \chi(G) \leq c \) for a given integer \( c \), requires \( \Omega((n - c)^2/(c \log n + \log^2 n)) \) rounds.

We give a detailed lower bound construction for the first part of the theorem, by showing that distinguishing \( \chi \leq 3 \) from \( \chi \geq 4 \) is hard. Then, we extend our construction to deal with deciding whether \( \chi \leq c \).
Figure 5.2: The family of lower bound graphs for coloring, with many edges omitted for clarity. The node $C^a_1$ is connected to all the nodes in $F_{A_1} \cup T_{A_1}$ and $C^a_1$ is connected to all the nodes in $F_{B_1} \cup T_{B_1}$. The node $C^a_2$ is connected to all the nodes in $F_{A_2} \cup T_{A_2}$ and $C^b_2$ is connected to all the nodes in $F_{B_2} \cup T_{B_2}$.

The fixed graph construction: We describe a family of lower bound graphs, which builds upon the family of graphs defined in Section 5.3.1. We define $G = (V, E)$ as follows (see Figure 5.2).

There are four sets of size $k$: $A_1 = \{a^i_1 | 0 \leq i \leq k - 1\}$, $A_2 = \{a^i_2 | 0 \leq i \leq k - 1\}$, $B_1 = \{b^i_1 | 0 \leq i \leq k - 1\}$ and $B_2 = \{b^i_2 | 0 \leq i \leq k - 1\}$. As opposed to the construction in Section 5.3.1, the nodes of these sets are not connected to one another. In addition, as in Section 5.3.1, for each set $S \in \{A_1, A_2, B_1, B_2\}$, there are two corresponding sets of nodes of size $\log k$, denoted $F_S = \{f^h_S | 0 \leq h \leq \log k - 1\}$ and $T_S = \{t^h_S | 0 \leq h \leq \log k - 1\}$. For each $h \in \{0, \ldots, \log k - 1\}$, the nodes $(f^h_{A_1}, t^h_{A_1}, f^h_{B_1}, t^h_{B_1})$ constitute a 4-cycle. Each node $s^i_\ell$ in a set $S \in \{A_1, A_2, B_1, B_2\}$ is connected to all nodes in $\text{bin}(s^i_\ell)$.

Up to here, the construction differs from the construction in Section 5.3.1 only by not having edges inside the sets $A_1, A_2, B_1, B_2$.

We now add the following two gadgets to the graph.

1. We add three nodes $C^0_a, C^1_a, C^2_a$ connected as a triangle, another set of three nodes $C^0_b, C^1_b, C^2_b$ connected as a triangle, and edges connecting $C^i_a$ to $C^j_b$ for each $i \neq j \in \{0, 1, 2\}$. We connect all the nodes of the form $f^h_{A_1}, t^h_{A_1}, h \in \{0, \ldots, \log k - 1\}$, to
Similarly, we connect all the nodes $f_{B_1}^h, t_{B_1}^h$ to $C_a^1$, the nodes $f_{A_2}^h, t_{A_2}^h$ to $C_a^2$ and the nodes $f_{B_2}^h, t_{B_2}^h$ to $C_b^2$.

2. For each set $S \in \{A_1, A_2, B_1, B_2\}$, we add two sets of nodes, $S = \{s_i^j \mid s_i^j \in S\}$ and $\bar{S} = \{s_i^j \mid s_i^j \in S\}$. For each $\ell \in \{1, 2\}$ and $i \in \{0, \ldots, k-1\}$ we connect a path $(s_i^\ell, s_i^j, \bar{s}_i^j, \bar{s}_i^\ell)$, and for each $\ell \in \{1, 2\}$ and $i \in \{0, \ldots, n-2\}$, we connect $s_i^\ell$ to $s_i^{\ell+1}$.

In addition, we connect the gadgets by the edges:

(a) $(C_a^0, a_i^1)$ and $(C_a^1, \bar{a}_i^1)$, for each $i \in \{0, \ldots, k-1\}$; $(C_a^2, a_i^0)$ and $(C_a^2, \bar{a}_i^{k-1})$.

(b) $(C_b^0, b_i^1)$ and $(C_b^1, \bar{b}_i^1)$, for each $i \in \{0, \ldots, k-1\}$; $(C_b^0, b_i^0)$ and $(C_b^0, \bar{b}_i^{k-1})$.

(c) $(C_a^1, a_i^2)$ and $(C_a^2, \bar{a}_i^2)$, for each $i \in \{0, \ldots, k-1\}$; $(C_a^1, \bar{a}_i^0)$ and $(C_a^1, \bar{a}_i^{k-1})$.

(d) $(C_b^1, b_i^2)$ and $(C_b^2, \bar{b}_i^2)$, for each $i \in \{0, \ldots, k-1\}$; $(C_b^0, b_i^0)$ and $(C_b^0, \bar{b}_i^{k-1})$.

Assume there is a proper 3-coloring of $G$. Denote by $c_0, c_1$ and $c_2$ the colors of $C_a^0, C_a^1$ and $C_a^2$ respectively. By construction, these are also the colors of $C_b^0, C_b^1$ and $C_b^2$, respectively. For the nodes appearing in Section 5.3.1, coloring a node by $c_0$ is analogous to not including it in the vertex cover.

**Claim 5.9.** In each set $S \in \{A_1, A_2, B_1, B_2\}$, at least one node is colored by $c_0$.

**Proof.** We start by proving the claim for $S = A_1$. Assume, towards a contradiction, that all nodes of $A_1$ are colored by $c_1$ and $c_2$. All these nodes are connected to $C_a^2$, so they must all be colored by $c_1$. Hence, all the nodes $\bar{a}_i^1, i \in \{0, \ldots, k-1\}$, are colored by $c_0$ and $c_2$. The nodes $\bar{a}_i^1, i \in \{0, \ldots, k-1\}$, are connected to $C_a^1$, so they are colored by $c_0$ and $c_2$ as well.

Hence, we have a path $(\bar{a}_i^1, \bar{a}_0^1, \bar{a}_1^1, \bar{a}_1^1, \ldots, \bar{a}_i^{k-1}, \bar{a}_i^{k-1})$ with an even number of nodes, starting in $\bar{a}_0^1$ and ending in $\bar{a}_i^{k-1}$. This path must be colored by alternating $c_0$ and $c_2$, but both $\bar{a}_0^1$ and $\bar{a}_i^{k-1}$ are connected to $C_a^2$, so they cannot be colored by $c_2$, a contradiction.

A similar proof shows the claim for $S = B_1$. For $S \in \{A_2, B_2\}$, we use a similar argument but change the roles of $c_1$ and $c_2$. \hfill \Box

**Claim 5.10.** For each $i \in \{0, \ldots, k-1\}$, the node $a_i^1$ is colored by $c_0$ iff $b_i^1$ is colored by $c_0$ and the node $a_i^2$ is colored by $c_0$ iff $b_i^2$ is colored by $c_0$.

**Proof.** Assume $a_i^1$ is colored by $c_0$, so all of its adjacent nodes $\bin(a_i^1)$ can only be colored by $c_1$ or $c_2$. As all of these nodes are connected to $C_a^0$, they must be colored by $c_2$. Similarly, if a node $b_i^1$ in $B_1$ is colored by $c_0$, then the nodes $\bin(b_i^1)$, which are also adjacent to $C_b^0$, must be colored by $c_2$.

If $i \neq j$ then there must be a bit $i$ such that $i_h \neq j_h$, and there must be a pair of neighboring nodes $(f_{A_1}^h, t_{B_1}^h)$ or $(f_{A_2}^h, t_{B_2}^h)$ which are colored by $c_2$. Thus, the only option is $i = j$. By Claim 5.9, there is a node in $B_1$ that is colored by $c_0$, and so it must be $b_i^1$.

An analogous argument shows that if $b_i^1$ is colored by $c_0$, then so does $a_i^1$. For $a_i^2$ and $b_i^2$, similar arguments apply, where $c_1$ plays the role of $c_2$. \hfill \Box
Adding edges corresponding to the strings $x$ and $y$: Given two bit strings $x, y \in \{0,1\}^k$, we augment the graph $G$ described above with additional edges, which defines $G_{x,y}$.

Assume $x$ and $y$ are indexed by pairs of the form $(i,j) \in \{0,\ldots,k-1\}^2$. To construct $G_{x,y}$, add edges to $G$ by the following rules: if $x_{i,j} = 0$ then add the edge $(a_1^i, a_2^j)$, and if $y_{i,j} = 0$ then add the edge $(b_1^i, b_2^j)$. To prove that $\{G_{x,y}\}$ is a family of lower bound graphs, it remains to prove the next lemma.

Lemma 5.11. The graph $G_{x,y}$ is 3-colorable iff $\text{DISJ}(x,y) = \text{FALSE}$.

Proof. For the first direction, assume $G_{x,y}$ is 3-colorable, and denote the colors by $c_0$, $c_1$ and $c_2$, as before. By Claim 5.9, there are nodes $a_1^i \in A_1$ and $a_2^j \in A_2$ that are both colored by $c_0$. Hence, the edge $(a_1^i, a_2^j)$ does not exist in $G_{x,y}$, implying $x_{i,j} = 1$. By Claim 5.10 the nodes $b_1^i$ and $b_2^j$ are also colored $c_0$, so $y_{i,j} = 1$ as well, giving that $\text{DISJ}(x,y) = \text{FALSE}$, as needed.

For the other direction, assume $\text{DISJ}(x,y) = \text{FALSE}$, i.e., there is an index $(i,j) \in \{0,\ldots,k-1\}^2$ such that $x_{i,j} = y_{i,j} = 1$. Consider the following coloring.

1. Color $C_a^i$ and $C_b^i$ by $c_i$, for $i \in \{0,1,2\}$.

2. Color the nodes $a_1^i, b_1^i, a_2^j$ and $b_2^j$ by $c_0$. Color the nodes $a_1^{i'}$ and $b_1^{i'}$, for $i' \neq i$, by $c_1$, and the nodes $a_2^{j'}$ and $b_2^{j'}$, for $j' \neq j$, by $c_2$.

3. Color the nodes of $\text{bin}(a_1^i)$ by $c_2$, and similarly color the nodes of $\text{bin}(b_1^i)$ by $c_2$. Color the rest of the nodes in this gadget, i.e. $\text{bin}(a_1^{k-i})$ and $\text{bin}(b_1^{k-i})$, by $c_0$. Similarly, color $\text{bin}(a_2^j)$ and $\text{bin}(b_2^j)$ by $c_0$ and $\text{bin}(a_2^{k-j})$ and $\text{bin}(b_2^{k-j})$ by $c_1$.

4. Finally, color the nodes of the forms $\bar{s}_i^i$ and $\bar{s}_j^j$ as follows.

   (a) Color $\bar{a}_1^i$ and $\bar{b}_1^i$ by $c_1$, all nodes $\bar{a}_1^{i'}$ and $\bar{b}_1^{i'}$ with $i' < i$ by $c_0$, and all nodes $\bar{a}_1^{i'}$ and $\bar{b}_1^{i'}$ with $i' > i$ by $c_2$.

   (b) Similarly, color $\bar{a}_2^i$ and $\bar{b}_2^i$ by $c_2$, all nodes $\bar{a}_2^{i'}$ and $\bar{b}_2^{i'}$ with $i' < i$ by $c_0$, and all nodes $\bar{a}_2^{i'}$ and $\bar{b}_2^{i'}$ with $i' > i$ by $c_1$.

   (c) Color all nodes $\bar{a}_1^{i'}$ and $\bar{b}_1^{i'}$ with $i' < i$ by $c_2$, and all nodes $\bar{a}_1^{i'}$ and $\bar{b}_1^{i'}$ with $i' \geq i$ by $c_0$.

   (d) Similarly, color all nodes $\bar{a}_2^{i'}$ and $\bar{b}_2^{i'}$ with $i' < i$ by $c_1$, and all nodes $\bar{a}_2^{i'}$ and $\bar{b}_2^{i'}$ with $i' \geq i$ by $c_0$.

It is not hard to verify that the suggested coloring is indeed a proper 3-coloring of $G_{x,y}$, which completes the proof.

Having constructed the family of lower bound graphs, we are now ready to prove Theorem 5.8.
Proof of Theorem 5.8: To complete the proof of Theorem 5.8, we divide the nodes of G (which are also the nodes of $G_{x,y}$) into two sets. Let $V_A = A_1 \cup A_2 \cup F_{A_1} \cup T_{A_1} \cup F_{A_2} \cup T_{A_2} \cup \{C^0_a, C^1_a, C^2_a\} \cup \tilde{A}_1 \cup \tilde{A}_2 \cup \tilde{A}_2$, and $V_B = V \setminus V_A$. Note that $n \in \Theta(k)$.

The edges in the cut $E(V_A, V_B)$ are the 6 edges connecting $\{C^0_a, C^1_a, C^2_a\}$ and $\{C^0_b, C^1_b, C^2_b\}$, and 2 edges for every 4-cycle of the nodes of $F_{A_1} \cup T_{A_1} \cup F_{B_1} \cup T_{B_1}$ and $F_{A_2} \cup T_{A_2} \cup F_{B_2} \cup T_{B_2}$, for a total of $\Theta(\log n)$ edges. Since Lemma 5.11 shows that $\{G_{x,y}\}$ is a family of lower bound graphs with respect to $\text{DISJ}_K$, $K = k^2 \in \Theta(n^2)$ and the predicate $\chi \leq 3$, we can apply Theorem 5.2 on the above partition to deduce that any algorithm in the CONGEST model for deciding whether a given graph is 3-colorable requires at least $\Omega(n^2/\log^2 n)$ rounds.

Any algorithm that computes $\chi$ of the input graph, or produces a $\chi$-coloring of it, may be used to deciding whether $\chi \leq 3$, in at most $O(D)$ additional rounds. Thus, the lower bound applies to these problems as well.

Our construction and proof naturally extend to handle $c$-coloring, for any $c \geq 3$. To this end, we add to $G$ (and to $G_{x,y}$) new nodes denoted $C^i_a$, $i \in \{3, \ldots, c - 1\}$, and connect them to all of $V_A$, and new nodes denoted $C^i_b$, $i \in \{3, \ldots, c - 1\}$, and connect them to all of $V_B$ and also to $C^0_a, C^1_a$ and $C^2_a$. The nodes $C^i_a$ are added to $V_A$, and the rest are added to $V_B$, which increases the cut size by $\Theta(c)$ edges.

Assume the extended graph is colorable by $c$ colors, and denote by $c_i$ the color of the node $C^i_a$ (these nodes are connected by a clique, so their colors are distinct). The nodes $C^i_b$, $i \in \{2, \ldots, c - 1\}$ form a clique, and they are all connected to the nodes $C^0_a, C^1_a$ and $C^2_a$, so they are colored by the colors $\{c_3, \ldots, c_{c-1}\}$, in some order. All the original nodes of $V_A$ are connected to $C^i_a$, $i \in \{3, \ldots, c - 1\}$, and all the original nodes of $V_B$ are connected to $C^i_b$, $i \in \{3, \ldots, c - 1\}$, so the original graph must be colored by 3 colors, which we know is possible iff $\text{DISJ}(x, y) = \text{false}$.

We added $2c - 6$ nodes to the graph, so the inputs strings are of length $K = n - 2c + 6$. Thus, the new graphs constitute a family of lower bound graphs with respect to $\text{EQ}_K$ and the predicate $\chi \leq c$, the communication complexity of $\text{EQ}_K$ is in $\Omega(K^2) = \Omega((n - c)^2)$, the cut size is $\Theta(c + \log n)$, and Theorem 5.2 completes the proof.

A lower bound for $(4/3 - \epsilon)$-approximation: Finally, we extend our construction to give a lower bound for approximate coloring. That is, we show a similar lower bound for computing a $(4/3 - \epsilon)$-approximation to $\chi$ and for finding a coloring in $(4/3 - \epsilon)\chi$ colors.

Observe that since $\chi$ is integral, any $(4/3 - \epsilon)$-approximation algorithm must return the exact solution in case $\chi = 3$. Thus, in order to rule out the possibility for an algorithm which is allowed to return a $(4/3 - \epsilon)$-approximation which is not the exact solution, we need a more general construction. For any integer $c$, we show a lower bound for distinguishing between the case $\chi \leq 3c$ and $\chi \geq 4c$.

Claim 5.12. Given an integer $c$, any distributed algorithm in the CONGEST model that distinguishes a graph $G$ with $\chi(G) \leq 3c$ from a graph with $\chi(G) \geq 4c$ requires
\(\Omega(n^2/(c^3 \log^2 n))\) rounds.

To prove Claim 5.12 we show a family of lower bound graphs with respect to the \(\text{DISJ}_K\) function, where \(K \in \Theta(n^2/c^2)\), and the predicate \(\chi \leq 3c\) (true) or \(\chi \geq 4c\) (false). The predicate is not defined for other values of \(\chi\).

We create a graph \(G_{c,x,y}\), composed of \(c\) copies of \(G_{x,y}\). The \(i\)-th copy is denoted \(G_{x,y}^i\), and its nodes are partitioned into \(V_A^i\) and \(V_B^i\). We connect all the nodes of \(V_A^i\) to all nodes of \(V_A^j\), for each \(i \neq j\). Similarly, we connect all the nodes of \(V_B^i\) to all the nodes of \(V_B^j\). This construction guarantees that each copy is colored by different colors, and hence if \(\text{DISJ}(x, y) = \text{false}\) then \(\chi(G_{c,x,y}^i) = 3c\) and otherwise \(\chi(G_{c,x,y}^i) \geq 3c\). Therefore, \(G_{c,x,y}^i\) is a family of lower bound graphs.

Proof of Claim 5.12. Note that \(n \in \Theta(kc)\). Thus, \(K = |x| = |y| = n^2/c^2\). Furthermore, observe that for each \(G_{x,y}^i\), there are \(O(\log n)\) edges in the cut, so in total \(G_{c,x,y}^i\) contains \(O(c \log n)\) edges in the cut. Since we showed that \(G_{c,x,y}^i\) is a family of lower bound graphs, we can apply Theorem 5.2 to deduce that because of the lower bound for Set Disjointness, any algorithm in the CONGEST model for distinguishing between \(\chi \leq 3c\) and \(\chi \geq 4c\) requires at least \(\Omega(n^2/c^3 \log^2 n)\) rounds. \(\square\)

For any \(\epsilon > 0\) and any \(c\) it holds that \((4/3 - \epsilon)3c < 4c\). Thus, we can choose \(c\) to be an arbitrary constant to achieve the following theorem.

Theorem 5.13. For any constant \(\epsilon > 0\), any distributed algorithm in the CONGEST model that computes a \((4/3 - \epsilon)\)-approximation to \(\chi\) requires \(\Omega(n^2/\log^2 n)\) rounds.

5.4 Quadratic and Near-Quadratic Lower Bounds for Problems in P

In this section we support our claim that what makes problems hard for the CONGEST model is not necessarily them being NP-hard problems. First, we address a class of subgraph detection problems, which requires detecting cycles of length 8 and a given weight, and show a near-quadratic lower bound on the number of rounds required for solving it, although its sequential complexity is polynomial. Then, we define a problem which we call the Identical Subgraphs Detection problem, in which the goal is to decide whether two given subgraphs are identical. While this last problem is rather artificial, it allows us to obtain a strictly quadratic lower bound for the CONGEST model, with a problem that requires only a single-bit output.

5.4.1 Weighted Cycle Detection

In this section we show a lower bound on the number of rounds needed in order to decide the graph contains a cycle of length 8 and weight \(W\), such that \(W\) is a polylog\((n)\)-bit
Figure 5.3: The family of lower bound graphs for detecting weighted cycles, with many edges omitted for clarity. Here, $x_{0,1} = 1$ and $y_{k-1,k-1} = 1$. Thus, $a_1^0$ is connected to $a_2^1$ by an edge of weight $k^3 + k \cdot 0 + 1 = k^3 + 1$, and $b_{k-1}^1$ is connected to $b_{k}^{k-1}$ by an edge of weight $k^3 - (k(k-1) + k - 1) = k^3 - k^2 + 1$. All the dashed edges are of weight 0.

value given as an input. Note that this problem can be solved easily in polynomial time in the sequential setting by simply checking all of the at most $\binom{n}{8}$ cycles of length 8.

**Theorem 5.14.** Any distributed algorithm in the **CONGEST** model that decides if a graph with edge weights $w : E \to [0, \text{poly}(n)]$ contains a cycle of length 8 and weight $W$ requires $\Omega\left(\frac{n^2}{\log^2 n}\right)$ rounds.

Similarly to the previous sections, to prove Theorem 5.14 we describe a family of lower bound graphs with respect to the Set Disjointness function and the predicate $P$ that says that the graph contains a cycle of length 8 and weight $W$.

**The fixed graph construction:** The fixed graph construction $G = (V, E)$ is defined as follows. The set of nodes contains four sets $A_1, A_2, B_1$ and $B_2$, each of size $k$. To simplify our proofs in this section, we assume that $k \geq 3$. For each set $S \in \{A_1, A_2, B_1, B_2\}$ there is a node $c_S$, which is connected to each of the nodes in $S$ by an edge of weight 0. In addition there is an edge between $c_{A_1}$ and $c_{B_1}$ of weight 0 and an edge between $c_{A_2}$ and $c_{B_2}$ of weight 0 (see Figure 5.3).

**Adding edges corresponding to the strings $x$ and $y$:** Given two binary strings $x, y \in \{0, 1\}^{k^2}$, we augment the fixed graph $G$ defined in the previous section with additional edges, which defines $G_{x,y}$. Recall that we assume that $k \geq 3$. Let $x$ and $y$ be indexed by pairs of the form $(i, j) \in \{0, \ldots, k-1\}^2$. For each $(i, j) \in \{0, \ldots, k-1\}^2$, we add to $G_{x,y}$ the following edges. If $x_{i,j} = 1$, then we add an edge of weight $k^3 + ki + j$ between the nodes $a_1^i$ and $a_2^j$. If $y_{i,j} = 1$, then we add an edge of weight $k^3 - (ki + j)$ between the nodes $b_1^i$ and $b_2^j$. We denote by $InputEdges$ the set of edges $\{(u, v) \mid u \in
A_1 \land v \in A_2 \} \cup \{(u,v) \mid u \in B_1 \land v \in B_2 \}, and we denote by w(u,v) the weight of the edge (u,v).

Observe that the graph does not contain edges of negative weight. Furthermore, the weight of any edge in InputEdges does not exceed k^3 + k^2 - 1, which is the weight of the edge (a_1^{k-1}, a_2^{k-1}), in case x_{k-1,k-1} = 1. Similarly, the weight of an edge in InputEdges is not less than k^3 - k^2 + 1, which is the weight of the edge (b_1^{k-1}, b_2^{k-1}), in case y_{k-1,k-1} = 1.

Using these two simple observations, we deduce the following claim.

Claim 5.15. For any cycle of weight 2k^3, the number of edges it contains that are in InputEdges is exactly two.

Proof. Let C be a cycle of weight 2k^3, and assume for the sake of contradiction that C does not contain exactly two edges from InputEdges. In case C contains exactly one edge from InputEdges, then the weight of C is at most k^3 + k^2 - 1 < 2k^3, because all the other edges of C are of weight 0. Otherwise, in case C contains three or more edges from InputEdges, it holds that the weight of C is at least 3k^3 - 3k^2 + 3 > 2k^3, because all the other edges on C are of non-negative weight.

To prove that \{G_{x,y}\} satisfies the definition of a family of lower bound graphs, we prove the following lemma.

Lemma 5.16. The graph G_{x,y} contains a cycle of length 8 and weight W = 2k^3 if and only if DISJ(x, y) = false.

Proof. For the first direction, assume that DISJ(x, y) = false and let 0 \leq i, j \leq k - 1 be such that x_{i,j} = 1 and y_{i,j} = 1. Consider the cycle (a_1^i, c_{A_1}, c_{B_1}, b_1^j, c_{B_2}, c_{A_2}, a_2^j). It is easy to verify that this is a cycle of length 8 and weight w(a_1^i, a_2^j) + w(b_1^j, b_2^j) = k^3 + ki + j + k^3 - ki - j = 2k^3, as needed.

For the other direction, assume that the graph contains a cycle C of length 8 and weight 2k^3. By Claim 5.15 C contains exactly two edges from InputEdges. Denote these two edges by (u_1, v_1) and (u_2, v_2). Since all the other edges in C are of weight 0, the weight of C is w(u_1, v_1) + w(u_2, v_2). The rest of the proof is by case analysis, as follows. First, it is not possible that (u_1, v_1), (u_2, v_2) \in \{(u,v) \mid u \in A_1 \land v \in A_2 \}, since in this case w(u_1, v_1) + w(u_2, v_2) \leq w(a_1^0, a_2^0) + w(a_1^0, a_2^0) = 2k^3 + 1. Similarly, it is not possible that (u_1, v_1), (u_2, v_2) \in \{(u,v) \mid u \in B_1 \land v \in B_2 \}, since in this case w(u_1, v_1) + w(u_2, v_2) \leq w(b_1^0, b_2^0) + w(b_1^0, b_2^0) = 2k^3 - 1. Finally, suppose without loss of generality that (u_1, v_1) \in \{(u,v) \mid u \in A_1 \land v \in A_2 \} and (u_2, v_2) \in \{(u,v) \mid u \in B_1 \land v \in B_2 \}. Denote u_1 = a_1^i, u_2 = a_1^j, v_1 = b_1^j and v_2 = b_2^i. It holds that w(a_1^i, a_2^j) + w(b_1^j, b_2^i) = 2k^3 if and only if i = i' and j = j', which implies that x_{i,j} = 1 and y_{i,j} = 1 and DISJ(x, y) = false.

Having constructed the family of lower bound graphs, we are now ready to prove Theorem 5.14.
**Proof of Theorem 5.14** To complete the proof of Theorem 5.14, we divide the nodes of $G$ (which are also the nodes of $G_{x,y}$) into two sets. Let $V_A = A_1 \cup A_2 \cup \{c_{A_1}, c_{A_2}\}$ and $V_B = V \setminus V_A$. Note that $n \in \Theta(k)$. Thus, $K = |x| = |y| = \Theta(n^2)$. Furthermore, note that the only edges in the cut $E(V_A, V_B)$ are the edges $(c_{A_1}, c_{B_1})$ and $(c_{A_2}, c_{B_2})$. Since Lemma 5.16 shows that $\{G_{x,y}\}$ is a family of lower bound graphs, we apply Theorem 5.2 on the above partition to deduce that because of the lower bound for Set Disjointness, any algorithm in the CONGEST model for deciding whether a given graph contains a cycle of length 8 and weight $W = 2k^3$ requires at least $\Omega(K/\log n) = \Omega(n^2/\log n)$ rounds.

5.4.2 Identical Subgraphs Detection

In this section we show the strongest possible, quadratic lower bound, for a problem which can be solved in linear time in the sequential setting.

Consider the following sequential specification of a graph problem.

**Definition 5.17.** *(The Identical Subgraphs Detection Problem)*

Given a weighted input graph $G = (V, E, w)$, with an edge-weight function $w : E \rightarrow \{0, \ldots, W - 1\}$, $W \in \text{poly} n$, such that the set of nodes $V$ is partitioned into two enumerated sets of the same size, $V_A = \{a_0, \ldots, a_{k-1}\}$ and $V_B = \{b_0, \ldots, b_{k-1}\}$, the Identical Subgraphs Detection problem is to determine whether the subgraph induced by the set $V_A$ is identical to the subgraph induced by the set $V_B$, in the sense that for each $0 \leq i, j \leq k - 1$ it holds that $(a_i, a_j) \in E$ if and only if $(b_i, b_j) \in E$ and $w(a_i, a_j) = w(b_i, b_j)$ if these edges exist.

The identical subgraphs detection problem can be solved easily in linear time in the sequential setting by a single pass over the set of edges. However, as we prove next, it requires a quadratic number of rounds in the CONGEST model, in any deterministic solution (note that this restriction did not apply in the previous sections). For clarity, we emphasize that in the distributed setting, the input to each node in the identical subgraphs detection problem is its enumeration $a_i$ or $b_i$, as well as the enumerations of its neighbors and the weights of the respective edges. The outputs of all nodes should be true if the subgraphs are identical, and false otherwise.

**Theorem 5.18.** Any distributed deterministic algorithm in the CONGEST model for solving the identical subgraphs detection problem requires $\Omega(n^2)$ rounds.

To prove Theorem 5.18 we describe a family of lower bound graphs.

**The fixed graph construction:** The fixed graph $G$ is composed of two $k$-node cliques on the node sets $V_A = \{a_0, \ldots, a_{k-1}\}$ and $V_B = \{b_0, \ldots, b_{k-1}\}$, and one extra edge $(a_0, b_0)$.  

74
Adding edge weights corresponding to the strings \( x \) and \( y \): Given two binary strings \( x \) and \( y \), each of size \( K = \left( \frac{n}{2} \right) \log n \), we augment the graph \( G \) with additional edge weights, which define \( G_{x,y} \). For simplicity, assume that \( x \) and \( y \) are vectors of \( \log n \)-bit numbers each having \( \frac{n}{2} \) entries enumerated as \( x_{i,j} \) and \( y_{i,j} \), with \( i < j \), \( i, j \in \{0, \ldots, K-1\} \). For each such \( i \) and \( j \) we set the weights of \( w(a_i, a_j) = x_{i,j} \) and \( w(b_i, b_j) = y_{i,j} \), and we set \( w(a_0, b_0) = 0 \). Note that \( \{G_{x,y}\} \) is a family of lower bound graphs with respect to \( \text{EQ}_K \) and the predicate \( P \) that says that the subgraphs are identical in the aforementioned sense.

Proof of Theorem 5.18: Note that \( n \in \Theta(k) \), and thus \( K = |x| = |y| = \Theta(n^2 \log n) \). Furthermore, the only edge in the cut \( E(V_A, V_B) \) is the edge \((a_0, b_0)\). Since we showed that \( \{G_{x,y}\} \) is a family of lower bound graphs, we can apply Theorem 5.2 on the above partition to deduce that because of the lower bound for \( \text{EQ}_K \), any deterministic algorithm in the \( \text{CONGEST} \) model for solving the identical subgraphs detection problem requires at least \( \Omega(K/\log n) = \Omega(n^2) \) rounds.

We remark that in a distributed algorithm for the identical subgraphs detection problem running on our family of lower bound graphs, information about essentially all the edges and weights in the subgraphs induced on \( V_A \) and \( V_B \) needs to be sent across the edge \((a_0, b_0)\). This might raise the suspicion that this problem is reducible to learning the primes uniformly at random. She treats her input string \( x \) as a binary representation of an integer \( \bar{x} = \sum_{\ell=0}^{K-1} 2^\ell x_\ell \), and sends \( p \) and \( \bar{x} \mod p \) to Bob. Bob similarly computes \( \bar{y} \), compares \( \bar{x} \mod p \) with \( \bar{y} \mod p \), and returns \text{true} if they are equal and \text{false} otherwise. The error probability of this protocol is at most \( 1/K \).

Our starting point is the following randomized algorithm for the \( \text{EQ}_K \) problem, presented in \cite{KN97} Exercise 3.6. Alice chooses a prime number \( p \) among the first \( K^2 \) primes uniformly at random. She treats her input string \( x \) as a binary representation of an integer \( \bar{x} = \sum_{\ell=0}^{K-1} 2^\ell x_\ell \), and sends \( p \) and \( \bar{x} \mod p \) to Bob. Bob similarly computes \( \bar{y} \), compares \( \bar{x} \mod p \) with \( \bar{y} \mod p \), and returns \text{true} if they are equal and \text{false} otherwise. The error probability of this protocol is at most \( 1/K \).

We present a simple adaptation of this algorithm for the identical subgraph detection problem. Consider the following encoding of a weighted induced subgraph on \( V_A \): for each pair \( i, j \) of indices, we have \( \lceil \log W \rceil + 1 \) bits, indicating the existence of the edge and its weight (recall that \( W \in \text{poly} n \) is an upper bound on the edge weights). This weighted induced subgraph is thus represented by a \( K \in O(n^2 \log n) \) bit-string, denoted \( x = x_0, \ldots, x_{K-1} \), and each pair \((i, j)\) has a set \( S_{i,j} \) of indices representing the edge \((a_i, a_j)\). Note that the bits \( \{x_\ell \mid \ell \in S_{i,j}\} \) are known to both \( a_i \) and \( a_j \), and in the algorithm we use the node with smaller index in order to encode these bits. Similarly, a \( K \in O(n^2 \log n) \) bit-string, denoted \( y = y_0, \ldots, y_{K-1} \) encodes a weighted induced subgraph on \( V_B \).
The Algorithm. As standard, assume the input graph is connected. The nodes are enumerated as in Definition 5.17. The algorithm starts with some node, say, \( a_0 \), constructing a BFS tree, which completes in \( O(D) \) rounds. Then, \( a_0 \) chooses a prime number \( p \) among the first \( K^2 \) primes uniformly at random and sends \( p \) to all the nodes over the tree, which takes \( O(D) \) rounds.

Each node \( a_i \) computes the sum \( \sum_{j>i} \sum_{\ell \in S_{i,j}} x_\ell 2^\ell \mod p \), and the nodes then aggregate these local sums modulo \( p \) up the tree, until \( a_0 \) computes the sum \( \bar{x} \mod p = \sum_{j \neq i} \sum_{\ell \in S_{i,j}} x_\ell 2^\ell \mod p \). A similar procedure is then invoked w.r.t \( \bar{y} \). Finally, \( a_0 \) compares \( \bar{x} \mod p \) and \( \bar{y} \mod p \), and downcasts over the BFS tree its output, which is \( \text{TRUE} \) if these values are equal and is \( \text{FALSE} \) otherwise.

If the subgraphs are identical, \( a_0 \) always returns \( \text{TRUE} \), while otherwise their encoding differs in at least one bit, and as in the case of \( \text{EQ}_K \), \( a_0 \) returns \( \text{TRUE} \) falsely with probability at most \( 1/K \in O(1/n^2) \).

**Theorem 5.19.** There is a randomized algorithm in the Congest model that solves the identical subgraphs detection problem on any connected graph in \( O(D) \) rounds.

5.5 Weighted APSP

In this section we use the following, natural extension of Definition 5.1, in order to address more general 2-party functions, as well as distributed problems that are not decision problems.

For a function \( f : \{0, 1\}^{K_1} \times \{0, 1\}^{K_2} \to \{0, 1\}^{L_1} \times \{0, 1\}^{L_2} \), we define a family of lower bound graphs in a similar way as Definition 5.1, except that we replace item (4) in the definition with a generalized requirement that says that for \( G_{x,y} \), the values of the of nodes in \( V_A \) uniquely determine the left-hand side of \( f(x,y) \), and the values of the of nodes in \( V_B \) determine the right-hand side of \( f(x,y) \). Next, we argue that theorem similar to Theorem 5.2 holds for this case.

**Theorem 5.20.** Fix a function \( f : \{0, 1\}^{K_1} \times \{0, 1\}^{K_2} \to \{0, 1\}^{L_1} \times \{0, 1\}^{L_2} \) and a graph problem \( P \). If there is a family \( \{G_{x,y}\} \) of lower bound graphs with \( C = E(V_A, V_B) \) then any deterministic algorithm for solving \( P \) in the Congest model requires \( \Omega(CC(f)/|C| \log n) \) rounds, and any randomized algorithm for deciding \( P \) in the Congest model requires \( \Omega(CC^R(f)/|C| \log n) \) rounds.

The proof is similar to that of Theorem 5.2. Notice that the only difference between the theorems, apart from the sizes of the inputs and outputs of \( f \), are with respect to item (4) in the definition of a family of lower bound graphs. However, the essence of this condition remains the same and is all that is required by the proof: the values that a solution to \( P \) assigns to nodes in \( V_A \) determines the output of Alice for \( f(x,y) \), and the values that a solution to \( P \) assigns to nodes in \( V_B \) determines the output of Bob for \( f(x,y) \).
5.5.1 A Linear Lower Bound for Weighted APSP

Nanongkai [Nan14] showed that any algorithm in the CONGEST model for computing a poly\((n)\)-approximation for weighted all pairs shortest paths (APSP) requires at least \(\Omega\left(\frac{n}{\log n}\right)\) rounds. In this section we show that a slight modification to this construction yields an \(\Omega(n)\) lower bound for computing exact weighted APSP. As explained in the introduction, this gives a separation between the complexities of the weighted and unweighted versions of APSP. At a high level, while we use the same simple topology for our lower bound as in [Nan14], the reason that we are able to shave off the extra logarithmic factor is because our construction uses \(O(\log n)\) bits for encoding the weight of each edge out of many optional weights, while in [Nan14] only a single bit is used per edge for encoding one of only two options for its weight.

**Theorem 5.21.** Any distributed algorithm in the CONGEST model for computing exact weighted all pairs shortest paths requires at least \(\Omega(n)\) rounds.

The reduction is from the following, perhaps simplest, 2-party communication problem. Alice has an input string \(x\) of size \(K\) and Bob needs to learn the string of Alice. Any algorithm (possibly randomized) for solving this problem requires at least \(\Omega(K)\) bits of communication, by a trivial information theoretic argument.

Notice that the problem of having Bob learn Alice’s input is not a binary function as addressed in Section 5.2. Similarly, computing weighted APSP is not a decision problem, but rather a problem whose solution assigns a value to each node (which is its vector of distances from all other nodes). We therefore use the extended Theorem 5.20 above.

**The fixed graph construction:** The fixed graph construction \(G = (V, E)\) is defined as follows. It contains a set of \(n - 2\) nodes, denoted \(A = \{a_0, \ldots, a_{n-3}\}\), which are all connected to an additional node \(a\). The node \(a\) is connected to the last node \(b\), by an edge of weight 0.

**Adding edge weights corresponding to the string \(x\):** Given the binary string \(x\) of size \(K = (n - 2)\log n\) we augment the graph \(G\) with edge weights, which defines \(G_x\), by having each non-overlapping batch of \(\log n\) bits encode a weight of an edge from \(A\) to \(a\). It is straightforward to see that \(G_x\) is a family of lower bound graphs for a function \(f\) where \(K_2 = L_1 = 0\), since the weights of the edges determine the right-hand side of the output (while the left-hand side is empty).

**Proof of Theorem 5.21** To prove Theorem 5.21 we let \(V_A = A \cup \{a\}\) and \(V_B = \{b\}\). Note that \(K = |x| = \Theta(n \log n)\). Furthermore, note that the only edge in the cut \(E(V_A, V_B)\) is the edge \((a, b)\). Since we showed that \(\{G_x\}\) is a family of lower bound graphs, we apply Theorem 5.20 on the above partition to deduce that because \(K\) bits are required to be communicated in order for Bob to know Alice’s \(K\)-bit input, any algorithm in the
CONGEST model for computing weighted APSP requires at least $\Omega(K/\log n) = \Omega(n)$ rounds.

5.5.2 The Alice-Bob Framework Cannot Give a Super-Linear Lower Bound for Weighted APSP

In this section we argue that a reduction from any 2-party function with a constant partition of the graph into Alice and Bob’s sides is provably incapable of providing a super-linear lower bound for computing weighted all pairs shortest paths in the CONGEST model. A more detailed inspection of our analysis shows a stronger claim: our claim also holds for algorithms for the CONGEST-BROADCAST model, where in each round each node must send the same $(\log n)$-bit message to all of its neighbors. The following theorem states our claim.

**Theorem 5.22.** Let $f : \{0,1\}^{K_1} \times \{0,1\}^{K_2} \to \{0,1\}^{L_1} \times \{0,1\}^{L_2}$ be a function and let $G_{x,y}$ be a family of lower bound graphs w.r.t. $f$ and the weighted APSP problem. When applying Theorem 5.20 to $f$ and $G_{x,y}$, the lower bound obtained for the number of rounds for computing weighted APSP is at most linear in $n$.

Roughly speaking, we show that given an input graph $G = (V,E)$ and a partition of the set of vertices into two sets $V = V_A \cup V_B$, such that the graph induced by the nodes in $V_A$ is simulated by Alice and the graph induced by nodes in $V_B$ is simulated by Bob, Alice and Bob can compute weighted all pairs shortest paths by communicating $O(n \log n)$ bits of information for each node touching the cut $C = (V_A, V_B)$ induced by the partition. This means that for any 2-party function $f$ and any family of lower bound graphs w.r.t. $f$ and weighted APSP according to the extended definition of Section 5.5.1, since Alice and Bob can compute weighted APSP which determines their output for $f$ by exchanging only $O(|V(C)|n \log n)$ bits, where $V(C)$ is the set of nodes touching $C$, the value $CC(f)$ is at most $O(|V(C)|n \log n)$. But then the lower bound obtained by Theorem 5.20 cannot be better than $\Omega(n)$, and hence no super-linear lower can be deduced by this framework as is.

Formally, given a graph $G = (V = V_A \cup V_B, E)$ we denote $C = E(V_A, V_B)$. Let $V(C)$ denote the nodes touching the cut $C$, with $C_A = V(C) \cap V_A$ and $C_B = V(C) \cap V_B$. Let $G_A = (V_A, E_A)$ be the subgraph induced by the nodes in $V_A$ and let $G_B = (V_B, E_B)$ be the subgraph induced by the nodes in $V_B$. For a graph $H$, we denote the weighted distance between two nodes $u, v$ by $wdist_H(u, v)$.

**Lemma 5.23.** Let $G = (V = V_A \cup V_B, E,w)$ be a graph with an edge-weight function $w : E \to \{1, \ldots, W\}$, such that $W \in \text{poly} \ n$. Suppose that $G_A$, $C_B$, $C$ and the values of $w$ on $E_A$ and $C$ are given as input to Alice, and that $G_B$, $C_A$, $C$ and the values of $w$ on $E_B$ and $C$ are given as input to Bob.
Then, Alice can compute the distances in $G$ from all nodes in $V_A$ to all nodes in $V$ and Bob can compute the distances from all nodes in $V_B$ to all the nodes in $V$, using $O(|V(C)| \cdot n \log n)$ bits of communication.

Proof. We describe a protocol for the required computation, as follows. For each node $u \in C_B$, Bob sends to Alice the weighted distances in $G_B$ from $u$ to all nodes in $V_B$, that is, Bob sends $\{\text{wdist}_{G_B}(u, v) \mid u \in C_B, v \in V_B\}$ (or $\infty$ for pairs of nodes not connected in $G_B$).

Alice constructs a virtual graph $G'_A = (V'_A, E'_A, w'_A)$ with the nodes $V'_A = V_A \cup C_B$ and edges $E'_A = E_A \cup C \cup (C_B \times C_B)$. The edge-weight function $w'_A$ is defined by $w'_A(e) = w(e)$ for each $e \in E_A \cup C$, and $w'_A(u, v)$ for $u, v \in C_B$ is defined to be the weighted distance between $u$ and $v$ in $G_B$, as received from Bob. Alice then computes the set of all weighted distances in $G'_A$, $\{\text{wdist}_{G'_A}(u, v) \mid u, v \in V'_A\}$.

Alice assigns her output for the weighted distances in $G$ as follows. For two nodes $u, v \in V_A \cup C_B$, Alice outputs their weighted distance in $G'_A$, $\text{wdist}_{G'_A}(u, v)$. For a node $u \in V'_A$ and a node $v \in V_B \setminus C_B$, Alice outputs $\min\{\text{wdist}_{G'_A}(u, x) + \text{wdist}_{G_B}(x, v) \mid x \in C_B\}$, where $\text{wdist}_{G'_A}$ is the distance in $G'_A$ as computed by Alice, and $\text{wdist}_{G_B}$ is the distance in $G_B$ that was sent by Bob.

For Bob to compute his required weighted distances, for each node $u \in C_A$, similar information is sent by Alice to Bob, that is, Alice sends to Bob the weighted distances in $G_A$ from $u$ to all nodes in $V_A$. Bob constructs the analogous graph $G'_B$ and outputs his required distance. The next paragraph formalizes this for completeness, but may be skipped by a convinced reader.

Formally, Alice sends $\{\text{wdist}_{G'_A}(u, v) \mid u \in C_A, v \in V_A\}$. Bob constructs $G'_B = (V'_B, E'_B, w'_B)$ with $V'_B = V_B \cup C_A$ and edges $E'_B = E_B \cup C \cup (C_A \times C_A)$. The edge-weight function $w'_B$ is defined by $w'_B(e) = w(e)$ for each $e \in E_B \cup C$, and $w'_B(u, v)$ for $u, v \in C_A$ is defined to be the weighted distance between $u$ and $v$ in $G_A$, as received from Alice (or $\infty$ if they are not connected in $G_A$). Bob then computes the set of all weighted distances in $G'_B$, $\{\text{wdist}_{G'_B}(u, v) \mid u, v \in V'_B\}$. Bob assigns his output for the weighted distances in $G$ as follows. For two nodes $u, v \in V_B \cup C_A$, Bob outputs their weighted distance in $G'_B$, $\text{wdist}_{G'_B}(u, v)$. For a node $u \in V'_B$ and a node $v \in V_A \setminus C_A$, Bob outputs $\min\{\text{wdist}_{G'_B}(u, x) + \text{wdist}_{G_A}(x, v) \mid x \in C_A\}$, where $\text{wdist}_{G'_B}$ is the distance in $G'_B$ as computed by Bob, and $\text{wdist}_{G_A}$ is the distance in $G_A$ that was sent by Alice.

Complexity. Bob sends to Alice the distances from all nodes in $C_B$ to all node in $V_B$, which takes $O(|C_B| \cdot |V_B| \log n)$ bits, and similarly Alice sends $O(|C_A| \cdot |V_A| \log n)$ bits to Bob, for a total of $O(|V(C)| \cdot n \log n)$ bits.

Correctness. By construction, for every edge $(u, v) \in C_B \times C_B$ in $G'_A$ with weight $\text{wdist}_{G'_A}(u, v)$, there is a corresponding shortest path $P_{u,v}$ of the same weight in $G_B$. Hence, for any path $P' = (v_0, v_1, \ldots, v_k)$ in $G'_A$ between $v_0, v_k \in V'_A$, there is a corresponding path $P_{v_0,v_k}$ of the same weight in $G$, where $P$ is obtained from $P'$ by replacing every
two consecutive nodes \( v_i, v_{i+1} \) in \( P \cap C_B \) by the path \( P_{v_i,v_{i+1}} \). Thus, \( \text{wdist}_G(v_0, v_k) \geq \text{wdist}_G(v_0, v_k) \).

On the other hand, for any shortest path \( P = (v_0, v_1, \ldots, v_k) \) in \( G \) connecting \( v_0, v_k \in V'_A \), there is a corresponding path \( P' \) of the same weight in \( G'_A \), where \( P' \) is obtained from \( P \) by replacing any sub-path \((v_i, \ldots, v_j)\) of \( P \) contained in \( G_B \) and connecting \( v_i, v_j \in C_B \) by the edge \((v_i, v_j)\) in \( G'_A \). Thus, \( \text{wdist}_G(v_0, v_k) \geq \text{wdist}_{G'_A}(v_0, v_k) \). Alice thus correctly computes the weighted distances between pairs of nodes in \( V'_A \).

It remains to argue about the weighted distances that Alice computes to nodes in \( V_B \setminus C_B \). Any lightest path \( P \) in \( G \) connecting a node \( u \in V'_A \) and a node \( v \in V_B \setminus C_B \) must cross at least one edge of \( C \) and thus must contain a node in \( C_B \). Therefore, \( \text{wdist}_G(u, v) = \min\{\text{wdist}_G(u, x) + \text{wdist}_G(x, v) \mid x \in C_B\} \). Recall that we have shown that \( \text{wdist}_{G'_A}(u, x) = \text{wdist}_G(u, x) \) for any \( u, x \in V'_A \). The sub-path of \( P \) connecting \( x \) and \( v \) is a shortest path between these nodes, and is contained in \( G_B \), so \( \text{wdist}_{G_B}(x, v) = \text{wdist}_G(x, v) \). Hence, the distance \( \min\{\text{wdist}_{G'_A}(u, x) + \text{wdist}_{G_B}(x, v) \mid x \in C_B\} \) returned by Alice is indeed equal to \( \text{wdist}_G(u, v) \).

The outputs of Bob are correct by the analogous arguments, completing the proof. □

**Proof of Theorem 5.22** Let \( f : \{0,1\}^{K_1} \times \{0,1\}^{K_2} \rightarrow \{0,1\}^{L_1} \times \{0,1\}^{L_2} \) be a function and let \( G_{x,y} \) be a family of lower bound graphs w.r.t. \( f \) and the weighted APSP problem. By Lemma 5.23, Alice and Bob can compute the weighted distances for any graph in \( G_{x,y} \) by exchanging at most \( O(|V(C)|n \log n) \) bits, which is at most \( O(|C|n \log n) \) bits. Since \( G_{x,y} \) is a family of lower bound graphs w.r.t. \( f \) and weighted APSP, condition (4) gives that this number of bits is an upper bound for \( CC(f) \). Therefore, when applying Theorem 5.20 to \( f \) and \( G_{x,y} \), the lower bound obtained for the number of rounds for computing weighted APSP is \( \Omega(CC(f)/|C| \log n) \), which is no higher than a bound of \( \Omega(n) \). □

**Extending to \( t \) players:** We argue that generalizing the Alice-Bob framework to a shared-blackboard multi-party setting is still insufficient for providing a super-linear lower bound for weighted APSP. Suppose that we increase the number of players in the above framework to \( t \) players, \( P_0, \ldots, P_{t-1} \), each simulating the nodes in a set \( V_i \) in a partition of \( V \) in a family of lower bound graphs w.r.t. a \( t \)-party function \( f \) and weighted APSP. That is, the outputs of nodes in \( V_i \) for an algorithm \( ALG \) for solving a problem \( P \) in the CONGEST model, uniquely determines the output of player \( P_i \) in the function \( f \). The function \( f \) is now a function from \( \{0,1\}^{K_0} \times \cdots \times \{0,1\}^{K_{t-1}} \) to \( \{0,1\}^{L_0} \times \cdots \times \{0,1\}^{L_{t-1}} \).

The communication complexity \( CC(f) \) is the total number of bits written on the shared blackboard by all players. Denote by \( C \) the set of cut edges, that is, the edge whose endpoints do not belong to the same \( V_i \). Then, if \( ALG \) is an \( R \)-rounds algorithm, we have that writing \( O(R|C| \log n) \) bits on the shared blackboard suffice for computing \( f \), and so \( R = \Omega(CC(f)/|C| \log n) \).
We now consider the problem $P$ to be weighted APSP. Let $f$ be a $t$-party function and let $G_{x_0,\ldots,x_{t-1}}$ be a family of lower bound graphs w.r.t. $f$ and weighted APSP. We first have the players write all the edges in $C$ on the shared blackboard, for a total of $O(|C|\log n)$ bits. Then, in turn, each player $P_i$ writes the weighted distances from all nodes in $V_i$ to all nodes in $V(C) \cap V_i$. This requires no more than $O(|V(C)|n \log n)$ bits.

It is easy to verify that every player $P_i$ can now compute the weighted distances from all nodes in $V_i$ to all nodes in $V$, in a manner that is similar to that of Lemma 5.23.

This gives an upper bound on $CC(f)$, which implies that any lower bound obtained by a reduction from $f$ is $\Omega(CC(f)/|C|\log n)$, which is no larger than $\Omega((|V(C)|n + |C|)\log n/|C|\log n)$, which is no larger than $\Omega(n)$, since $|V(C)| \leq 2|C|$.

**Remark 1:** Notice that the $t$-party simulation of the algorithm for the CONGEST model does not require a shared blackboard and can be done in the peer-to-peer multiparty setting as well, since simulating the delivery of a message does not require the message to be known globally. This raises the question of why would one consider a reduction to the CONGEST model from the stronger shared-blackboard model to begin with. Notice that our argument above for $t$ players does not translate to the peer-to-peer multiparty setting, because it assumes that the edges of the cut $C$ can be made global knowledge within writing $|C|\log n$ bits on the blackboard. However, what our extension above shows is that if there is a lower bound that is to be obtained using a reduction from peer-to-peer $t$-party computation, it must use a function $f$ that is strictly harder to compute in the peer-to-peer setting compared with the shared-blackboard setting.

**Remark 2:** We suspect that a similar argument can be applied for the framework of non-fixed Alice-Bob partitions (e.g., [DSHK+12]), but this requires precisely defining this framework which is not addressed in this version.

### 5.6 Discussion

This work provides the first super-linear lower bounds for the CONGEST model, raising a plethora of open questions. First, we showed for some specific problems, namely, computing a minimum vertex cover, a maximum independent set and a $\chi$-coloring, that they are nearly as hard as possible for the CONGEST model. However, we know that approximate solutions for some of these problems can be obtained much faster, in a polylogarithmic number of rounds or even less. A family of specific open questions is then to characterize the exact trade-off between approximation factors and round complexities for these problems.

Another specific open question is the complexity of weighted APSP, which has also been asked in previous work [Elk04a, Nan14]. Our proof that the Alice-Bob framework is incapable of providing super-linear lower bounds for this problem may be viewed as
providing evidence that weighted APSP can be solved much faster than is currently known. Together with the recent sub-quadratic algorithm of [Elk17], this brings another angle to the question: can weighted APSP be solved in linear time?

Finally, we propose a more general open question which addresses a possible classification of complexities of global problems in the CONGEST model. Some such problems have complexities of $\Theta(D)$, such as constructing a BFS tree. Others have complexities of $\tilde{\Theta}(D + \sqrt{n})$, such as finding an MST. Some problems have near-linear complexities, such as unweighted APSP. And now we know about the family of hardest problems for the CONGEST model, whose complexities are near-quadratic. Do these complexities capture all possibilities, when natural global graph problems are concerned? Or are there such problems with a complexity of, say, $\Theta(n^{1+\delta})$, for some constant $0 < \delta < 1$? A similar question was recently addressed in [CP17] for the LOCAL model, and we propose investigating the possibility that such a hierarchy exists for the CONGEST model.
Chapter 6

Algebraic Methods in the Congested Clique

6.1 Introduction

In this chapter, we use algebraic methods for solving distance computation and subgraph detection tasks in the CLIQUE model. Specifically, we adapt parallel matrix multiplication implementations to the CLIQUE, obtaining an $O(n^{1-2/\omega})$ round matrix multiplication algorithm, where $\omega < 2.3728639$ is the exponent of sequential matrix multiplication. In conjunction with known techniques from centralized algorithms, this gives significant improvements over previous best upper bounds in the CLIQUE model.

Our main results include triangle and 4-cycle counting algorithms, both running in $O(n^{0.158})$ rounds, improving upon the $O(n^{1/3})$ algorithm of Dolev et al. [DLP12]. We also present a $(1 + o(1))$-approximation of all-pairs shortest paths with the same running time of $O(n^{0.158})$ rounds, improving upon the $\tilde{O}(n^{1/2})$-round $(2 + o(1))$-approximation algorithm given by Nanongkai [Nan14]. Moreover, we present an algorithm for computing the girth in $O(n^{0.158})$ rounds, which is the first non-trivial solution in this model. In addition, we present a novel constant-round combinatorial algorithm for detecting 4-cycles. Table 6.1 contains a summary of our results, which we overview in more details in what follows.

Our result regarding the computation of matrix multiplication in the CLIQUE model is stated in the next theorem.

**Theorem 6.1.** The product of two $n \times n$ matrices can be computed in a CLIQUE of $n$ nodes in $O(n^{1/3})$ rounds over semirings. Over rings, this product can be computed in $O(n^{1-2/\omega+\varepsilon})$ rounds for any constant $\varepsilon > 0$.

**Distributed matrix multiplication exponent.** Analogously with the matrix multiplication exponent, we denote by $\rho$ the exponent of matrix multiplication in the congested clique model, that is, the infimum over all values $\sigma$ such that there exists a matrix.
<table>
<thead>
<tr>
<th>Problem</th>
<th>This work</th>
<th>Prior work</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix multiplication (semiring)</td>
<td>$O(n^{1/3})$</td>
<td>—</td>
</tr>
<tr>
<td>matrix multiplication (ring)</td>
<td>$O(n^{0.158})$</td>
<td>$O(n^{0.373})$ [DKO14]</td>
</tr>
<tr>
<td>triangle counting</td>
<td>$O(n^{0.158})$</td>
<td>$O(n^{1/3}/\log n)$ [DLP12]</td>
</tr>
<tr>
<td>4-cycle detection</td>
<td>$O(1)$</td>
<td>$O(n^{1/2}/\log n)$ [DLP12]</td>
</tr>
<tr>
<td>4-cycle counting</td>
<td>$O(n^{0.158})$</td>
<td>$O(n^{1/2}/\log n)$ [DLP12]</td>
</tr>
<tr>
<td>$k$-cycle detection</td>
<td>$2^{O(k)n^{0.158}}$</td>
<td>$O(n^{1-2/k}/\log n)$ [DLP12]</td>
</tr>
<tr>
<td>girth</td>
<td>$O(n^{0.158})$</td>
<td>—</td>
</tr>
<tr>
<td>all-pairs shortest paths</td>
<td>$O(n^{1/3}\log n)$</td>
<td>—</td>
</tr>
<tr>
<td>· weighted diameter $U$</td>
<td>$O(Un^{0.158})$</td>
<td>—</td>
</tr>
<tr>
<td>· $(1+o(1))$-approximation</td>
<td>$O(n^{0.158})$</td>
<td>—</td>
</tr>
<tr>
<td>· undirected, $(2+o(1))$-approximation</td>
<td>$\tilde{O}(n^{1/2})$ [Nan14]</td>
<td>—</td>
</tr>
<tr>
<td>· undirected, unweighted</td>
<td>$O(n^{0.158})$</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 6.1: Our results versus prior work, for the currently best known bound $\omega < 2.3729$ [LG14]

multiplication algorithm in the congested clique running in $O(n^\sigma)$ rounds. In this notation, Theorem 6.1 gives us

$$\rho \leq 1 - 2/\omega < 0.15715;$$

prior to this work, it was known that $\rho \leq \omega - 2$ [DKO14].

For the rest of this thesis, we will – analogously with the convention in centralized algorithms – slightly abuse this notation by writing $n^\rho$ for the complexity of matrix multiplication in the congested clique. This hides factors up to $O(n^\varepsilon)$ resulting from the fact that the exponent $\rho$ is defined as infimum of an infinite set.

The above addresses matrices whose entries can be encoded with $O(\log n)$ bits, which is sufficient for dealing with integers of absolute value at most $n^{O(1)}$. In general, if $b$ bits are sufficient to encode matrix entries, the bounds above hold with a multiplicative factor of $[b/\log n]$; for example, working with integers with absolute value at most $2^{n^\varepsilon}$ merely incurs a factor $n^\varepsilon$ overhead in running times.

### 6.1.1 Applications in Subgraph Detection

**Cycle detection and counting.** Our first application of fast matrix multiplication is to the problems of triangle counting [IR78] and 4-cycle counting.

**Corollary 6.2.** For directed and undirected graphs, the number of triangles and 4-cycles can be computed in $O(n^{\rho})$ rounds.
For $\rho \leq 1 - 2/\omega$, this is an improvement upon the previously best known $O(n^{1/3})$-round triangle detection algorithm of Dolev et al. [DLP12] and an $O(n^{\omega-2+\varepsilon})$-round algorithm of Drucker et al. [DKO14]. Indeed, we disprove the conjecture of Dolev et al. that any deterministic oblivious algorithm for detecting triangles requires $\tilde{\Omega}(n^{1/3})$ rounds.

When only detection of cycles is required, we observe that combining the fast distributed matrix multiplication with the well-known technique of color-coding [AYZ95] allows us to detect $k$-cycles in $\tilde{O}(n^\rho)$ rounds for any constant $k$. This improves upon the subgraph detection algorithm of Dolev et al. [DLP12], which requires $\tilde{O}(n^{1 - 2/k})$ rounds for detecting subgraphs of $k$ nodes. However, we do not improve upon the algorithm of Dolev et al. for general subgraph detection.

**Theorem 6.3.** For directed and undirected graphs, the existence of $k$-cycles can be detected in $2^{O(k)}n^\rho \log n$ rounds.

For the specific case of $k = 4$, we provide a novel algorithm that does not use matrix multiplication and detects 4-cycles in $O(1)$ rounds on undirected graphs.

**Theorem 6.4.** For undirected graphs, the existence of 4-cycles can be detected in $O(1)$ rounds.

**Girth.** The cycle detection methods can also be adapted to find the length of the shortest cycle, that is, the girth of a graph. For undirected graphs, we leverage a known trade-off between the girth and the number of edges of the graph [Mat02]. Roughly, we detect short cycles fast, and if they do not exist then the graph must have sufficiently few edges to be learned by all nodes. For directed graphs, we adapt a simpler method of Itai and Rodeh [IR78].

**Theorem 6.5.** For directed and undirected unweighted graphs, the girth can be computed in $\tilde{O}(n^\rho)$ rounds.

As far as we are aware, these are the first algorithms to compute the girth in this setting.

### 6.1.2 Applications in Distance Computation

**Shortest paths.** The all-pairs shortest paths problem (APSP) likewise admits algorithms based on matrix multiplication. The basic idea is to compute the $n^{th}$ power of the input graph’s weight matrix over the min-plus semiring, by iteratively computing squares of the matrix [Fur70, Mun71, FM71].

**Corollary 6.6.** For directed graphs with edge weights in $\{0, \pm 1, \ldots, \pm M\}$ and for undirected graphs with edge weights in $\{0, 1, \ldots, M\}$, all-pairs shortest paths can be computed in $O(n^{1/3}\log n[\log M/\log n])$ communication rounds.
We can leverage fast ring matrix multiplication to improve upon the above result; however, the use of ring matrix multiplication necessitates some trade-offs or extra assumptions. For example, for unweighted and undirected graphs, it is possible to recover the exact shortest paths from powers of the adjacency matrix over the Boolean semiring \[\text{Sei95}\].

**Corollary 6.7.** For undirected, unweighted graphs, all-pairs shortest paths can be computed in $\tilde{O}(n^\rho)$ rounds.

For small integer weights, we use the well-known idea of embedding a min-plus semiring matrix product into a matrix product over a ring; this gives a multiplicative factor to the running time proportional to the maximum distance between two nodes, that is, the weighted diameter.

**Corollary 6.8.** For directed and undirected graphs with positive integer edge weights and weighted diameter $U$, all-pairs shortest paths can be computed in $\tilde{O}(U n^\rho)$ rounds.

While this corollary is only relevant for graphs of small weighted diameter, the same idea can be combined with weight rounding \[\text{RT85}, \text{Zwi02}, \text{Nan14}\] to obtain a fast approximate APSP algorithm without such limitations.

**Theorem 6.9.** For directed and undirected graphs with edge weights in $\{0, 1, \ldots, 2^{n^{o(1)}}\}$, all-pairs shortest paths can be $(1 + o(1))$-approximated in $O(n^{\rho + o(1)})$ rounds.

For comparison, the previously best known combinatorial algorithm for APSP on the congested clique gives a $(2 + o(1))$-approximation in $\tilde{O}(n^{1/2})$ rounds on undirected graphs with positive weights \[\text{Nan14}\].

### 6.1.3 Additional Related Work

Computing distances in graphs, such as the diameter, all-pairs shortest paths (APSP), and single-source shortest paths (SSSP) are fundamental problems in most computational settings. The reason for this lies in the abundance of applications of such computations, evident also by the huge amount of research dedicated to it \[\text{Cha10, HT12, Han08, Tak04, Zwi06, Tak05, Cha08, Fre76, Wil14, Zwi01, Zwi02}\].

In particular, computing graph distances is vital for many distributed applications and, as such, has been widely studied in the CONGEST model of computation \[\text{Pel00}\], where $n$ processors located in $n$ distinct nodes of a graph $G$ communicate over the graph edges using $O(\log n)$-bit messages. Specifically, many algorithms and lower bounds were given for computing and approximating graph distances in this setting \[\text{DSHK}^{+12}, \text{Nan14}, \text{LP13b}, \text{HW12}, \text{PRT12}, \text{FHW12}, \text{LP13a}, \text{HPRW14}, \text{KP98}, \text{PR00}\]. Some lower bounds apply even for graphs of small diameter; however, these lower bound constructions boil down to graphs that contain \textit{bottleneck} edges limiting the amount of information that can be exchanged between different parts of the graph quickly.
The intuition that the congested clique model would abstract away distances and bottlenecks and bring to light only the congestion challenge has proven inaccurate. Indeed, a number of tasks have been shown to admit sub-logarithmic or even constant-round solutions, exceeding by far what is possible in the CONGEST model with only low diameter. The pioneering work of Lotker et al. [LPPP05] shows that a minimum spanning tree (MST) can be computed in $O(\log \log n)$ rounds, which was recently improved to $O(\log \log \log n)$ by Hegeman et al. [HPP+15]. Hegeman et al. [HPS14] show how to construct a 3-ruling set, with applications to maximal independent set and an approximation of the MST in certain families of graphs; sorting and routing have been recently addressed by various authors [Len13, LWT11, PTT11]. A connection between the congested clique model and the MapReduce model is discussed by Hegeman and Pemmaraju [HP14], where algorithms are given for coloring problems. On top of these positive results, Drucker et al. [DKO14] recently proved that essentially any non-trivial unconditional lower bound for an explicit function on the congested clique would imply novel circuit complexity lower bounds.

The same work also points out the connection between fast matrix multiplication algorithms and triangle detection in the congested clique. Their construction yields an $O(n^{\omega-2+\varepsilon})$ round algorithm for matrix multiplication over rings in the congested clique model, giving also the same running bound for triangle detection; if $\omega = 2$, this gives $\rho = 0$, matching our result. However, with the currently best known centralized matrix multiplication algorithm, the running time of the resulting triangle detection algorithm is $O(n^{0.3729})$ rounds, still slower than the combinatorial triangle detection of Dolev et al. [DLP12], and if $\omega > 2$, the solution presented in this thesis is faster.

### 6.2 Matrix Multiplication Algorithms

In this section, we consider computing the product $P = ST$ of two $n \times n$ matrices $S = (S_{ij})$ and $T = (T_{ij})$ on the congested clique with $n$ nodes. For convenience, we tacitly assume that nodes $v \in V$ are identified with \{1, 2, ..., $n$\}, and use nodes $v \in V$ to directly index the matrices. The local input in the matrix multiplication task for each node $v \in V$ is the row $v$ of both $S$ and $T$, and at the end of the computation each node $v \in V$ will output the row $v$ of $P$. However, we note that the exact distribution of the input and output is not important, as we can re-arrange the entries in constant rounds as long as each node has $O(n)$ entries [Len13]. Furthermore, we assume that the matrix entries and any intermediate result can be transmitted in a single message of size $O(\log n)$ bits; for example, this holds when $S$ and $T$ consist of integers of absolute value at most $O(1)$.

**Theorem 6.1.** The product of two $n \times n$ matrices can be computed in a CLIQUE of $n$ nodes in $O(n^{1/3})$ rounds over semirings. Over rings, this product can be computed in $O(n^{1-2/\omega+\varepsilon})$ rounds for any constant $\varepsilon > 0$.

**Theorem 6.1** follows directly by simulating known parallel matrix multiplication
algorithms in the congested clique model using a result of Lenzen [Len13]. Lenzen discusses simulation of the bulk-synchronous parallel (BSP) model, which we can use to obtain Theorem 6.1 as a corollary from known BSP matrix multiplication results [McC95, McC96, Tis99]. However, essentially the same matrix multiplication algorithms have been widely studied in various parallel computation models, and the routing scheme underlying the aforementioned simulation result allows also simulation of these other models on the congested clique:

1. The first part of [Theorem 6.1] is based on the so-called parallel 3D matrix multiplication algorithm [ABG+95, McC95], essentially a parallel implementation of the school-book matrix multiplication; alternatively, the same algorithm can be obtained by slightly modifying the triangle counting algorithm of Dolev et al. [DLP12].

2. The second part uses a scheme that allows one to adapt any bilinear matrix multiplication algorithm into a fast parallel matrix multiplication algorithm [LD95, McC96, Tis99, BDH+12].

A more detailed examination in fact shows that the matrix multiplication algorithms are oblivious, that is, the communication pattern is pre-defined and only the content of the messages depends on the input. This further allows us to use the static routing scheme of Dolev et al. [DLPT2], resulting in simpler algorithms with smaller constant factors in the running time.

The obliviousness of the algorithm also means that the messages consist solely of encodings of matrix entries and intermediate results. Thus, the matrix multiplication algorithms can be used in settings where either the edge bandwidth or the encoding size of the values is not necessarily $\Theta(\log n)$. Specifically, given $B$ bits of bandwidth per edge per round and $b$ bits required for the encoding of the values, bounds of Theorem 6.1 hold with multiplicative factor of $[b/B]$, as we can transmit a single $b$-bit value sequentially in $[b/B]$ messages.

To account for all the details, and to provide an easy access for readers not familiar with the parallel computing literature, we present the congested clique versions of these algorithms in full detail in Sections 6.2.1 and 6.2.2.

### 6.2.1 Semiring matrix multiplication

**Preliminaries.** For convenience, let us assume that the number of nodes is such that $n^{1/3}$ is an integer. We view each node $v \in V$ as a three-tuple $v_1v_2v_3$ where $v_1, v_2, v_3 \in [n^{1/3}]$; for concreteness, we may think that $v_1v_2v_3$ is the representation of $v$ as a three-digit number in base $n^{1/3}$.

For a matrix $S$ and index sets $U, W \subseteq V$, use the notation $S[U, W]$ to refer to the submatrix obtained by taking all rows $u$ with $u \in U$ and columns $w$ with $w \in W$. To easily refer to specific subsets of indices, we use * as a wild-card in this notation; specifically, we
use notation \( x^{**} = \{ v : v_1 = x \} \), \( *x = \{ v : v_2 = x \} \) and \( **x = \{ v : v_3 = x \} \). Finally, in conjunction with this notation, we use the shorthand \(*\) to denote the whole index set \( V \) and \( v \) to refer to a singleton set \( \{ v \} \). See Figure 6.1.

**Overview.** The distributed implementation of the school-book matrix multiplication we present is known as the 3D algorithm. To illustrate why, we note that the school-book matrix multiplication involves \( n^3 \) element-wise multiplications of the form

\[
S_{uv}T_{vw}, \quad u, v, w \in V,
\]

which can be viewed as points in the cube \( V \times V \times V \). To split the element-wise multiplications equally among the nodes, we partition this cube into \( n \) subcubes of size \( n^{2/3} \times n^{2/3} \times n^{2/3} \). Specifically, each node \( v \) is assigned the subcube \( v^{**} \times \times v^{**} \times v^{**} \), corresponding to the multiplication task

\[
S[v^{**}, v^{**}]T[v^{**}, v^{**}] .
\]

**Algorithm description.** The algorithm computes \( n \times n \) intermediate matrices \( P^{(w)} = S[*w][w**, \times] \) for \( w \in [n^{1/3}] \), so that each node \( v \) computes the block

\[
P^{(v_2)}[v^{**}, v^{**}] = S[v^{**}, v^{**}]T[v^{**}, v^{**}] .
\]

Specifically, this is done as follows.

**Step 1: Distributing the entries.** Each \( v \in V \) sends, for each node \( u \in v^{**} \), the submatrix \( S[v, u^{**}] \) to node \( u \), and for each node \( w \in *v^{**} \), the submatrix \( T[v, w^{**}] \) to \( w \). Each such submatrix has size \( n^{2/3} \) and there are \( 2n^{2/3} \) recipients, for a total of \( 2n^{4/3} \) messages per node.

Dually, each node \( v \in V \) receives the submatrix \( S[v^{**}, v^{**}] \) and the submatrix \( T[v^{**}, v^{**}] \). In particular, the submatrix \( S[u, v^{**}] \) is received from the node \( u \) for
\[ u \in v_1^{**}, \text{ and similarly the submatrix } T[w, v_3^{**}] \text{ is received from the node } w \in v_2^{**}. \]

In total, each node receives \( 2n^{4/3} \) messages.

**Step 2: Multiplication.** Each node \( v \in V \) computes the product of \( S[v_1^{**}, v_2^{**}] \) and \( T[v_2^{**}, v_3^{**}] \) to get the \( n^{2/3} \times n^{2/3} \) product matrix \( P^{(v_2)}[v_1^{**}, v_2^{**}] \).

**Step 3: Distributing the products.** Each node \( v \in V \) sends submatrix \( P^{(v_2)}[u, v_3^{**}] \) to each node \( u \in v_1^{**} \). Each such submatrix has size \( n^{2/3} \times n^{2/3} \) and there are \( n^{2/3} \) recipients, for a total of \( n^{4/3} \) messages per node.

Dually, each node \( v \in V \) receives the submatrices \( P^{(w)}[v, \ast] \) for each \( w \in [n^{1/3}] \). In particular, the submatrix \( P^{(w)}[v, u_3^{**}] \) is received from the node \( u \in v_1^{**} \). The total number of received messages is \( n^{4/3} \) per node.

**Step 4: Assembling the product.** Each node \( v \in V \) computes the submatrix \( P[v, \ast] = \sum_{w \in [n^{1/3}]} P^{(w)}[v, \ast] \) of the product \( P = ST \).

**Analysis.** The maximal number of messages sent or received in one of the above steps is \( O(n^{4/3}) \). Moreover, the communication pattern clearly does not depend on the input matrices, so the algorithm can be implemented in an oblivious way on the congested clique using the routing scheme of Dolev et al. [DLP12, Lemma 1]; the running time is \( O(n^{1/3}) \) rounds.

### 6.2.2 Fast Matrix Multiplication

**Bilinear matrix multiplication.** Consider a bilinear algorithm multiplying two \( d \times d \) matrices using \( m < d^3 \) scalar multiplications, such as the Strassen algorithm [Str69]. Such an algorithm computes the matrix product \( P = ST \) by first computing \( m \) linear combinations of entries of both matrices,

\[
\hat{S}^{(w)} = \sum_{(i,j) \in [d]^2} \alpha_{ijw} S_{ij}, \quad \text{and} \\
\hat{T}^{(w)} = \sum_{(i,j) \in [d]^2} \beta_{ijw} T_{ij} \quad \text{(6.1)}
\]

for each \( w \in [m] \), then computing the products \( \hat{P}^{(w)} = \hat{S}^{(w)} \hat{T}^{(w)} \) for \( w \in [m] \), and finally obtaining \( P \) as

\[
P_{ij} = \sum_{w \in [m]} \lambda_{ijw} \hat{P}^{(w)}, \quad \text{for } (i, j) \in [d]^2, \quad \text{(6.2)}
\]

where \( \alpha_{ijw}, \beta_{ijw} \) and \( \lambda_{ijw} \) are scalar constants that define the algorithm, and the other elements are scalars depending on the input matrices. In this section we show that any bilinear matrix multiplication algorithm can be efficiently translated to the congested clique model.
Lemma 6.10. Let \( R \) be a ring, and assume there exists a family of bilinear matrix multiplication algorithms that can compute product of \( n \times n \) matrices with \( O(n^\omega) \) multiplications. Then matrix multiplication over \( R \) can be computed in the congested clique in \( O(n^{1-2/\omega}[b/\log n]) \) rounds, where \( b \) is the number of bits required for encoding a single element of \( R \).

In particular for integers, rationals and their extensions, it is known that for any constant \( \varepsilon > 0 \) there is a bilinear algorithm for matrix multiplication that uses \( O(n^{\omega+\varepsilon}) \) multiplications \cite{BCS97}; thus, the second part of Lemma 6.1 follows from the above lemma.

Preliminaries. Let us fix a bilinear algorithm that computes the product of \( d \times d \) matrices using \( m(d) = O(d^\sigma) \) scalar multiplications for any \( d \), where \( 2 \leq \sigma \leq 3 \). To multiply two \( n \times n \) matrices on a congested clique of \( n \) nodes, fix \( d \) so that \( m(d) = n \), assuming for convenience that \( n \) is such that this is possible. Note that we have \( d = O(n^{1/\sigma}) \).

Similarly with the semiring matrix multiplication, we view each node \( v \) as three-tuple \( v_1v_2v_3 \), where we assume that \( v_1 \in [d] \), \( v_2 \in [n^{1/2}] \), \( v_3 \in [n^{1/2}/d] \); one can draw an analogy with the semiring matrix multiplication algorithm, where a similar three-tuple presentation was considered as a base \( n^{1/3} \) representation of \( v \), and view \( v_1v_2v_3 \) as a mixed-radix representation of the integer \( v \). This induces a partitioning of the input matrices \( S \) and \( T \) into a two-level grid of submatrices; using the same wild-card notation as before, \( S \) is partitioned into a \( d \times d \) grid of \( n/d \times n/d \) submatrices \( S[i**,j**] \) for \( (i,j) \in [d]^2 \), and each of these submatrices is further partitioned into an \( n^{1/2} \times n^{1/2} \) grid of \( n^{1/2}/d \times n^{1/2}/d \) submatrices \( S[ix*,jy*] \) for \( x,y \in [n^{1/2}] \). The other input matrix \( T \) is partitioned similarly; see Figure 6.2.

Finally, we give each node \( v \in V \) a unique secondary label \( \ell(v) = x_1x_2 \in [n^{1/2}]^2 \); again, for concreteness we assume that \( x_1x_2 \) is the representation of \( v \) in base-\( n^{1/2} \) system, so this label can be computed from \( v \) directly.

Overview. The basic idea of the fast distributed matrix multiplication is that we view the matrices \( S \) and \( T \) as \( d \times d \) matrices \( S' \) and \( T' \) over the ring of \( n/d \times n/d \) matrices, where

\[
S'_{ij} = S[i**,j**], \quad T'_{ij} = T[i**,j**]
\]

for \( i,j \in [d] \), which allows us to use (6.1) and (6.2) to compute the matrix product using the fixed bilinear algorithm; specifically, this reduces the \( n \times n \) matrix product into \( n \) instances of \( n^{1-1/\sigma} \times n^{1-1/\sigma} \) matrix products, each of which is given to a different node. For the linear combination steps, we use a partitioning scheme where each node \( v \) with secondary label \( \ell(v) = x_1x_2 \) is responsible for a specific \( n^{1/2}/d \times n^{1/2}/d \) submatrix of the matrices involved in the computation.
Algorithm description. The algorithm computes the matrix product $P = ST$ as follows.

Step 1: Distributing the entries. Each $v \in V$ sends, for $x_2 \in [n^{1/2}]$, the two submatrices $S[v, \ast x_2 \ast]$ and $T[v, \ast x_2 \ast]$ to the node $u$ with label $\ell(u) = v_2 x_2$. Each submatrix has $n^{1/2}$ entries and there are $n^{1/2}$ recipients each receiving two submatrices, for a total of $2n$ messages per node.

Dually, each node $u$ with label $\ell(u) = x_1 x_2$ receives the submatrices $S[v, \ast x_2 \ast]$ and $T[v, \ast x_2 \ast]$ from the nodes $v = v_1 v_2 v_3$ with $v_2 = x_1$. In particular, node $u$ now has the submatrices $S[\ast x_1 \ast, \ast x_2 \ast]$ and $T[\ast x_1 \ast, \ast x_2 \ast]$. The total number of received messages is $2n$ per node.

Step 2: Linear combination of entries. Each node $v \in V$ with label $\ell(v) = x_1 x_2$ computes for $w \in V$ the $n^{1/2}/d \times n^{1/2}/d$ matrices

$$
\hat{S}^{(w)}[x_1 \ast, x_2 \ast] = \sum_{(i,j) \in [d]^2} \alpha_{ijw} S[i x_1 \ast, j x_2 \ast],
$$

$$
\hat{T}^{(w)}[x_1 \ast, x_2 \ast] = \sum_{(i,j) \in [d]^2} \beta_{ijw} T[i x_1 \ast, j x_2 \ast].
$$
The computation is performed entirely locally.

**Step 3: Distributing the combinations.** Each node \( v \in V \) with label \( \ell(v) = x_1x_2 \) sends, for \( w \in V \), the submatrices \( \hat{S}^{(w)}[x_1*, x_2*] \) and \( \hat{T}^{(w)}[x_1*, x_2*] \) to node \( w \). Each submatrix has \( (n^{1/2}/d)^2 = O(n^{1-2/\sigma}) \) entries and there are \( n \) recipients each receiving two submatrices, for a total of \( O(n^{2-2/\sigma}) \) messages per node.

Dually, each node \( w \in V \) receives the submatrices \( \hat{S}^{(w)}[x_1*, x_2*] \) and \( \hat{T}^{(w)}[x_1*, x_2*] \) from node \( v \in V \) with label \( \ell(v) = x_1x_2 \). Node \( u \) now has the matrices \( \hat{S}^{(w)} \) and \( \hat{T}^{(w)} \). The total number of received messages is \( O(n^{2-2/\sigma}) \) per node.

**Step 4: Multiplication.** Each node \( w \in V \) computes the product \( \hat{P}^{(w)} = \hat{S}^{(w)}\hat{T}^{(w)} \).

The computation is performed entirely locally.

**Step 5: Distributing the products.** Each \( w \in V \) sends, for \( x_1, x_2 \in [n^{1/2}] \), the submatrix \( \hat{P}^{(w)}[x_1*, x_2*] \) to node \( v \in V \) with label \( x_1x_2 \). Each submatrix has \( (n^{1/2}/d)^2 = O(n^{1-2/\sigma}) \) entries and there are \( n \) recipients, for a total of \( O(n^{2-2/\sigma}) \) messages sent by each node.

Dually, each node \( v \) with label \( \ell(v) = x_1x_2 \) receives the submatrix \( \hat{P}^{(w)}[x_1*, x_2*] \) from each node \( w \in V \). The total number of received messages is \( O(n^{2-2/\sigma}) \) per node.

**Step 6: Linear combination of products.** Each node \( v \in V \) with label \( \ell(v) = x_1x_2 \) computes for \( i, j \in [d] \) the linear combination

\[
P[i\bar{x}_1*, j\bar{x}_2*] = \sum_{w \in V} \lambda_{ijw} \hat{P}^{(w)}[x_1*, x_2*].
\]

Node \( v \in V \) now has the submatrix \( P[*x_1*, *x_2*] \). The computation is performed entirely locally.

**Step 7: Assembling the product.** Each node \( v \in V \) with label \( \ell(v) = x_1x_2 \) sends, for each node \( u \in V \) with \( u_2 = x_1 \), the submatrix \( P[u, *x_2*] \) to the node \( u \). Each submatrix has \( n^{1/2} \) entries and there are \( n^{1/2} \) recipients, for a total of \( n \) messages sent by each node.

Dually, each node \( u \in V \) receives the submatrix \( P[u, *x_2*] \) from the node \( v \) with label \( \ell(v) = u_2x_2 \). Node \( u \) now has the row \( P[u, *] \) of the product matrix \( P \). The total number of received messages is \( n \) per node.

**Analysis.** The maximal number of messages sent or received by a node in the above steps is \( O(n^{2-2/\sigma}) \). Moreover, the communication pattern clearly does not depend on the input matrices, so the algorithm can be implemented in an oblivious way on the congested clique using the routing scheme of Dolev et al. [DLPI12, Lemma 1]; the running time is \( O(n^{1-2/\sigma}) \) rounds.
6.3 Upper Bounds

6.3.1 Subgraph Detection and Counting

We start by giving algorithms for various subgraph detection and counting problems in the congested clique. In subgraph detection problems, we have a fixed target graph $H$, and the task is to decide whether there is a subgraph of the input graph $G = (V, E)$ that is isomorphic to $H$. The local input for node $v \in V$ consists of the identities of the incident edges, and the local output for all nodes is either 0 or 1 depending on whether a desired subgraph exists. In subgraph counting the task is to count the number of subgraphs of $G$ isomorphic to $H$, and each node is required to output this number when the algorithm terminates.

The subgraph detection and counting algorithms we present are mainly based on applying the fast matrix multiplication to the adjacency matrix $A$ of a graph $G = (V, E)$, defined as

$$A_{uv} = \begin{cases} 1 & \text{if } (u, v) \in E, \\ 0 & \text{if } (u, v) \notin E, \end{cases}$$

where we assume that edges $\{u, v\} \in E$ are oriented both ways for undirected graphs.

Counting triangles and 4-cycles. For counting triangles, that is, 3-cycles, we use a technique first observed by Itai and Rodeh [IR78]. That is, in an undirected graph with adjacency matrix $A$, the number of triangles is known to be $\frac{1}{6} \text{tr}(A^3)$, where the trace $\text{tr}(S)$ of a matrix $S$ is the sum of its diagonal entries $S_{uu}$. Similarly, for directed graphs, the number of triangles is $\frac{1}{3} \text{tr}(A^3)$.

Alon et al. [AYZ97] generalize the above formula to counting undirected and directed $k$-cycles for small $k$. For example, the number of 4-cycles in an undirected graph is given by

$$\frac{1}{8} \left[ \text{tr}(A^4) - \sum_{v \in V} \left( 2(\deg(v))^2 - \deg(v) \right) \right].$$

Likewise, if $G$ is a loopless directed graph and we denote for $v \in V$ by $\delta(v)$ the number of nodes $u \in V$ such that $\{(u, v), (v, u)\} \subseteq E$, then the number of directed 4-cycles in $G$ is

$$\frac{1}{4} \left[ \text{tr}(A^4) - \sum_{v \in V} \left( 2(\delta(v))^2 - \delta(v) \right) \right].$$

Combining these observations with Theorem 6.1, we immediately obtain Corollary 6.2:

**Corollary 6.2.** For directed and undirected graphs, the number of triangles and 4-cycles can be computed in $O(n^6)$ rounds.

We note that similar trace formulas exist for counting $k$-cycles for $k \in \{5, 6, 7\}$, requiring only computation of small powers of $A$ and local information. We omit the detailed discussion of these in the context of the congested clique; see Alon et al. [AYZ97] for details.
Detecting k-cycles. For detection of k-cycles we leverage the color-coding techniques of Alon et al. [AYZ95] in addition to the matrix multiplication. Again, the distributed algorithm is a straightforward adaptation of a centralized one.

Fix a constant \( k \in \mathbb{N} \). Let \( c: V \to [k] \) be a labelling (or coloring) of the nodes by \( k \) colors, such that node \( v \) knows its color \( c(v) \); it should be stressed here that the coloring need not be a proper coloring in the sense of the graph coloring problem. As a first step, we consider the problem of finding a colorful k-cycle, that is, a k-cycle such that each color occurs exactly once on the cycle. We present the details assuming that the graph \( G \) is directed, but the technique works in an identical way for undirected graphs.

**Lemma 6.11.** Given a graph \( G = (V,E) \) and a coloring \( c: V \to [k] \), a colorful k-cycle can be detected in \( O(3^k n^\rho) \) rounds.

**Proof.** For each subset of colors \( X \subseteq [k] \), let \( C^{(X)} \) be a Boolean matrix such that \( C^{(X)}_{uv} = 1 \) if there is a path of length \( |X| - 1 \) from \( u \) to \( v \) containing exactly one node of each color from \( X \), and \( C^{(X)}_{uv} = 0 \) otherwise. For a singleton set \( \{i\} \subseteq [k] \), the matrix \( C^{(\{i\})} \) contains 1 only on the main diagonal, and only for nodes \( v \) with \( c(v) = i \); hence, node \( v \) can locally compute the row \( v \) of the matrix from its color. For a non-singleton color set \( X \), we have that

\[
C^{(X)} = \bigvee_{Y \subseteq X} C^{(Y)} AC^{(X \setminus Y)},
\]

(6.3)

where the products are computed over the Boolean semiring and \( \vee \) denotes element-wise logical or. Thus, we can compute \( C^{(X)} \) for all \( X \subseteq [k] \) by applying (6.3) recursively; there is a colorful k-cycle in \( G \) if and only if there is a pair of nodes \( u,v \in V \) such that \( C^{([k])}_{uv} = 1 \) and \( (v,u) \in E \).

To leverage fast matrix multiplication, we simply perform the operations stated in (6.3) over the ring \( \mathbb{Z} \) and observe that an entry of the resulting matrix is non-zero if and only if the corresponding entry of \( C^{(X)} \) is non-zero. The application of (6.3) needs two matrix multiplications for each pair \( (Y,X) \) with \( Y \subseteq [k] \) and \( |Y| = \lceil |X|/2 \rceil = \lceil k/2 \rceil \). The number of such pairs is bounded by \( 3^k \); to see this, note that the set \( \{(Y,X) : Y \subseteq X \subseteq [k]\} \) can be identified with the set \( \{0,1,2\}^k \) of trinary strings of length \( k \) via the bijection \( w_1 w_2 \ldots w_k \mapsto (\{i : w_i = 0\}, \{i : w_i \leq 1\}) \), and the set \( \{0,1,2\}^k \) has size exactly \( 3^k \). Thus, the total number of matrix multiplications used is at most \( O(3^k) \).

We can now use Lemma 6.11 to prove Theorem 6.3; while we cannot directly construct a suitable coloring from scratch for an uncolored graph, we can try an exponential in \( k \) number of colorings to find a suitable one.

**Theorem 6.3.** For directed and undirected graphs, the existence of k-cycles can be detected in \( 2O(k)n^\rho \log n \) rounds.

**Proof.** To apply Lemma 6.11, we first have to obtain a coloring \( c: V \to [k] \) that assigns each color once to at least one k-cycle in \( G \), assuming that one exists. If we pick a color
$c(v) \in [k]$ for each node uniformly at random, then for any $k$-cycle $C$ in $G$, the probability that $C$ is colorful in the coloring $c$ is $k!/k^k > e^{-k}$. Thus, by picking $e^k \log n$ uniformly random colorings and applying Lemma 6.11 to each of them, we find a $k$-cycle with high probability if one exists.

This algorithm can also be derandomized using standard techniques. A $k$-perfect family of hash functions $\mathcal{H}$ is a collection of functions $h: V \to [k]$ such that for each $U \subseteq V$ with $|U| = k$, there is at least one $h \in \mathcal{H}$ such that $h$ assigns a distinct color to each node in $U$. There are known constructions that give such families $\mathcal{H}$ with $|\mathcal{H}| = 2^{O(k)} \log n$ and these can be efficiently constructed \cite{AYZ99}; thus, it suffices to take such an $\mathcal{H}$ and apply Lemma 6.11 for each coloring $h \in \mathcal{H}$.

Detecting 4-cycles. We have seen how to count 4-cycles with the help of matrix multiplication in $O(n^\rho)$ rounds. We now show how to detect 4-cycles in $O(1)$ rounds; this situation is analogous with the centralized setting, where the fastest known triangle detection algorithm runs in $O(n^\omega)$ time, while 4-cycles can be detected in $O(n^2)$ time \cite{RL85}. Indeed, our distributed 4-cycle detection is inspired by the centralized algorithm, based on the observation that sufficiently dense graphs necessarily have a 4-cycle; however, taking advantage of this observation requires more effort in the distributed setting.

Like the centralized $O(n^2)$ algorithm, our algorithm does not make direct use of matrix multiplication algorithms. However, the key part of the algorithm can be interpreted as an efficient routine for sparse matrix multiplication, under a specific definition of sparseness.

Let

$$P(X, Y, Z) = \{(x, y, z): x \in X, y \in Y, z \in Z,\{x, y\} \in E, \{y, z\} \in E\}$$

consist of all distinct 2-walks (paths of length 2) from $X$ through $Y$ to $Z$. We will use again the shorthand notation $v$ for $\{v\}$ and $*$ for $V$; for example, $P(x, *, *)$ consists of all walks of length 2 from node $x$. There exists a 4-cycle if and only if $|P(x, *, z)| \geq 2$ for some $x \neq z$.

On a high level, the algorithm proceeds as follows.

1. Each node $x$ computes $|P(x, *, *)|$. If $|P(x, *, *)| \geq 2n - 1$, then there has to be some $z \neq x$ such that $|P(x, *, z)| \geq 2$, which implies that there exists a 4-cycle, and the algorithm stops.

2. Otherwise, each node $x$ finds $P(x, *, *)$ and checks if there exists some $z \neq x$ such that $|P(x, *, z)| \geq 2$.

The first phase is easy to implement in $O(1)$ rounds. The key idea is that if the algorithm does not stop in the first phase, then the total volume of $P(*, *, *)$ is sufficiently small so that we can afford to gather $P(x, *, *)$ for each node $x$ in $O(1)$ rounds.
We now present the algorithm in more detail. We write $N(x)$ for the neighbours of node $x$. To implement the first phase, it is sufficient for each node $y$ to broadcast $\deg(y) = |N(y)|$ to all other nodes; we have

$$|P(x,*,*)| = \sum_{y \in N(x)} \deg(y).$$

Now let us explain the second phase. Each node $y$ is already aware of $N(y)$ and hence it can construct $P(*,y,*) = N(y) \times \{y\} \times N(y)$. Our goal is to distribute the set of all 2-walks

$$\bigcup_y P(*,y,*) = P(*,*,*) = \bigcup_x P(x,*,*)$$

so that each node $x$ will know $P(x,*,*)$.

In the second phase, we have

$$\sum_y \deg(y)^2 = \sum_y |P(*,y,*)| = \sum_x |P(x,*,*)| < 2n^2.$$

Using this bound, we obtain the following lemma.

**Lemma 6.12.** There exist sets $A(y) \subseteq V$ and $B(y) \subseteq V$ for each $y \in V$ such that the following holds:

- $|A(y)| = |B(y)| \geq \deg(y)/8$ for all $y \in V$, and
- the tiles $A(y) \times B(y)$ and $A(x) \times B(x)$ are disjoint subsets of the square $V \times V$ for $y \neq x$.

Moreover, such sets can be constructed in $O(1)$ rounds in the congested clique model so that all nodes know the sets $A(y)$ and $B(y)$ for all $y \in V$.

**Proof.** Let $f(y)$ be $\deg(y)/4$ rounded down to the nearest power of 2, and let $k$ be $n$ rounded down to the nearest power of 2; note that we have $\sum_y f(y)^2 \leq \sum \deg(y)^2/16 < n^2/8 < k^2$. The idea is to now show that we can pack tiles of dimensions $f(y) \times f(y)$ for $y \in V$ inside a square of dimensions $k \times k$ without any overlap. This can be done using the following iterative procedure for $i = 1, 2, \ldots$:

- Assume that before step $i$, we have partitioned the square in sub-squares of dimensions $k/2^{i-1} \times k/2^{i-1}$, and each sub-square is either completely full or completely empty. In particular, this is trivially true for $i = 1$, as the only sub-square is the whole $[k] \times [k]$.

- During step $i$, we divide each sub-square into 4 parts, and place all tiles of dimensions $f(y) = k/2^i$ to the new empty sub-squares. There are sufficiently many new sub-squares, as all empty spaces are in the $k/2^i \times k/2^i$ sub-squares, so running out of space would imply $\sum_y f(y)^2 > k^2$. 

97

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-16 - 2017
After step $i$, we have partitioned the square in sub-squares of dimensions $k/2^i \times k/2^i$, and each sub-square is either completely full or completely empty.

The process terminates when all the tiles have been assigned; this way we have allocated disjoint tiles $A(y) \times B(y) \subseteq [k] \times [k] \subseteq V \times V$ for each $y$, with $|A(y)| = |B(y)| = f(y) \geq \deg(y)/8$.

To implement this in the congested clique model, it is sufficient that each $y$ broadcasts $\deg(y)$ to all other nodes, and then all nodes follow the above procedure to compute the sets $A(y)$ and $B(y)$ locally.

Now we will use the tiles $A(y) \times B(y)$ to implement the second phase of 4-cycle detection. Since $|A(y)| = |B(y)| \geq \deg(y)/8$, we can for each $y \in V$ partition the neighbourhood $N(y)$ into $|A(y)| = |B(y)|$ sets of size at most 8. For convenience, let us fix a such partition, and label the sets in the partition with members of both $A(y)$ and $B(y)$ to obtain two set families:

- The sets $N_A(y, a)$ for $a \in A(y)$, which form a partition of $N(y)$ with $|N_A(y, a)| \leq 8$.
- The sets $N_B(y, b)$ for $b \in B(y)$, which form a partition of $N(y)$ with $|N_B(y, b)| \leq 8$.

Note that we can assume that $A(y)$ and $B(y)$ are globally known by Lemma 6.12. Hence we can assume that a node can compute $N_A(y, a)$ and $N_B(y, b)$ if it knows $N(y)$.

With this notation, the algorithm proceeds as follows (see Figure 6.3):

- **p1sep=1.5ex** For all $y \in V$ and $a \in A(y)$, node $y$ sends $N_A(y, a)$ to $a$.
  
  This step can be implemented in $O(1)$ rounds.

- **p2sep=2.5ex** For each $y$ and each pair $(a, b) \in A(y) \times B(y)$, node $a$ sends $N_A(y, a)$ to $b$.
  
  Note that for each $(a, b)$ there is at most one $y$ such that $(a, b) \in A(y) \times B(y)$; hence over each edge we send only $O(1)$ words. Therefore this step can be implemented in $O(1)$ rounds.
At this point, each $b \in V$ has received a copy of $N(y)$ for all $y$ with $b \in B(y)$. Node $b$ computes

$$W(y, b) = N(y) \times \{y\} \times N_B(y, b),$$

$$W(b) = \bigcup_{y:b \in B(y)} W(y, b).$$

This is local computation; it takes 0 rounds.

We now give a lemma that captures the key properties of the algorithm.

**Lemma 6.13.** The sets $W(b)$ form a partition of the set $P(*, *, *)$. Moreover, for each $b$ we have $|W(b)| = O(n)$.

**Proof.** For the first claim, observe that the sets $P(*, y, *)$ for $y \in V$ form a partition of $P(*, *, *)$, the sets $W(y, b)$ for $b \in B(y)$ form a partition of $P(*, y, *)$, and each set $W(y, b)$ is part of exactly one $W(b)$.

For the second claim, let $Y$ consist of all $y \in V$ with $b \in B(y)$. As the tiles $A(y) \times B(y)$ are disjoint for all $y \in Y$, and all $y \in Y$ have the common value $b \in B(y)$, it has to hold that the sets $A(y)$ are disjoint subsets of $V$ for all $y \in Y$. Therefore

$$\sum_{y \in Y} |N(y)| = \sum_{y \in Y} \deg(y) \leq \sum_{y \in Y} 8|A(y)| \leq 8|V| = 8n.$$

With $|N_B(y)| \leq 8$ we get

$$|W(b)| = \sum_{y \in Y} |W(y, b)| \leq 8 \sum_{y \in Y} |N(y)| \leq 64n. \quad \square$$

Now we are almost done: we have distributed the elements of $P(*, *, *)$ evenly among $V$ so that each node only holds $O(n)$ elements. Finally, we use the dynamic routing scheme [Len13] to gather $P(x, *, *)$ at each node $x \in V$; here each node needs to send $O(n)$ words and receive $O(n)$ words, and the running time is therefore $O(1)$ rounds. In conclusion, we can implement both phases of 4-cycle detection in $O(1)$ rounds.

**Theorem 6.4.** For undirected graphs, the existence of 4-cycles can be detected in $O(1)$ rounds.

### 6.3.2 Girth

**Undirected girth.** Recall that the *girth* $g$ of an undirected unweighted graph $G = (V, E)$ is the length of the shortest cycle in $G$. To compute the girth in the congested clique model, we leverage the fast cycle detection algorithm and the following lemma giving a trade-off between the girth and the number of edges. A similar approach of bounding from above the number of edges of a graph that contains no copies of some given subgraph was taken by Drucker et al. [DKO14].
Lemma 6.14 ([Mat02 pp. 362–363]). A graph with girth \( g \) has at most \( n^{1+1/[(g-1)/2]} + n \) edges.

If the graph is dense, then by the above lemma it must have small girth and we can use fast cycle detection to compute it; otherwise, the graph is sparse and we can learn the complete graph structure.

**Theorem 6.15.** For undirected graphs, the girth can be computed in \( \tilde{O}(n^\rho) \) rounds (or in \( n^{o(1)} \) rounds, if \( \rho = 0 \)).

**Proof.** Assume for now that \( \rho > 0 \), and fix \( \ell = \lceil 2 + 2/\rho \rceil \). Each node collects the degrees of all nodes and computes the total number of edges. If there are at most \( n^{1+1/\ell} + n = O(n^{1+\rho}) \) edges, we can collect full information about the graph structure at all nodes in \( O(n^\rho) \) rounds using an algorithm of Dolev et al. [DLP12], and each node can then compute the girth locally.

Otherwise, by Lemma 6.14, the graph has girth at most \( \ell \). Thus, for \( k = 3, 4, \ldots, \ell \), we try to find a \( k \)-cycle using Theorem 6.3 in \( \ell \cdot 2^{O(\ell)} n^\rho \log n = \tilde{O}(n^\rho) \) rounds. When such a cycle is found for some \( k \), we stop and return \( k \) as the girth.

Finally, if \( \rho = 0 \), we pick \( \ell = \log \log n \), and both cases take \( n^{o(1)} \) rounds.

**Directed girth.** For a directed graph, the girth is defined as the length of the shortest directed cycle; the main difference is that directed girth can be 1 or 2. While the trade-off of Lemma 6.14 cannot be used for directed graphs, we can use a simpler technique of Itai and Rodeh [IR78].

Let \( G = (V, E) \) be a directed graph; we can assume that there are no self-loops in \( G \), as otherwise girth is 1 and we can detect this with local computation. Let \( B^{(i)} \) be a Boolean matrix defined by setting \( B^{(i)}_{uv} = 1 \) if there is a path of length \( \ell \) from \( u \) to \( v \) for \( 1 \leq \ell \leq i \), and \( B^{(i)}_{uv} = 0 \) otherwise. Clearly, we have that \( B^{(1)} = A \). Moreover, if \( i = j + k \), we have

\[
B^{(i)} = (B^{(j)}B^{(k)}) \lor A,
\]

where the matrix product is over the Boolean semiring and \( \lor \) denotes element-wise logical or.

**Corollary 6.16.** For directed graphs, the girth can be computed in \( \tilde{O}(n^\rho) \) rounds.

**Proof.** It suffices to find smallest \( \ell \) such that there is \( v \in V \) with \( B^{(\ell)}_{vv} = 1 \); clearly \( \ell \) is then the girth of graph \( G \). We first compute \( A = B^{(1)}, B^{(2)}, B^{(4)}, B^{(8)}, \ldots \) using (6.4) with \( j = k = i/2 \) until we find \( i \) such that \( B^{(i)}_{vv} = 1 \) for some \( v \in V \). We then know that the girth is between \( i \) and \( i/2 \); we can perform binary search on this interval to find the girth, using (6.4) to evaluate the intermediate matrices. This requires \( O(\log n) \) calls to the matrix multiplication algorithm.

\[ \square \]
6.3.3 Routing and Shortest Paths

In this section, we present algorithms for variants of the all-pairs shortest paths (APSP) problem. In the congested clique model, the local input for a node \( u \in V \) in the APSP problem is a vector containing the local edge weights \( W(u, v) \) for \( v \in V \). The output for \( u \in V \) is the actual shortest path distances \( d(u, v) \) for each other node \( v \in V \), along with the routing table entries \( R[u, v] \), where each entry \( R[u, v] = w \in V \) is a node such that \((u, w) \in V \) and \( w \) lies on a shortest path from \( u \) to \( w \). For convenience, we use the same notation for directed and undirected graphs, assume \( W(u, v) = \infty \) if \((u, v) \notin E \), and for unweighted graphs, we set \( W(u, v) = 1 \) for each \((u, v) \in E \).

For a graph \( G = (V, E) \) with edge weights \( W \), we define the weight matrix \( W \) as

\[
W_{uv} = \begin{cases} W(u, v) & \text{if } u \neq v, \\ 0 & \text{if } u = v. \end{cases}
\]

Our APSP algorithms are mostly based on the manipulation of the weight matrix \( W \) and the adjacency matrix \( A \), as defined in Section 6.3.1.

**Distance product and iterated squaring.** Matrix multiplication can be used to compute the shortest path distances via *iterated squaring* of the weight matrix over the min-plus semiring \([\text{Fur70}, \text{Mun71}, \text{FM71}]\). That is, the matrix product is the distance product, also known as the min-plus product or tropical product, defined as

\[
(S \star T)_{uv} = \min_w (S_{uw} + T_{wv}).
\]

Given a graph \( G = (V, E) \) with weight matrix \( W \), the \( n \)th distance product power \( W^n \) gives the actual distances in \( G \) as \( d(v, u) = W^n_{vu} \). Computing \( W^n \) can be done with \( \lceil \log n \rceil \) distance products by iteratively squaring \( W \). Moreover, sending integer weights in \( \{0, \pm 1, \ldots, \pm M\} \) takes \( O(\lceil \log M / \log n \rceil) \) rounds, and so does sending a sum of at most \( n \) such values. Combining these observation with the semiring algorithm from Theorem 6.1, we immediately obtain a simple APSP algorithm for the congested clique.

**Corollary 6.6.** For directed graphs with edge weights in \( \{0, \pm 1, \ldots, \pm M\} \) and for undirected graphs with edge weights in \( \{0, 1, \ldots, M\} \), all-pairs shortest paths can be computed in \( O(n^{1/3} \log n \lceil \log M / \log n \rceil) \) communication rounds.

The subsequent APSP algorithms we discuss in this section are, for the most part, similarly based on the iterated squaring of the weight matrix; the main difference is that we replace the semiring matrix multiplication with distance product algorithms derived from the fast matrix multiplication algorithm.

**Constructing routing tables.** The iterated squaring algorithm of Corollary 6.6 can be adapted to also compute a routing table \( R \) as follows. Assume that our distance
product algorithm also provides for the distance product $S \star T$ a witness matrix $Q$ such that if $Q_{uv} = w$, then $(S \star T)_{uv} = S_{uw} + T_{wv}$. With this information, we can compute the routing table $R$ during the iterated squaring algorithm; when we compute the product $W^{2i} = W^i \star W^i$, we also obtain a witness matrix $Q$, and update the routing table by setting 

$$R[u,v] = R[u, Q_{uv}]$$

for each $u, v \in V$ with $W^{2i} < W^i_{uv}$.

The semiring matrix multiplication can be easily modified to produce witnesses, but for the subsequent distance product algorithms based on fast matrix multiplication this is not directly possible. However, we can apply known techniques from the centralized setting to obtain witnesses also in these cases \[\text{AN96, Sei95, Zwi02}\]; we refer to Section 6.3.4 for details.

**Unweighted undirected APSP.** In the case of unweighted undirected graphs, we can obtain exact all-pairs shortest paths via a technique of Seidel \[\text{Sei95}\]. Specifically, let $G = (V, E)$ an unweighted undirected graph with adjacency matrix $A$; the $k^{th}$ power $G^k$ of $G$ is a graph with node set $V$ and edge set $\{(u,v): d(u,v) \leq k\}$. In particular, the square graph $G^2$ can be constructed in $O(n^\rho)$ rounds from $G$, as the adjacency matrix of $G^2$ is $A^2 \lor A$, where the product is over the Boolean semiring and $\lor$ denotes element-wise logical or.

The following lemma of Seidel allows us to compute distances in $G$ if we already know distances in the square graph $G^2$; to avoid ambiguity, we write in this subsection $d_G(u,v)$ for the distances in a graph $G$.

**Lemma 6.17** (\[\text{Sei95}\]). Let $G = (V, E)$ be an unweighted undirected graph with adjacency matrix $A$, and let $D$ be a distance matrix for $G^2$, that is, a matrix with the entries $D_{uv} = d_G^2(u,v)$. Let $S = DA$, where the product is computed over integers. Then

$$d_G(u,v) = \begin{cases} 2d_G^2(u,v) & \text{if } S_{uv} \geq d_G^2(u,v) \deg_G(v), \\ 2d_G^2(u,v) - 1 & \text{if } S_{uv} < d_G^2(u,v) \deg_G(v). \end{cases}$$

We can now recover all-pairs shortest distances in an undirected unweighted graph by recursively applying Lemma 6.17.

**Corollary 6.7.** For undirected, unweighted graphs, all-pairs shortest paths can be computed in $\tilde{O}(n^\rho)$ rounds.

*Proof.* Let $G = (V, E)$ be an unweighted undirected graph with adjacency matrix $A$. We first compute the adjacency matrix for $G^2$; as noted above, this can be done in $O(n^\rho)$ rounds. There are now two cases to consider.

1. If $G = G^2$, then $d_G(u,v) = 1$ if $u$ and $v$ are adjacent in $G$, and $d_G(u,v) = \infty$ otherwise; thus, we are done.
2. Otherwise, we compute all-pairs shortest path distances in the graph $G^2$; since we have already constructed the adjacency matrix for $G^2$, we can do the distance computation in $G^2$ by recursively calling this algorithm with input graph $G^2$. Then, we construct the matrix $D$ with entries $D_{uv} = d_{G^2}(u, v)$ as in Lemma 6.17 and compute $S = DA$. We can recover distances in $G$ using Lemma 6.17 as each node can transmit its degree in $G$ to each other node in a single round and then check the conditions of the lemma locally.

The recursion terminates in $O(\log n)$ calls, as the graph $G^n$ consists of disjoint cliques.

**Weighted APSP with small weights.** By embedding the distance product of two matrices into a suitable ring, we can use fast ring matrix multiplication to compute all-pairs shortest distances \([Yuv76]\); however, this is only practical for very small weights, as the ring embedding exponentially increases the amount of bits required to transmit the matrix entries. The following lemma encapsulates this idea.

**Lemma 6.18.** Given $n \times n$ matrices $S$ and $T$ with entries in \(\{0, 1, \ldots, M\} \cup \{\infty\}\), the distance product $S \star T$ can be computed in $O(Mn^\rho)$ rounds.

**Proof.** We encode the distance product into a product of matrices whose entries are polynomials in the polynomial ring $\mathbb{Z}[X]$, where $X$ is a formal variable, and use fast matrix multiplication to compute the matrix product in the polynomial ring. That is, we construct new matrices $S^*$ and $T^*$ by replacing each matrix entry $w$ with the polynomial $X^w$; values $\infty$ are replaced by 0. We then compute the product $S^* \cdot T^*$ over $\mathbb{Z}[X]$; all polynomials involved in the computation have degree at most $2M$ and their coefficients are integers of absolute value at most $n^{O(1)}$, so this computation can be done in $O(Mn^\rho)$ rounds. Finally, we can recover each matrix entry $(S \star T)_{uv}$ in the original distance product by taking the degree of the lowest-degree monomial in $(S^* \cdot T^*)_{uv}$.

Using iterated squaring together with Lemma 6.18, we can compute all-pairs shortest paths up to a small distance $M$ quickly; that is, we want to compute a matrix $B$ such that

$$B_{uv} = \begin{cases} 
d(u, v) & \text{if } d(u, v) \leq M, \\
\infty & \text{if } d(u, v) > M.
\end{cases}$$

This can be done by replacing all weights over $M$ with $\infty$ before each squaring operation to ensure that we do not operate with too large values, giving us the following lemma.

**Lemma 6.19.** Given a directed, weighted graph with non-negative integer weights, all-pairs shortest paths up to distance $M$ can be computed in $O(Mn^\rho)$ rounds.

The above lemma can be used to compute all-pairs shortest paths quickly assuming that the weighted diameter of the graph is small; recall that the weighted diameter of a weighted graph is the maximum distance between any pair of nodes.
Corollary 6.8. For directed and undirected graphs with positive integer edge weights and weighted diameter $U$, all-pairs shortest paths can be computed in $\tilde{O}(U^n)$ rounds.

Proof. If we know that the weighted diameter is $U$, we can simply apply Lemma 6.19 with $M = U$. However, if we do not know $U$ beforehand, we can (1) first compute the reachability matrix $R$ with $R_{uv} = 1$ if there is a path from $u$ to $v$, and $R_{uv} = 0$ otherwise, using iterated squaring of the adjacency matrix over the Boolean semiring as in the directed girth algorithm in Section 6.3.2 (2) guess $U = 1$ and compute all-pairs shortest paths up to distance $U$, and (3) check if we obtained distances for all pairs that are reachable according to the reachability matrix; if not, then we double our guess for $U$ and repeat steps (2) and (3). \qed

Approximate weighted APSP. We can leverage the above result and a rounding technique to obtain a fast $(1 + o(1))$-approximation algorithm for the weighted directed APSP problem. Similar rounding-based approaches were previously used by Zwick [Zwi02] in the centralized setting and by Nanongkai [Nan14] in the distributed setting; however, the idea can be traced back much further [RT85].

We first consider the computation of a $(1 + \delta)$-approximate distance product over integers for a given $\delta > 0$; the following lemma is an analogue of one given by Zwick [Zwi02] in a centralized setting.

Lemma 6.20. Given two $n \times n$ matrices $S$ and $T$ with entries in $\{0, 1, \ldots, M\} \cup \{\infty\}$, a matrix $\tilde{P}$ satisfying

$$ P_{uv} \leq \tilde{P}_{uv} \leq (1 + \delta)P_{uv} \quad \text{for } u, v \in V, $$

where $P = S \star T$ is the distance product of $S$ and $T$, can be computed in $O\left(n^\rho(\log_2 M)/\delta\right)$ rounds.

Proof. For $i \in \{0, \ldots, \lceil \log_2 M \rceil\}$, let $S^{(i)}$ be the matrix defined as

$$ S^{(i)}_{uv} = \begin{cases} \left\lceil S_{uv}/(1 + \delta)^i \right\rceil & \text{if } S_{uv} \leq 2(1 + \delta)^{i+1}/\delta, \\ \infty & \text{otherwise}, \end{cases} $$

and let $T^{(i)}$ be defined similarly for $T$. Furthermore, let us define $P^{(i)} = S^{(i)} \star T^{(i)}$. We now claim that selecting

$$ \tilde{P}_{uv} = \min_i \left\{ \left\lceil (1 + \delta)^i P^{(i)}_{uv} \right\rceil \right\} $$

gives a matrix $\tilde{P}$ with the desired properties.

It follows directly from the definitions that $P_{uv} \leq \tilde{P}_{uv}$, so it remains to prove the other inequality. Thus, let us fix $u, v \in V$, and let $w \in V$ be such that

$$ P_{uv} = S_{uw} + T_{wv}. $$
Let \( j = \lceil \log_{1+\delta}(\delta P_{uv}/2) \rceil \); this choice of \( j \) means that \( 2(1 + \delta)^j \delta \leq P_{uv} \leq 2(1 + \delta)^{j+1}/\delta \). Since \( S_{uv} \) and \( T_{uv} \) are both bounded from above by \( P_{uv} \), the entries \( S_{uv}^{(j)} \) and \( T_{uv}^{(j)} \) are finite. It follows that we have

\[
(1 + \delta)^j S_{uv}^{(j)} \leq S_{uv} + (1 + \delta)^j,
\]

\[
(1 + \delta)^j T_{uv}^{(j)} \leq T_{uv} + (1 + \delta)^j,
\]

and therefore

\[
(1 + \delta)^j P_{uv}^{(j)} \leq (1 + \delta)^j (S_{uv}^{(j)} + T_{uv}^{(j)})
\]

\[
\leq S_{uv} + T_{uv} + 2(1 + \delta)^j
\]

\[
\leq P_{uv} + \delta P_{uv} = (1 + \delta)P_{uv}.
\]

Finally, we have \( \tilde{P}_{uv} \leq \lfloor (1 + \delta)^j P_{uv}^{(j)} \rfloor \leq (1 + \delta)P_{uv} \).

To see that we can compute the matrix \( \tilde{P} \) in the claimed time, we first note that each of the matrices \( S^{(i)} \) and \( T^{(i)} \) can be constructed locally by the nodes. The product \( P^{(i)} = S^{(i)} \times T^{(i)} \) can be computed in \( O(n^\rho/\delta) \) rounds for a single index \( i \) by Lemma 6.18 as the entries of \( S^{(i)} \) and \( T^{(i)} \) are integers bounded from above by \( O(1/\delta) \); this is repeated for each index \( i \), and the number of iterations is thus \( O(\log_{1+\delta} n) \). Finally, the matrix \( \tilde{P} \) can be constructed from matrices \( P^{(i)} \) locally.

Using Lemma 6.20, we obtain a \((1 + o(1))\)-approximate APSP algorithm.

**Theorem 6.9.** For directed and undirected graphs with edge weights in \( \{0, 1, \ldots, 2^{n^{\rho(1)}}\} \), all-pairs shortest paths can be \((1 + o(1))\)-approximated in \( O(n^{\rho + o(1)}) \) rounds.

**Proof.** Let \( G = (V, E) \) be a directed weighted graph with edge weights in \( \{0, 1, \ldots, M\} \), where \( M = 2^{n^{\rho(1)}} \). To compute the approximate shortest paths, we apply iterated squaring over the min-plus semiring to the weight matrix \( W \) of \( G \), but use the approximate distance product algorithm of Lemma 6.20 to compute the products. After \( \lceil \log n \rceil \) iterations, we obtain a matrix \( \tilde{D} \); by induction we have for \( u, v \in V \) that

\[
d(u, v) \leq \tilde{D}_{uv} \leq (1 + \delta)^{\lceil \log n \rceil} d(u, v).
\]

Selecting \( \delta = o(1/\log n) \), this gives a \((1 + o(1))\)-approximation for the shortest distances.

To analyse the running time, we observe that we call the algorithm of Lemma 6.20 \( \lceil \log n \rceil \) times; as the maximum distance between nodes in \( G \) is \( nM = 2^{n^{\rho(1)}} \), the running time of each call is bounded by

\[
O \left( \frac{n^{\rho} \log_{1+\delta}(nM)}{\delta} \right) = O \left( \frac{n^{\rho+o(1)}}{\delta \log(1 + \delta)} \right).
\]

For sufficiently small \( \delta \), we have \( 1/(\delta \log(1 + \delta)) = O(1/\delta^2) \). Thus, for, e.g., \( \delta = 1/\log^2 n = o(1/\log n) \), the total running time is \( O(n^{\rho+o(1)}) \), as the polylogarithmic factors are subsumed by \( n^{o(1)} \).

\( \square \)
6.3.4 Witness Detection for Distance Product

Witness problem for the distance product. As noted in Section 6.3.3, to recover the routing table in the APSP algorithms based on fast matrix multiplication in addition to computing the shortest path lengths, we need the ability to compute a witness matrix for the distance product $S \star T$. That is, we need to find a matrix $Q$ such that if $Q_{uv} = w$, then $(S \star T)_{uv} = S_{uw} + T_{wv}$; in this case, the index $w$ is called a witness for the pair $(u, v)$.

While one can easily modify the semiring matrix multiplication algorithm to provide witnesses, this is not directly possible with the fast matrix multiplication algorithms. However, known techniques from centralized algorithms [AN96, Sei95, Zwi02] can be adapted to the congested clique to bridge this gap.

Lemma 6.21. If the distance product for two $n \times n$ matrices $S$ and $T$ can be computed in $M$ rounds, then a witness matrix for $S \star T$ can be computed in $M \operatorname{polylog}(n)$ rounds.

The rest of this section outlines the proof of this lemma. While we have stated it for the distance product, it should be noted that the same techniques also work for the Boolean semiring matrix product.

Preliminaries. For matrix $S$ and index subsets $U, W \subseteq V$, we define the matrix $S(U, W)$ as

$$S(U, W)_{uw} = \begin{cases} S_{uw} & \text{if } u \in U \text{ and } w \in W, \\ \infty & \text{otherwise.} \end{cases}$$

That is, we set all rows and columns not indexed by $U$ and $W$ to $\infty$. As before, we use $\star$ as a shorthand for the whole index set $V$.

Finding unique witnesses. As a first step, we compute witnesses for all $(u, v)$ that have a unique witness, that is, there is exactly one index $w$ such that $(S \star T)[u, v] = S[u, w] + T[w, v]$. To construct a candidate witness matrix $Q$, let $V(i) \subseteq V$ be the set of indices $v$ such that bit $i$ in the binary presentation of $v$ is 1. For $i = 1, 2, \ldots, \lceil \log n \rceil$, we compute the distance product $P(i) = S(\ast, V_i) \star T(V_i, \ast)$. If $P(i)_{uv} = (S \star T)_{uv}$, then we set the $i^{th}$ bit of $Q_{uv}$ to 1, and otherwise we set it to 0.

If there is a unique witness for $(u, v)$, then $Q_{uv}$ is correct, and we can check if the candidate witness $Q_{uv} = w$ is correct by computing $S_{uw} + T_{wv}$; see Zwick [Zwi02, §. 3]. The algorithm clearly uses $O(\log n)$ matrix multiplications.

Finding witnesses in the general case. To find witnesses for all indices $(u, v)$, we reduce the general case to the case of unique witnesses. For simplicity, we only present a randomized version of this algorithm; for derandomisation see Zwick [Zwi02] and Alon and Naor [AN96].

Let $i \in \{0, 1, \ldots, \lceil \log n \rceil - 1\}$. We use the following procedure to attempt to find witnesses for all $(u, v)$ that have exactly $r$ witnesses for $n/2^{i+1} \leq r < n/2^i$.
1. Let \( m = \lceil c \log n \rceil \) for a sufficiently large constant \( c \). For \( j = 1, 2, \ldots, m \), construct a subset \( V_j \subseteq V \) by picking \( 2^i \) values \( v_1, v_2, \ldots, v_2^i \) from \( V \) with replacement, and let \( V_j = \{ v_1, v_2, \ldots, v_2^i \} \).

2. For each \( V_j \), use the unique witness detection for the product \( S(*, V_j) \ast T(V_j, *) \) to find candidate witnesses \( Q_{uv} \) for all pairs \( (u, v) \), and keep those \( Q_{uv} \) that are witnesses for \( S \ast T \).

Let \( (u, v) \) be a pair with \( r \) witnesses for \( n/2^{i+1} \leq r < n/2^i \). For each \( j = 1, 2, \ldots, m \), the probability that \( V_j \) contains exactly one witness for \( (u, v) \) is at least \( (2e)^{-1} \) (see Seidel [Sei95, Claim 8]). Thus, the probability that we do not find a witness for \( (u, v) \) is bounded by \( (1 - (2e)^{-1})^{c \log n} = n^{-\Omega(c)} \).

Repeating the above procedure for \( i = 0, 1, \ldots, \lfloor \log n \rfloor - 1 \) ensures that the probability of not finding a witness for any fixed \( (u, v) \) is at most \( n^{-\Omega(c)} \). By the union bound, the probability that there is any pair of indices \( (u, v) \) for which no witness is found is \( n^{-\Omega(c)} \), i.e., with high probability the algorithm succeeds. Moreover, the total number of calls to the distance product is \( O((\log n)^3) \), giving Lemma 6.21.

### 6.4 Lower Bounds

**Lower bounds for matrix multiplication implementations.** While proving unconditional lower bounds for matrix multiplication in the congested clique model seems to be beyond the reach of current techniques, as discussed in Section 6.1.3, it can be shown that the results given in Theorem 6.1 are essentially optimal distributed implementations of the corresponding centralized algorithms. To be more formal, let \( C \) be an arithmetic circuit for matrix multiplication; we say that an implementation of \( C \) in the congested clique model is a mapping of the gates of \( C \) to the nodes of the congested clique. This naturally defines a congested clique algorithm for matrix multiplication, with the wires in \( C \) between gates assigned to different nodes defining the communication cost of the algorithm.

Various authors, considering different parallel models, have shown that in any implementation of the trivial \( \Theta(n^3) \) matrix multiplication on a parallel machine with \( P \) processors there is at least one processor that has to send or receive \( \Omega(n^2/P^{2/3}) \) matrix entries [ACS90, ITT04, Tis98]. As these models can simulate the congested clique, a similar lower bound holds for congested clique implementations of the trivial \( O(n^3) \) matrix multiplication. In the congested clique, each processor sends and receives \( n \) messages per round (up to logarithmic factors) and \( P = n \), yielding a lower bound of \( \tilde{\Omega}(n^{1/3}) \) rounds.

The trivial \( \Theta(n^3) \) matrix multiplication is optimal for circuits using only semiring addition and multiplication [Ker70]. The task of \( n \times n \) matrix multiplication over the min-plus semiring can be reduced to APSP with a constant blowup [AHU74, pp. 202–205], hence the above bound applies also to any APSP algorithm that only uses minimum and
addition operations. This means that current techniques for similar problems, like the one used in the fast MST algorithm of Lotker et al. [LPPP05] cannot be extended to solve APSP.

**Corollary 6.22.** *Any implementation of the trivial $\Theta(n^3)$ matrix multiplication, and any APSP algorithm which only sums weights and takes the minimum of such sums, require $\tilde{\Omega}(n^{1/3})$ communication rounds in the congested clique model.*

However, known results on centralized APSP and distance product computation give reasons to suspect that this bound can be broken if we allow subtraction; in particular, translating the recent result of Williams [Wil14] might allow for running time of order $n^{1/3}/2^{\Omega(\sqrt{\log n})}$ for APSP in the congested clique.

Concerning fast matrix multiplication algorithms, Ballard et al. [BDHS12] have proven lower bounds for parallel implementations of Strassen-like algorithms. Their seminal work is based on building a DAG representing the linear combinations of the inputs before the block multiplications, and the linear combinations of the results of the multiplications (“decoding”) as the output matrix. The parallel computation induces an assignment of the graph vertices to the processes, and the edges crossing the partition represent the communication. Using an expansion argument, Ballard et al. show that in any partition a graph representing an $\Omega(n^\sigma)$ algorithm there is a process communicating $\Omega(n^{2-2/\sigma})$ values. See also [BDHS14] for a concise account of the technique.

The lower bound holds for Strassen’s algorithm, and for a family of similar algorithms, but not for any matrix multiplication algorithm (See [BDHS12, § 5.1.1]). A matrix multiplication algorithm is said to be *Strassen-like* if it is recursive, its decoding graph discussed above is connected, and it computes no scalar multiplication twice. As each process communicates at most $O(n)$ values in a round, the implementation of an $\Omega(n^\sigma)$ strassen-like algorithm must take $\Omega(n^{1-2/\sigma})$ rounds.

**Corollary 6.23.** *Any implementation of a Strassen-like matrix multiplication algorithm using $\Omega(n^\sigma)$ element multiplications requires $\tilde{\Omega}(n^{1-2/\sigma})$ rounds in the congested clique model.*

**Lower bound for Clique-Bcast.** Recall that the broadcast congested clique is a version of the congested clique model with the additional constraint that all $n - 1$ messages sent by a node in a round must be identical.

Frischknecht et al. [FHW12] have shown that approximating the diameter of an unweighted graph any better than factor $3/2$ requires $\tilde{\Omega}(n)$ rounds in the CONGEST model; the same can be applied to the CLIQUE-BCAST. A variation of the approach was recently used by Holzer and Pinsker [HP15] to show that computing any approximation better than factor 2 to all-pairs shortest paths in weighted graphs takes $\tilde{\Omega}(n)$ rounds as well. As discussed in Section 6.3.3, $\tilde{o}(n)$-round matrix multiplication algorithms imply $\tilde{o}(n)$-round
algorithms for exact unweighted and \((1 + o(1))\)-approximate weighted APSP. Together, this immediately implies that matrix multiplication on the CLIQUE-Bcast is hard.

**Corollary 6.24.** In the broadcast CLIQUE-Bcast, matrix multiplication algorithms that are applicable to matrices over the Boolean semiring and APSP algorithms require \(\tilde{\Omega}(n)\) communication rounds.

We remark that the phrase “that is applicable to matrices over the Boolean semiring” refers to the issue that, in principle, it is possible that matrix multiplication exponents may be different for different underlying semirings. However, at the very least the lower bound applies for matrix multiplication over Booleans, integers, and rationals, as well as the min-plus semiring. We stress that, unlike the lower bounds presented beforehand, this bound holds without any assumptions on the algorithm itself.

### 6.5 Conclusions

In this chapter, we demonstrate that algebraic methods – especially fast matrix multiplication – can be used to design efficient algorithms in the congested clique model, resulting in algorithms that outperform the previous combinatorial algorithms; moreover, we have certainly not exhausted the known centralized literature of algorithms based on matrix multiplication, so similar techniques should also give improvements for other problems. Likewise, it seems likely that matrix multiplication algorithms themselves can be generalized in various ways; for example, the algorithms presented in Section 6.2 can be easily adapted to multiply \(m \times m\) matrices on a clique of \(n\) nodes in \(O(m^2/n^{1+2/3})\) rounds over semirings and \(O(m^2/n^{1+2/\omega})\) rounds over rings \([\text{Tis}99, \text{McC}96]\). It also remains open whether corresponding lower bounds exist; however, it increasingly looks like lower bounds for the congested clique would imply lower bounds for centralized algorithms, and are thus significantly more difficult to prove than for the CONGEST model.

While the present work focuses on a fully connected communication topology (clique), we expect that the same techniques can be applied more generally in the usual CONGEST model. For example, fast triangle detection in the CONGEST model is trivial in those areas of the network that are sparse. Only dense areas of the network are non-trivial, and in those areas we may have enough overall bandwidth for fast matrix multiplication algorithms. On the other hand, there are non-trivial lower bounds for distance computation problems in the CONGEST model \([\text{Elk}04a, \text{PR}06, \text{DSHK}^{+}12]\), though significant gaps still remain \([\text{Nan}14]\).
Chapter 7

Approximate Proof-Labeling Schemes

7.1 Introduction

We study a new model of verification of boolean predicates over distributed networks. Given a network configuration, the proof-labeling scheme (PLS) model defines a distributed proof in the form of a label that is given to each node, and all nodes locally verify that the network configuration satisfies the desired boolean predicate by exchanging labels with their neighbors. The proof size of the scheme is defined to be the maximum size of a label.

In this work, we extend this model by defining the approximate proof-labeling scheme (APLS) model. In this new model, the predicates for verification are of the form $\psi \leq \phi$, where $\psi, \phi : \mathcal{F} \rightarrow \mathbb{N}$ for a family of configurations $\mathcal{F}$. Informally, the predicates considered in this model are a comparison between two values of the configuration. As in the PLS model, nodes exchange labels in order to locally verify the predicate, and all must accept if the network satisfies the predicate. The soundness condition is relaxed with an approximation ration $\alpha$, so that only if $\psi > \alpha \phi$ some node must reject.

We show that in the APLS model, the proof size can be much smaller than the proof size of the same predicate in the PLS model. Moreover, we prove that there is a tradeoff between the approximation ratio and the proof size.

7.1.1 Our Contribution

We introduce and formalize the concept of approximate proof-labeling schemes, and then study the complexity of verification of two fundamental problems in this model: diameter and maximum weight matching. We start by considering the verification of a specified upper bound $k$ on the network diameter $D$ (see summary of results in Table 7.1), and show that for every $k = k(n)$, the proof size of any PLS for $D \leq k$ is $\Omega(n/k)$. In the APLS model, as outlined above, we present a $3/2$-APLS for $D \leq k$ with $O(\sqrt{n \log^2 n})$.
proof size, and prove that we cannot obtain a better approximation ratio with the same asymptotic proof size. Specifically, we prove that for every \( k \) there exists an \( \epsilon \in \Theta(1/k) \) such that the proof size of any \((3/2 - \epsilon)\)-APLS for \( D \leq k \) is \( \Omega(n/\log^2 n) \). Then, we turn to show that if we increase the approximation ratio we can construct an even more efficient scheme. In particular, we show a simple 2-APLS for \( D \leq k \) with proof size \( O(\log n) \). To our knowledge, the problem of verifying an upper bound on the diameter in general graphs has not been studied before in the context of PLSs.

The second property we consider is verifying that a specified matching \( M \) has the maximum possible weight (see summary of results in Table 7.2). For this property we are interested in bounding from below the weight of the matching w.r.t. the weight of the maximum matching \( w(MWM) \). We present a 2-APLS for \( w(M) \geq w(MWM) \) with \( O(\log W) \) proof size, where \( W \) is the maximum edge-weight in the network. This improves upon a previous result presented in [KKP10], with \( O(\log n + \log W) \) proof size for a 2-approximation of the maximum weight matching on trees. We note that the notion of approximation in [KKP10] is different from our definition: they argue that there exists a subset of 2-approximate configurations that the scheme verifies, but do not promise that any configuration with an optimal matching is verified successfully.

We use various techniques to obtain our results. The lower bounds for proof complexity are achieved using reductions for nondeterministic communication complexity [GS16], a lower bound graph presented in [HPRW14] and a recent constructions of [ACK16]. The design of the APLSs is based on approximation algorithms for the diameter problem [ACIM99], and on complementary slackness conditions for primal-dual problems.
7.1.2 Related Work

Approximation algorithms were studied extensively in both sequential and distributed computing. In the sequential model, unless $P = NP$, there are no polynomial-time algorithms for NP-hard problems, and thus efficient approximation algorithms for the related optimization problems are widely studied [Vaz01]. Moreover, even for problems for which polynomial time algorithms exist, there is sometimes a need for faster algorithms that give an approximate solution.

One example is the problem of determining the diameter of a graph. While the problem is solvable in polynomial time, faster approximation algorithms are studied. A trivial 2-approximation algorithm in unweighted graphs goes through building a single BFS tree in $O(n + m)$ time, and measuring its depth. An $\tilde{O}(m\sqrt{n} + n^2)$ time 3/2-approximation algorithm for the diameter was presented in [ACIM99], and was later improved in [RW13] to $\tilde{O}(m\sqrt{n})$ time algorithms using randomization. A deterministic improvement to [ACIM99] was presented in [CLR+14]. Distributed algorithms for computing the diameter were presented in [HPRW14] and [PRT12], and both also provide approximation algorithms for the problem. Lower bounds on computing and approximating the diameter in the CONGEST model were presented in [ACK16, HW12, FHW12].

Distributed decision and verification schemes deal with verifying that a given instance satisfies some given boolean predicate. Distributed decision and verification has been formalized in various models to suit its myriad applications. These models include proof-labeling schemes (PLSs) [KKP10], locally checkable proofs (LCP) [GS16], and several complexity classes [FKP13]. The complexity classes presented in the latter include LD—local decision—which includes all properties that can be decided using a constant number of rounds and no additional information, and NLD—nondeterministic local decision—which includes all properties that can be decided in a constant number of rounds with additional information in the form of a certificate given to each node. While NLD and PLS are closely related, they differ in that NLD certificates are independent of node identifiers. Since PLS labels may depend on node identifiers, there is a PLS for every sequentially decidable property on ID based networks, while not all sequentially decidable properties are in NLD. For more details, we refer the reader to a survey of this field of research [FF16].

The concept of PLS was introduced by Korman, Kutten, and Peleg in [KKP10]. Among other results, they show a $\Theta(\log n)$ bound on the proof size of the diameter of trees, and the same bound also for the proof size of a lower bound on the diameter in general graphs. In addition, they present two $O(\log n + \log W)$ schemes to verify a maximum weight matching: one on paths, and the other is a 2-approximation of maximum weight matching on trees.

Since the introduction of proof-labeling schemes, many aspects of this mechanism were studied. Schemes where nodes may communicate to a constant distance that is greater than 1 were studied in [GS16]. For the maximum cardinality matching problem, they
show that the proof size on the family of bipartite graphs is $\Theta(1)$, and on the family of cycle graphs is $\Theta(\log n)$. For maximum weight matching, they present a scheme for the family of bipartite graphs, with $O(\log W)$ proof size, using techniques similar to the ones we use. Moreover, $[GS16]$ was the first to use nondeterministic communication complexity lower bounds in order to achieve lower bounds on the verification complexity of a PLS.

Schemes with super-constant verification time were presented in [KKM11]. Verification processes in which the global result is not restricted to be the conjunction of local outputs had been studied in [AFIM14, AFP13]. The role of unique node identifiers in local decision and verification was extensively studied in [FHS15, PHK12, FGKS13]. The use of randomization in verification process in order to reduce communication was presented in [FP15]. Proof-labeling schemes in directed networks were studied in [FLSW16], where both one-way and two-way communication over directed edges had been considered. Verification schemes for dynamic networks, where edges may appear or disappear after label assignment and before verification, were studied in [FRSW17]. Finally, a hierarchy of local decision as an interaction between a prover and a disprover was presented in [FTH16].

7.2 Model and Definitions

7.2.1 Computational Framework

A network is modeled by a connected, undirected, simple graph $G = (V, E)$, with $|V| = n$ nodes and $|E| = m$ edges. Each node represents a processor, and each edge represents a communication link. We do not assume the a processor initially knows to which other processors it is connected, but only that its communication links are enumerated by port numbers. A configuration $G_s$ is graph $G = (V, E)$ along with a state assignment function $s : V \rightarrow S$, where $S$ is called the state space. The state $s(v)$ of a node $v$ includes all local input to $v$. In particular, the state includes port numbers of adjacent edges, the node’s identity (if the network is not anonymous) or other data, e.g., the result of an algorithm. We sometimes consider weighted networks, in which the graph is accompanied with an edge weight function $w : V \rightarrow \{1, \ldots, W\}$, in which case the state of a node includes the weights of its adjacent edges.

In this work, we always assume non-anonymous networks, i.e., every node $v$ is provided with a unique identity ID($v$), which is part of the state of $v$.

7.2.2 Proof-Labeling Schemes

Given a family $\mathcal{F}$ of network configurations and a boolean predicate $\mathcal{P}$ over $\mathcal{F}$, a proof-labeling scheme (PLS) for $(\mathcal{F}, \mathcal{P})$ is a mechanism for deciding $\mathcal{P}(G_s)$ for every $G_s \in \mathcal{F}$.

\[1\] Recall that $W$ is the maximum weight of an edge in the graph. If $W = 1$, we interpret $O(\log W)$ as $O(1)$. 

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-16 - 2017
A PLS consists of two components: a prover \( p \), and a verifier \( v \). Given any legal configuration \( G_s \in \mathcal{F} \) (i.e., a configuration satisfying \( \mathcal{P} \)), the prover assigns a bit string \( \ell(v) \) to every node \( v \), called the label of \( v \). The verifier is a local distributed algorithm running concurrently at every node. At each node \( v \), it takes as input the state \( s(v) \) of \( v \), its label \( \ell(v) \) and the labels of all its neighbors, i.e., the list \( (\ell(v_1) \ldots \ell(v_d)) \), where \( d \) is the degree of \( v \), and \( v_i \) is the neighbor of \( v \) reachable from port number \( i \). The outputs of the verifier at each node is a boolean value. If the outputs are \( \text{TRUE} \) at all nodes, \( v \) is said to accept the configuration, and otherwise (i.e., \( v \) outputs \( \text{FALSE} \) in at least one node) \( v \) is said to reject the configuration. For correctness, a PLS \((\mathcal{P}, \mathcal{V})\) for \((\mathcal{F}, \mathcal{P})\) must satisfy the following requirements, for every \( G_s \in \mathcal{F} \):

- If \( \mathcal{P}(G_s) = \text{true} \) then, using the labels assigned by \( p \), the verifier \( v \) accepts \( G_s \).
- If \( \mathcal{P}(G_s) = \text{false} \) then, for every label assignment, the verifier \( v \) rejects \( G_s \).

The proof size of a PLS \((p, v)\) is the maximum length of a label assigned by the prover \( p \) on a legal configuration \( G_s \in \mathcal{F} \).

### 7.2.3 The New Model: Approximate Proof-Labeling Schemes

In this thesis we focus on predicates that represent minimization or maximization problems. Formally, we are given two functions \( \psi, \varphi : \mathcal{F} \to \mathbb{N} \), and we are interested in the predicate \( \psi(G_s) \leq \varphi(G_s) \). Note that \( \psi \) or \( \varphi \) may be constant, e.g., in verifying an upper bound on the diameter of the graph, one can be interested in verifying \( D(G_s) \leq k \). In some cases, classic verification might be too expansive, as proven in Section 7.3, and so we extend the definition of PLSs to *approximate proof-labeling schemes* (APLSs). We relax the requirements of a PLS so that a configuration for which the inequality \( \psi(G_s) \leq \varphi(G_s) \) holds is guaranteed to be accepted by the scheme, while a configuration for which \( \psi(G_s) \) much larger than \( \varphi(G_s) \) is guaranteed to be rejected. Formally, for \( \alpha \geq 1 \), an \( \alpha \)-APLS \((p, v)\) for \((\mathcal{F}, (\psi \leq \varphi))\) must satisfy the following requirements, for every \( G_s \in \mathcal{F} \):

- If \( \psi(G_s) \leq \varphi(G_s) \) then, using the labels assigned by \( p \), the verifier \( v \) accepts \( G_s \).
- If \( \psi(G_s) > \alpha \varphi(G_s) \) then, for every label assignment, the verifier \( v \) rejects \( G_s \).

The proof size of an APLS is defined similarly to that of a PLS. Our definitions naturally extend to predicates of the form \( \psi \geq \varphi \), \( \psi < \varphi \) and \( \psi > \varphi \).

Finally, we note that although the definition of an APLS might seem to resemble definitions from the field of property testing, they are inherently different. Our measure for how close a graph is to satisfy a property is entirely algebraic, and has nothing to do with changing the graph by adding or removing edges. Moreover, all schemes presented in this thesis are deterministic.

### 7.2.4 Problem Definitions

**Diameter.** Given a configuration \( G_s \) with an underlying graph \( G = (V, E) \) and an edge weight function \( w \), for every two nodes \( u, v \in V \) denote by \( \text{dist}(u, v) \) the length of the
shortest (unweighted) path between $u$ and $v$ in $G_s$, and by $\text{dist}_w(u, v)$ the minimum weight of a path between $u$ and $v$ in $G_s$. The unweighted diameter of $G_s$, denoted by $D(G_s)$, is defined as $\max \{\text{dist}(u, v) \mid u, v \in V\}$. Similarly, The weighted diameter of $G_s$, denoted by $D_w(G_s)$, is defined as $\max \{\text{dist}_w(u, v) \mid u, v \in V\}$.

The first set of problems we consider in this work are problems of bounding the weighted and unweighted diameters from above.

**Definition 7.1.** Let $F$ be the family of all weighted connected undirected configurations and let $G_s \in F$. For every integer $k = k(n)$, we define the problems $(F, (D \leq k))$ and $(F, (D_w \leq k))$.

A breadth-first search (BFS) tree in a weighted or unweighted graph $G_s$ from a root $r \in V$ is a tree consisting of a shortest (unweighted) path from $r$ to every node in $V$. If the graph is weighted, we are also interested in a shortest weighted distance tree consisting of a shortest weighted path from a root node $r$ to every node in $V$. Throughout the thesis, we use known schemes for verification of a BFS tree and a shortest weighted distance tree [KKP10]. They prove that for the verification of these trees it is enough to give every node the identity of the root and the distance from the root. Therefore, proof size is $O(\log n)$ for a BFS tree and $O(\log n + \log W)$ for a shortest weighted distance tree.

**Matchings.** Given a configuration $G_s$ with an underlying graph $G = (V, E)$, an edge weight function $w$, and an edge subset $M \subseteq E$, $M$ is a matching in $G$ if no two edges in $M$ share a node. The weight of a matching $M$, denoted by $w(M)$, is the sum of weights of all edges in $M$. We say that a matching $M$ is a maximum weight matching (MWM) if $w(M) \geq w(M')$ for every matching $M'$ in $G$.

Another problem we consider, of a different flavor, is to verify that a specified matching is a maximum weight matching.

**Definition 7.2.** Let $F_M$ be the family of all weighted connected undirected configurations with a specified matching $M$. Let $G_s \in F$ and let MWM be a maximum weight matching in $G_s$. We define the problem $(F_M, (w(M) \geq w(\text{MWM})))$.

Note that although $w(M) > w(\text{MWM})$ is not possible (since $M$ is promised to be a matching), the problem is defined to follow the structure of APLSs.

### 7.3 PLS and APLS for Diameter

Verifying that the diameter of the graph is bounded from above by a specified value can be done by a PLS with $O(n \log n)$ proof size (and $O(n(\log n + \log W))$ for weighted diameter). Simply construct a BFS tree (respectively, a shortest weighted distance tree) from every node, verify it and locally verify at each node that all of its distances are bounded by the specified value. We now show that in the PLS model, for a constant bound $k$, the proof
Figure 7.1: The diameter lower bound construction for $s = 6$. Here, $x = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$ and $y = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$, where the matrix rows are indexed by $\{1, 2, 3\}$ and the columns by $\{4, 5, 6\}$. Since $x_{16} = y_{16} = 1$, the dotted edges are missing and the distance between $a_1$ and $b_6$ is greater than $k$.

size cannot be improved by more than a $\Theta(\log n)$ factor, i.e., it must have an $\Omega(n)$ proof size. Moreover, for every $k = k(n)$, we show a lower bound for the PLS proof size.

Let $s = \frac{n}{k} - 1$, and assume $s$ is an even integer. Consider the following graph family $\{G_{x,y}\}$ over the following set of $n$ nodes. Let $A_1 = a_1, ..., a_{s/2}$, $A_2 = a_{s/2+1}, ..., a_s$, $B_1 = b_1, ..., b_{s/2}$, and $B_2 = b_{s/2+1}, ..., b_s$ be four cliques, where each $a_i$ is connected to $b_i$ with a path of length $k - 1$, consisting of $a_i$, $b_i$, and $k - 2$ new nodes unique to this path. An additional node $a$ is connected to every $a_i$ by an edge, an additional node $b$ is connected to every $b_i$ by an edge, and there is a $(k - 1)$-node path connecting $a$ and $b$ with another new $k - 2$ nodes. Given an instance $(x, y)$ of the Set-Disjointness problem over $(s/2)^2$ elements, enumerate Alice’s input as $x_{ij}$ with $i \in \{1, \ldots, s/2\}$ and $j \in \{s/2 + 1, \ldots, s\}$, and similarly for Bob’s input, $y_{ij}$. To complete the construction of $G_{x,y}$, add an edge $(a_i, a_j)$ if and only if $x_{ij} = 0$, and we add an edge $(b_i, b_j)$ if and only if $y_{ij} = 0$.

If $\frac{n}{k} - 1$ is not an even integer, we choose $s$ to be the largest even integer such that $s < \frac{n}{k} - 1$, add nodes to described construction to complement the number of nodes to $n$, and connect all additional nodes to all neighbors of $b$.

**Lemma 7.3.** $D(G_{x,y}) \leq k$ if and only if $\text{DISJ}(x, y) = \text{true}$.

**Proof.** If $\text{DISJ}(x, y) = \text{true}$, then for each $\{i, j\}$, at least one of the edges $(a_i, a_j)$ or $(b_i, b_j)$ exists in $G_{x,y}$. Let $u$ and $v$ be any two nodes in $G_{x,y}$. Suppose that $u$ is on the path $(a_i \leadsto b_i)$ and $v$ is on the path $(a_j \leadsto b_j)$, where $i, j \in \{1, \ldots, s/2\}$. If $i = j$, clearly, $\text{dist}(u, v) \leq k - 1$. If $i \neq j$, then the path from $u$ to $a_i$ consists of at most $k - 1$ edges, and the path from $a_i$ to $b_i$ consists of at most $k - 1$ edges, and similarly for $a_j$ to $b_j$.
Otherwise, by assumption, either the cycle \((a \rightarrow a_i \sim b_i \rightarrow b_j \sim a_j \rightarrow a)\) or the cycle \((b \rightarrow b_j \sim a_j \rightarrow a_i \sim b_i \rightarrow b)\) exists and its length is \(2k + 1\). Hence, every two nodes in the cycle are at distance at most \(k\) from each other, and \(\text{dist}(u, v) \leq k\). Suppose now that either \(u\) or \(v\) is on the path \((a \sim b)\) and the other node is on the path \((a_i \sim b_i), i \in \{1, \ldots, s\}\). The length of the cycle \((a \rightarrow a_i \sim b_i \rightarrow b \sim a)\) is \(2k\), and since \(u\) and \(v\) are on this cycle, \(\text{dist}(u, v) \leq k\). Finally, if both \(u\) and \(v\) are on the path \((a \sim b)\), clearly, \(\text{dist}(u, v) \leq k - 1\), and we conclude that \(D(G_{x,y}) \leq k\).

If \(\text{DISJ}(x, y) = \text{false}\), then there exist \(i \in \{1, \ldots, s/2\}\) and \(j \in \{s/2 + 1, \ldots, s\}\) such that \(x_{ij} = y_{ij} = 1\), and by the construction of \(G_{x,y}\), both edges \((a_i, a_j)\) and \((b_i, b_j)\) are absent. Every path from \(a_i\) to \(b_j\) must go through some \((a' \sim b')\) path of length \(k - 1\), and if \(\text{dist}(a_i, b_j) \leq k\) then the shortest path connecting \(a_i\) and \(b_j\) can only contain one more edge. However, since the edges \((a_i, a_j)\) and \((b_i, b_j)\) are both absent in \(G_{x,y}\), no such path exists, so \(\text{dist}(a_i, b_j) > k\), which implies that \(D(G_{x,y}) > k\).

**Theorem 7.4.** For every \(k\), the proof size of any PLS for \((\mathcal{F}, (D \leq k))\) is \(\Omega(n/k)\).

**Proof.** Consider any PLS for \((\mathcal{F}, (D \leq k))\), and construct a nondeterministic protocol for \(\text{DISJ}(x, y)\) as follows. Alice and Bob simulate the verification of \(D(G_{x,y}) \leq k\) using the PLS, such that Alice simulates the nodes in \(A = A_1 \cup A_2 \cup a\), and Bob simulates the rest of the nodes, denoted by \(B\). Each of the players nondeterministically chooses the labels of its nodes as his auxiliary bit-string. Alice and Bob then exchange the labels corresponding to the nodes touching the cut, and simulate the verification process in all nodes. Then, they compute \(a\) and \(b\), the conjunction of the returned values of \(A\) and \(B\) respectively. Finally, Alice sends \(a\) to Bob, Bob sends \(b\) to Alice, and they both output the conjunction \(a \land b\) as the solution for \(\text{DISJ}(x, y)\).

If \(\text{DISJ}(x, y) = \text{true}\) then \(D(G_{x,y}) \leq k\), there is an assignment of labels to the nodes such that all nodes output \(\text{true}\), and if both players choose these labels as their bit-strings then they both output \(\text{DISJ}(x, y) = \text{true}\). On the other hand, if \(\text{DISJ}(x, y) = \text{false}\) then \(D(G_{x,y}) > k\), for every assignment of labels to the nodes at least one node outputs \(\text{false}\), and Alice and Bob output \(\text{DISJ}(x, y) = \text{false}\) in all executions.

Thus, the simulation we presented is a nondeterministic protocol for deciding \(\text{DISJ}(x, y)\). We know that in any nondeterministic protocol for Set-Disjointness with \((s/2)^2\) elements, Alice and Bob must exchange \(\Omega((s/2)^2)\) bits. The number of edges in the cut of \(G_{x,y}\) induced by the partition of the nodes between Alice and Bob in the simulation is \(s + 1\). Therefore, the proof size of any PLS for \((\mathcal{F}, (D \leq k))\) is \(\Omega(s) \in \Omega(n/k)\).

We now show that in the APLS model there are schemes with much smaller proof size. We start with a 3/2-APLS and construct a scheme that is based on the randomized algorithm for a 3/2-approximation of the diameter presented in [RW13]. We use the following two lemmas.

**Lemma 7.5.** Let \(G = (V, E)\) be a graph, let \(S, N \subseteq V\) be two sets of nodes, and consider a node \(w \in V\). Assume that \(N\) is the set of \(z\) nodes closest to \(w\) for some parameter \(z\), \(w\)
The theorem is the farthest node from the set \( S \), and \( N \cap S \) is non-empty. Then, the largest depth \( D' \) of a BFS tree rooted at a node in \( R = N \cup S \cup \{w\} \) satisfies \( \frac{2}{3} D \leq D' \leq D \).

**Lemma 7.6.** Let \( G = (V, E) \) be a graph and \( z \in \mathbb{N} \) a parameter. For each \( v \in V \), let \( N_z(v) \) be the set of \( z \) nodes closest to \( v \). Then, there exists a hitting set \( \{N_z(v) \mid v \in V\} \) of size \( O(n \log n/z) \).

Lemma 7.5 corresponds to an adapted version of Lemma 4 of [RW13], and Lemma 7.6 is a corollary of Theorem 2.7 of [ACIM99]. We obtain the following result.

**Theorem 7.7.** There exists a 3/2-APLS for \((\mathcal{F}, (D \leq k))\) with proof size \( O(\sqrt{n} \log^2 n) \).

**Proof.** Our scheme is based on Lemma 7.5; it consists of a node \( w \), sets \( N \) and \( S \) and all the BFS trees rooted at \( R = N \cup S \cup \{w\} \). In addition, there is a node \( w' \) that is used to verify that the largest depth of a BFS tree rooted in \( R \) is as claimed, and a BFS tree rooted at \( w' \). The main task in our scheme is to verify the BFS trees described above, and to verify that the diameter estimation, i.e., the maximum depth of the trees, is at most \( k \).

Since a BFS tree verification is known from previous work, the challenges in the scheme construction is to verify locally that \( w \) is indeed the farthest node from the set \( S \), that \( N \) is the neighborhood of \( w \), and that the estimation is indeed the maximum depth of a tree.

Formally, let \( G_s \in \mathcal{F} \) be a configuration with the underlying graph \( G = (V, E) \) and \( D(G_s) \leq k \). For every \( v \in V \), denote by \( N_{\sqrt{n}}(v) \) the \( \sqrt{n} \) nodes closest to \( v \) (break ties according to IDs), and let \( S \subset V \) be a set of \( O(\sqrt{n} \log n) \) nodes such that \( S \) hits \( \{N_{\sqrt{n}}(v) \mid v \in V\} \), whose existence follows from Lemma 7.6.

Let \( h(v) = \min \{\text{dist}(v, u) \mid u \in S\} \), the distance of \( v \) from the set \( S \), and let \( w \) be the farthest node from \( S \), i.e., \( h(w) \geq h(v) \) for every \( v \in V \). Let \( q(w) \) be the largest distance from \( w \) to any node in \( N_{\sqrt{n}}(w) \). Let \( R = S \cup \{w\} \cup N_{\sqrt{n}}(w) \) be a set of \( |R| = O(\sqrt{n} \log n) \) nodes, and consider the set \( R_{BFS} \) of BFS trees rooted at nodes in \( R \). Let \( d_{\text{max}} \) be the maximum depth of a tree in \( R_{BFS} \) and let \( w' \) be a node at distance \( d_{\text{max}} \) from one of the roots.

The label assigned to a node \( v \in V \) is

\[
\ell(v) = (\ell_{\text{BFS}_{-S}}(v), \ell_{\text{BFS}_{-N}}(v), \ell_{\text{BFS}_{-w}}(v), \ell_{\text{dist}}(v), \ell_{\text{ID}}(v), \ell_{\text{dist}}(v))
\]

where \( \ell_{\text{BFS}_{-S}}(v) \) is a set of \( O(\sqrt{n} \log n) \) pairs \( \{(\text{ID}(u), \text{dist}(v, u)) \mid u \in S\} \); \( \ell_{\text{BFS}_{-N}}(v) \) is a set of \( \sqrt{n} \) pairs \( \{(\text{ID}(u), \text{dist}(v, w)) \mid u \in N_{\sqrt{n}}(w)\} \); \( \ell_{\text{BFS}_{-w}}(v) = (\text{ID}(w), \text{dist}(v, w)) \); and \( \ell_{\text{BFS}_{-w'}} = (\text{ID}(w'), \text{dist}(v, w')) \). Every pair mentioned above is the label needed in order to verify the correct structure of the corresponding BFS tree. In order to verify that \( w \) is indeed the farthest node from \( S \), every node is given the distance of \( w \) from \( S \), \( \ell_{w}(w) = h(w) \); To verify the consistency of \( N_{\sqrt{n}}(w) \), every node is given the radius of this neighborhood \( \ell_{q}(w) = q(w) \); and \( \ell_{\text{max-dist}} = d_{\text{max}} \) is given in order to verify the existence and maximality of the estimation \( d_{\text{max}} \).

In the verification process, a node \( v \) exchanges labels with all its neighbors, and verifies the following conditions:
1. Consistency of global parameters: For every neighbor $v'$ of $v$, it holds that $\ell_{hw}(v') = \ell_{hw}(v)$, $\ell_{qw}(v') = \ell_{qw}(v)$, and $\ell_{\max-dist}(v') = \ell_{\max-dist}(v)$.

2. All distances are bounded by $d_{\max}$ and $k$: For every pair $(ID, d)$ in $\ell_{\text{BFSs-}S(v)} \cup \ell_{\text{BFSs-}N(v)} \cup \{\ell_{\text{BFS-w}}(v)\} \cup \{\ell_{\text{BFS-w'}}(v)\}$, it holds that $0 \leq d \leq \ell_{\max-dist}(v) \leq k$.

3. Existence of a BFS tree of depth $d_{\max}$: If $\ell_{\text{BFS-w'}}(v) = (ID(v), 0)$ then there exists a pair $(ID, d) \in \ell_{\text{BFSs-}S(v)} \cup \ell_{\text{BFSs-}N(v)} \cup \{\ell_{\text{BFS-w}}(v)\}$ such that $d = \ell_{\max-dist}(v)$.

4. Only one pair for each node in $S$ and in $N_{\sqrt{k}}(w)$: For every two pairs $(ID, d), (ID', d') \in \ell_{\text{BFSs-}X(v)}$, for $X \in \{S, N\}$, if $d \neq d'$ then ID $\neq$ ID'.

5. BFS structures: For every neighbor $v'$ of $v$ and $X \in \{S, N\}$, the following holds. There exists a pair $(ID, d) \in \ell_{\text{BFSs-}X(v)}$, for some $d$ if and only if there exists a pair $(ID, d') \in \ell_{\text{BFSs-}X(v')}$ with the same ID and $d' \in \{d - 1, d, d + 1\}$. For $x \in \{w, w'\}$, $\ell_{\text{BFS-}x}(v) = (ID, d)$ for some $d$ if and only if $\ell_{\text{BFS-}x}(v') = (ID, d')$ for $d' \in \{d - 1, d, d + 1\}$.

6. Existence of roots: For every $X \in \{S, N\}$ and pair $(ID, d) \in \ell_{\text{BFSs-}X(v)}$, if $d > 0$ then there exists a neighbor $v'$ of $v$ with $(ID, d - 1) \in \ell_{\text{BFSs-}X(v')}$. For $x \in \{w, w'\}$, if $\ell_{\text{BFS-}x}(v) = (ID, d)$ and $d > 0$ then there exists a neighbor $v'$ of $v$ with $\ell_{\text{BFS-}x}(v') = (ID, d - 1)$.

7. Unique roots: For every pair $(ID, d)$ in $\ell_{\text{BFSs-}S(v)} \cup \ell_{\text{BFSs-}N(v)} \cup \{\ell_{\text{BFS-w}}(v)\} \cup \{\ell_{\text{BFS-w'}}(v)\}$, if $d = 0$ then ID $=$ ID($v$).

8. Non-empty intersection of $S$ and $N_{\sqrt{k}}(w)$: There exists a pair $(ID, d) \in \ell_{\text{BFSs-}S(v)} \cap \ell_{\text{BFSs-}N(v)}$.

9. Maximality and correctness of $h(w)$: There exists a pair $(ID, d) \in \ell_{\text{BFSs-}S(v)}$ such that $d \leq \ell_{hw}(v)$, and if $\ell_{\text{BFS-w}}(v) = (ID(v), 0)$ then there exists no pair $(ID, d) \in \ell_{\text{BFSs-}S(v)}$ such that $d < \ell_{hw}(v)$.

10. The neighborhood of $w$: Let $\ell_{\text{BFS-w}}(v) = (ID, d)$. If $d < \ell_{qw}(v)$ then there exists a pair $(ID(v), 0) \in \ell_{\text{BFSs-}N(v)}$, and if $d > \ell_{qw}(v)$ then there exists no pair $(ID(v), 0) \in \ell_{\text{BFSs-}N(v)}$.

The completeness of this 3/2-APLS follows from the fact that if $D(G_s) \leq k$ then the maximum depth of any BFS tree in $G_s$ is at most $k$.

For the soundness, consider a configuration $G_s \in \mathcal{F}$ with the underlying graph $G = (V, E)$ and label assignment $\ell$, and assume that all nodes output true. By (4), all nodes have the same values $\ell_{hw}$, $\ell_{qw}$ and $\ell_{\max-dist}$. By (1), (3), (6) and (7) for every node $v \in V$ and every pair $(ID, d) \in \ell_{\text{BFSs-}S(v)} \cup \ell_{\text{BFSs-}N(v)} \cup \{\ell_{\text{BFS-w}}(v)\} \cup \{\ell_{\text{BFS-w'}}(v)\}$, there exists a node $u$ such that ID $=$ ID($u$) and it holds that $d = \text{dist}(v, u)$.  

120
Let \( S(v) \) be the collection of IDs in \( \ell_{BFS-s}(v) \), let \( \overline{N}(v) \) be the collection of IDs in \( \ell_{BFS-N}(v) \), let \( \overline{w}(v) \) be the ID in \( \ell_{BFS-w}(v) \) and let \( \overline{w}'(v) \) be the ID in \( \ell_{BFS-w'}(v) \). By (5), for every two nodes \( v \) and \( u \) it holds that \( \overline{S}(v) = \overline{S}(u) \), \( \overline{N}(v) = \overline{N}(u) \), \( \overline{w}(v) = \overline{w}(u) \) and \( \overline{w}'(v) = \overline{w}'(u) \). We denote these values by \( \overline{S} \), \( \overline{N} \), \( \overline{w} \) and \( \overline{w}' \) respectively. By (10), \( \overline{N} \) is the set of closest nodes to \( \overline{w} \); by (9), \( \overline{w} \) is the farthest node from the set \( \overline{S} \); and by (8), there exists some node in the intersection of \( \overline{N} \) and \( \overline{S} \). By (3), the collection of pairs \( \ell_{BFS-w'}(v) \) of all nodes \( v \in V \) indicates a BFS rooted at \( \overline{w} \) with distance \( \ell_{max-dist} \) to one of the nodes in \( \overline{S} \cup \overline{N} \cup \{ \overline{w} \} \), and by (2) we know that this is the largest distance from any node to one of the nodes in \( \overline{S} \cup \overline{N} \cup \{ \overline{w} \} \) and this distance is at most \( k \).

Overall, we have a collection of BFS trees with depth at most \( \ell_{max-dist} \leq k \). Therefore, all conditions of Lemma 7.5 are satisfied, and we have \( (2/3)D(G_s) \leq \ell_{max-dist} \). Hence, \( D(G_s) \leq (3/2)k \) as desired.

The proof size of the scheme follows from Lemma 7.6, which implies that there exists a set \( S \) of size \( O(\sqrt{n} \log n) \) that is a hitting set for \( \{ N_{\sqrt{n}}(v) \mid v \in V \} \). In particular, the intersection \( N_{\sqrt{n}}(w) \cap S \), where \( w \) is the farthest node from \( S \), is not empty. Therefore, the label consists of \( O(\sqrt{n} \log n) \) sub-labels of size \( O(\log n) \) each. \( \square \)

The following result shows that with the proof size we obtain for \( 3/2 \)-APLS we cannot have a better approximation ratio that is correct for all possible bounds \( k \). To get a better approximation ratio, one needs to use labels that are almost as large as the labels used for exact PLS.

Let \( x \) and \( y \) be two \( s \)-bit strings, \( s \in \Omega(n/\log n) \). Our lower bound follows the recent construction of Abboud et al. [ACK16].

**Lemma 7.8 ([ACK16])**. Given two strings \( x,y \in \{0,1\}^s \), there exists a graph \( G_{x,y} = (V,E) \) and a partition of \( V \) into \( V_A \) and \( V_B \) such that:

1. The number of nodes in \( G_{x,y} \) is \( n \in \Theta(s \log s) \).
2. All the edges depending on \( x \) are between nodes in \( V_A \).
3. All the edges depending on \( y \) are between nodes in \( V_B \).
4. The number of edges between nodes in \( V_A \) and \( V_B \) is in \( \Theta(\log s) \).
5. If \( \text{DISJ}(x,y) = \text{true} \) then \( D(G_{x,y}) \leq k \), and otherwise \( D(G_{x,y}) > 3k/2 - 9 \).

From this construction we derive the following lower bound.

**Theorem 7.9.** For every \( k \), there exists an \( \epsilon \in \Theta(1/k) \) such that the proof size of any \( (3/2 - \epsilon) \)-APLS for \( (\mathcal{F}, (D \leq k)) \) is \( \Omega(n/\log^2 n) \).

**Proof.** Consider a \( (3/2 - 9/k) \)-APLS for \( (\mathcal{F}, (D \leq k)) \), and an instance \( (x, y) \) of the DISJ problem over \( s \) bits. Construct the graph \( G_{x,y} \) as in Lemma 7.8 with the same partition to \( V_A \) and \( V_B \). Alice and Bob nondeterministically choose the labels for the nodes of \( V_A \) and \( V_B \), simulate the verification algorithm together, and then compute \( a \) and \( b \), the

\[ \text{See Chapter 2.2 of [ACK16]. We use } P = [(k - 2)/4]. \]
conjunction of the returned values of \( V_A \) and \( V_B \). Finally, Alice sends \( a \) to Bob, Bob sends \( b \) to Alice, and they both output the conjunction \( a \land b \) as the solution for DISJ\((x,y)\).

By Lemma 7.8, if \( \text{DISJ}(x,y) = \text{true} \) then \( D \leq k \), all nodes must accept and Alice and Bob return \text{true}. On the other hand, if \( \text{DISJ}(x,y) = \text{false} \) then \( D > (3/2 - 9/k)k \), at least one node rejects, and Alice and Bob return \text{false}. Thus, Alice and Bob correctly solve the Set-Disjointness problem over \( s \) elements.

Note that \( \log n = \Theta(\log s) \). Alice and Bob must communicate \( \Omega(s) = \Omega(n/\log n) \) bits, and there are \( O(\log n) \) nodes touching the cut, so the proof size is \( \Omega(n/\log^2 n) \).

To further study the tradeoff between the approximation ratio and the proof size, we now prove that if we increase the approximation ratio we can construct an even more efficient scheme.

**Theorem 7.10.** There exists a 2-APLS for \((\mathcal{F}, (D_w \leq k))\) with proof size \( O(\log n + \log W) \).

**Proof.** Let \( G_s \in \mathcal{F} \) such that \( D_w(G_s) \leq k \), and let \( r \in V \) be some node. The label assigned to every node \( v \in V \) is \( \ell(v) = (\ell_{\text{dist}}(v), \ell_{\text{root}}(v)) \), where \( \ell_{\text{dist}}(v) = \text{dist}_w(r,v) \) and \( \ell_{\text{root}}(v) = \text{ID}(r) \). To verify that \( D_w(G_s) \leq k \), a node \( v \) exchanges labels with all its neighbors, and verifies the following conditions:

1. For every neighbor \( u \) of \( v \), it holds that \( \ell_{\text{root}}(u) = \ell_{\text{root}}(v) \).

2. \( 0 \leq \ell_{\text{dist}}(v) \leq k \).

3. If \( \ell_{\text{dist}}(v) > 0 \) then \( v \) has at least one neighbor \( u \) with \( \ell_{\text{dist}}(u) = \ell_{\text{dist}}(v) - w(u,v) \).

4. If \( \ell_{\text{dist}}(v) = 0 \) then \( \ell_{\text{root}}(v) = \text{ID}(v) \).

The completeness of this 2-APLS is clear: If \( D_w(G_s) \leq k \) and labels are assigned as described above, all nodes output \text{true}.

For the soundness, consider a configuration \( G_s \) with label assignment \( \ell \), such that all nodes output \text{true}. For a node \( v \) in the graph, follow the path from \( v \) constructed by repeatedly going from a node \( v' \) to its neighbor \( u \) with \( \ell_{\text{dist}}(u) = \ell_{\text{dist}}(v') - w(u,v') \), whose existence is guaranteed by Condition (3). By conditions (2) and (3), this path must end after traversing a weight of at most \( k \), at a node \( r \) with \( \ell_{\text{dist}}(r) = 0 \), and this node is unique by conditions (1) and (4). As this claim can be applied to each node in the graph, every two nodes in the graph are connected to each other by a path through \( r \), of weighted distance at most \( 2k \), and \( D_w(G_s) \leq 2k \) as desired. □

The following corollary is a direct result of Theorem 7.10 for the unweighted case.

**Corollary 7.11.** There exists a 2-APLS for \((\mathcal{F}, (D \leq k))\) with proof size \( O(\log n) \).
7.4 Maximum Weight Matching

Given a configuration \( G_s \in \mathcal{F}_M \) with the underlying graph \( G = (V, E) \), an edge weight function \( w \), and a specified matching \( M \subset E \), we wish to verify \( (\mathcal{F}_M, (w(M) \geq w(MWM))) \). Göös and Suomela \cite{GS16} present a PLS for this problem in bipartite graphs, using a linear programming (LP) formulation. Here, we extend their technique to present a 2-APLS for \( (\mathcal{F}_M, (w(M) \geq w(MWM))) \) on general graphs.

Our 2-APLS is simple: the label of a matched node is the weight of its matched edge, and the label of an unmatched node is 0. The verification process, and the proof that this is indeed a 2-APLS are slightly more involved, and use a relaxation of the complementary slackness conditions of a relaxation of a linear-programming formulation for the problem.

Consider the next integral-LP formulation of the MWM problem (cf. \cite{CCPS98}, Chapter 5):

Maximize \( \sum_{e \in E} w(e)x_e \)
Subject to \( \sum_{\{e \mid v \in e\}} x_e \leq 1, \forall v \in V \)
\( x_e \in \{0, 1\}, \forall e \in E, \)

and the LP obtained by relaxing the integrality condition into:

\( x_e \geq 0, \forall e \in E. \)

The dual linear-program of the relaxed problem is

Minimize \( \sum_{v \in V} y_v \)
Subject to \( y_u + y_v \geq w(e), \forall e = (u, v) \in E. \)

Given a pair consisting of a primal and a dual feasible solutions, their optimality can be verified by checking several conditions derived from the LP, conditions that are known as the complementary slackness conditions. For the aforementioned LP, the conditions are:

\[ x_e > 0 \implies y_u + y_v = w(e), \quad e = (u, v) \in E; \quad \text{and} \]
\[ y_v > 0 \implies \sum_{\{e \mid v \in e\}} x_e = 1, \quad v \in V. \]

If \( G \) is bipartite, then any pair of feasible optimal solutions satisfy the complementary slackness conditions, a fact that lies at the heart of the PLS presented by Göös and Suomela \cite{GS16}.

For general graphs, the same method fails miserably. The inherent obstacle that this approach faces is the integrality gap of the LP formulation: a fractional solution to the problem may be twice as large as the maximum integral solution. While there are LP formulations of the problem with an integrality gap of 1, it is not clear how to translate them into a PLS, since the number of dual variables in these LPs is substantially larger.

However, we observe that a relaxed version of these conditions is enough to prove that a primal solution is an approximation of the MWM.
Theorem 7.12 (See [Vaz01, Section 15.1]). If $x$ and $y$ are feasible primal and dual solutions in a graph $G$ satisfying

\[
x_e > 0 \implies w(e) \leq y_u + y_v \leq 2w(e), \quad e = (u, v) \in E; \text{ and}
\]
\[
y_v > 0 \implies \sum_{\{e|\in e\}} x_e = 1, \quad v \in V,
\]

then $x$ is a 2-approximation of the MWM in $G$.

Unlike the case of bipartite graphs, here the opposite implication does not hold: not every pair of 2-approximate solutions fulfill the conditions. Thus, given a matching represented by a vector $x$, we explicitly build a dual solution $y$ such that $x$ and $y$ satisfy above conditions. This dual solution $y$ will serve as a 2-APLS for $(F_M, (w(M) \geq w(MWM)))$ in a general graph.

Theorem 7.13. There exists a 2-APLS for $(F_M, (w(M) \geq w(MWM)))$ with proof size $O(\log W)$.

Proof. Let $G$ be a weighted graph with weights in $\{1, \ldots, W\}$ and $M$ a maximum weight matching in $G$. Let $(x_e)_{e \in E}$ be the indicator vector of $M$. Define the values of the dual variables $(y_v)_{v \in V}$ by $y_v = w(e)$ if there exist an edge $e \in M$ such that $v \in e$, and $y_v = 0$ otherwise. The label of a node $v$ is set to be $y_v$.

To verify $(F_M, (w(M) \geq w(MWM)))$, a node $v$ exchanges labels with its neighbors and check the next feasibility condition:

- For each neighbor $u$ of $v$, $y_u + y_v \geq w(u, v)$.

We start by showing that if $M$ is indeed a MWM, then the relaxed complementary slackness conditions hold. Let $e = (u, v)$ be an edge satisfying $x_e > 0$, i.e. $e \in M$, then $y_u = y_v = w(e)$ and indeed $w(e) \leq y_u + y_v \leq 2w(e)$. For the second complementary slackness condition, let $v$ be a node with $y_v > 0$, so there is exactly one edge $(u, v) \in M$ with $x_{(u,v)} = 1$, while for every other neighbor $u'$ of $v$, $x_{(u',v)} = 0$, so \( \sum_{\{e|\in e\}} x_e = 1 \).

For the feasibility, the input is a feasible matching, so $\sum_{\{e|\in e\}} x_e \leq 1$ for each node $v$ and $x_e \geq 0$ for each edge $e$, and the primal solution $x$ is feasible. For the dual solution $y$, assume towards contradiction that there is an edge $e = (u, v), e \notin M$, such that $y_u + y_v < w(e)$. Then, the matching obtained by removing any edge in $M$ that touches $u$ or $v$ and adding $e$ to $M$ has a weight $w(M) - (y_u + y_v) + w(e) > w(M)$, which contradicts the maximality of $M$. The case of $e \in M$ was considered in the previous paragraph. Thus, we have a pair of feasible primal and dual solutions satisfying the relaxed slackness conditions, and the solutions are 2-approximations of the optimal solutions.

Finally, consider a configuration $G_s$ with label assignment $(x_e)$, such that all nodes output true. The labels represent a dual solution that satisfies all the relaxed complementary slackness conditions, so by Theorem 7.12 the solution is a 2-approximation of the MWM.

We are unaware of any lower bound for the MWM problem in the PLS model, nor in the CONGEST and LOCAL models. We note that for every approximation ratio
\( \alpha \geq 1 \), some communication is needed in any \( \alpha \)-APLS for \((\mathcal{F}_M, (w(M) \geq w(MWM)))\). This is true since, for every configuration \( G_s \) with an empty matching \( M = \emptyset \) (not any approximation of MWM), the local view of every node is consistent with some legal configuration with matching \( M' \), where \( w(M') = w(MWM) \). Let \( v \) be a node and let \( u_1, \ldots, u_d \) be the neighbors of \( v \) where the weight of every edge \((v, u_i)\) is \( w_i \). The construction of the legal configuration \( G^v_s \) for \( v \) is as follows. Add nodes \( z_1, \ldots, z_d \) and an edge \( e_i = (z_i, u_i) \) of weight \( w_i + 1 \) for every \( 1 \leq i \leq d \). Finally, define \( M' = \{e_i \mid 1 \leq i \leq d\} \). It is easy to verify that there is no augmenting path for \( M' \) in this configuration, i.e., \( w(M') = w(MWM) \). However, the local view of \( v \) in \( G_s \) and in \( G^v_s \) is the same. Therefore, without communication, \( v \) must output \( \text{true} \). Since the same holds for every node, we conclude that some communication is necessary, regardless of the desired approximation ratio.

### 7.5 Discussion

This chapter presents the new model of approximate proof-labeling schemes. We illustrate the power of the APLS model with the \( D \leq k \) predicate. We prove a tight lower bound (up to a logarithmic factor) in the PLS model, and present two, more efficient, APLSs for this predicate. The two APLSs show a non-trivial tradeoff between the approximation ratio and the proof size.

We also present a 2-APLS for the predicate \( w(M) \geq w(MWM) \) on general graphs, a problem for which it is unknown if a non-trivial PLS exists. Presenting an efficient PLS for this problem, showing that a PLS with small proof size does not exist, or presenting an APLS with different approximation ratio or different proof size are interesting questions left open.

It would be interesting to study the APLS model on other graph predicates. For example, the chromatic number \( \chi(G) \) of a graph \( G \) is the minimal number of colors in a proper node coloring of \( G \). A PLS for \( \chi \leq k \) with proof size \( O(\log k) \) exists, where the proof is a proper coloring of the graph. However, it was proven in [GS16] that any PLS for \( \chi > 3 \) must have \( \Omega(n^2) \) proof size. Hence, also for this problem, the APLS model may allow a more efficient verification.

Finally, the idea of approximation in verification we present in this thesis can be extended to other decision and verification schemes, such as the complexity classes LD and NLD, generating a different classification of problems. For example, our 2-APLS for \( w(M) \geq w(MWM) \) on general graphs can also be used for 2-approximate NLD, under the relevant definitions, since the labels can be locally computed by the nodes.
Chapter 8

A $\left(2 + \epsilon\right)$-Approximation for
Maximum Weight Matching in the
Semi-Streaming Model

8.1 Introduction

While our main venue of research is distributed graph algorithms, this chapter considers
a different model for graph problems: the semi-streaming model. This model also
requires algorithms to make decisions under uncertainty regarding the input, but here the
uncertainty comes from memory restrictions and not from communication limitations.

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 algorithms
used clever ideas which were specifically crafted for the problem and model.

The Local-Ratio Technique  A simple greedy algorithm gives a 2-approximation for
MWM in the unweighted case. In the weighted case, a 2-approximation can be achieved
by first sorting the edges and then adding them greedily, from the heaviest to the lightest.
The local-ratio technique enables us to solve the weighted case without initially sorting
the edges; instead it enables us to ignore some of the edges, while processing the other
edges in an arbitrary order. In this work, we extend the local-ratio technique, in a way
that allows us to discard all but $O(n \log n)$ of the edges.

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

Technion - Computer Science Department - Ph.D. Thesis  PHD-2017-16 - 2017
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.

**Using the Local-Ratio Technique in the Semi-Streaming Model**  
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 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. Ghaffari [Gha17] has recently improved the analysis of our algorithm, and showed a slight modification to the algorithm, which achieves the optimal \(O(n \log n)\)-space bound.

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.
8.1.1 Related Work

The study of graph algorithms in the semi-streaming model was initiated by Feigenbaum et al. \cite{FKM05}, 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 \cite{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 \cite{Zel12} achieves a 5.585-approximation algorithm.

A different approach was taken by Epstein et al. \cite{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 \cite{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. \cite{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 \cite{CS14} prove this technique cannot give an approximation ratio better than 3.5. To circumvent this bound, we use a different approximation framework, the local-ratio technique. To the best of our knowledge, this is the first use of this technique in the streaming environment.

A related problem is estimating size of the maximum matching in a graph \cite{AKL17, Kap13, KKS14, GKK12} which is known to be related to matrix rank approximation. More general submodular-function matching problems in the semi-streaming model have been considered by Varadaraja and by Chakrabarti and Kale \cite{Var11, CK14}.

The problem of MWM was also considered in other streaming models, such as the MapReduce model \cite{CS14, LMSV11}, the sliding-window model \cite{CS14, CMS13} and the turnstile stream model (allowing deletions as well as insertions) \cite{Kon15, AKLY16, BS15, CCE16}. Extending our technique to other computational models is a challenge yet to be addressed.

Structure of this chapter We formally define the MWM problem and the semi-streaming model of computation in Section 8.2. In Section 8.3, we introduce the local-ratio theorem, present a sequential 2-approximation local-ratio algorithm for MWM in the
sequential model, and discuss our extensions to the theorem. In Section 8.4 we extend the 2-approximation algorithm to a more involved \((2 + \epsilon)\)-approximation algorithm for MWM and analyze its performance, and finally adapt the last algorithm to the semi-streaming model.

8.2 Preliminaries

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 \( N(e) = \{e' \mid |e \cap e'| = 1\} \), and \( N^+(e) = N(e) \cup \{e\} \). We usually assume edge weights can be represented by \( O(\log n) \) bits, and discuss greater weights at the end of the chapter.

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 constrains 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.

The Semi-Streaming Model  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.
8.3 Approximating Maximum Weight Matching

In this section we present the local-ratio theorem for maximization problems [BBFR04], and use it to present a sequential 2-approximation algorithm for MWM. We then present extensions of this technique and use them in order to adjust the sequential local-ratio algorithm to the semi-streaming model, incurring only a small loss in the approximation ratio.

8.3.1 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 reduced graph with weight vector $w_r$ and the residual graph with weight vector $\bar{w}_r$, such that $w = w_r + \bar{w}_r$. We start with the local-ratio theorem for maximization problems [BBFR04], Theorem 9], which we restate here for completeness. Note that this theorem applies even if $w_r$ takes negative values.

**Theorem 8.1.** Let $w \in \mathbb{R}^m$ be a vector, and consider the problem of maximizing the product $xw$ under a set of feasibility constraints. Let $w_r, \bar{w}_r \in \mathbb{R}^m$ be vectors such that $w = w_r + \bar{w}_r$. Let $x \in \mathbb{R}^m$ be a feasible solution that is a $p$-approximation with respect to $w_r$ and with respect to $\bar{w}_r$. Then $x$ is a $p$-approximation with respect to $w$ as well.

**Proof.** Let $x^*, x^+_r$ and $\bar{x}^*_r$ be maximum feasible solutions with respect to $w, w_r$ and $\bar{w}_r$. Then

\[
x^*w = x^*w_r + x^*\bar{w}_r \\
\leq x^+_rw_r + \bar{x}^*_r\bar{w}_r \\
\leq p \cdot xw_r + p \cdot x\bar{w}_r \\
= p \cdot xw,
\]

where the first inequality follows from the maximality of $x^*_r$ and $\bar{x}^*_r$, and the second form the assumption that $x$ is a $p$-approximation with respect to $w_r$ and $\bar{w}_r$. $\Box$

We apply weight reduction steps iteratively, while assuring that any $p$-approximate solution to $w_r$ can be easily extended into a $p$-approximate solution to $\bar{w}_r$.

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, $w_r$ and $\bar{w}_r$, as follows: $\bar{w}_r[e'] = w[e]$ for every $e' \in N^+(e)$, $\bar{w}_r[e'] = 0$ for every other edge, and $w_r = w - \bar{w}_r$. Any 2-approximate solution for the reduced 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
$w_r[e] = 0$, adding $e$ to the solution does not change the solution’s value. Thus, we get a 2-approximate solution for both $w_r$ and $\bar{w}_r$.

This simple technique is realized by Algorithm 2. 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 $\bar{w}_{i+1}$ (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.

**Algorithm 8.1: MWM-simple($V, E, w$).** A simple 2-approximation algorithm for MWM

1. $S \leftarrow \text{empty stack}$
2. $w_1 \leftarrow w; i \leftarrow 1$
3. foreach $e_i \in E$ s.t. $w_i(e_i) \geq 0$ do
   4. $S.push(e_i)$
   5. $w_{i+1} \leftarrow w_i$
   6. foreach $e' \in N^+(e_i)$ do
      7. $w_{i+1}[e'] \leftarrow w_i[e'] - w_i[e_i]$
      8. /* Implicit: $\bar{w}_{i+1}[e'] \leftarrow w_i[e_i]$ for every $e' \in N^+(e_i)$ */
   9. $i \leftarrow i + 1$
9. $k \leftarrow |S|$  
10. $x_{k+1} \leftarrow \vec{0}$
11. for $i \leftarrow k$ down to 1 do
   12. $x_i \leftarrow x_{i+1}$
   13. $e_i \leftarrow S[i]$  
14. if $\forall e \in N(e_i) : x_i[e] = 0$ then
15. $x_i[e_i] \leftarrow 1$
16. return $x_1$

We also note that this algorithm does not work in the semi-streaming model, as the stack can easily grow to contain $\Omega(n^2)$ edges.

### 8.3.2 Extending the Local-Ratio Technique

We now extend the approximation techniques used in Algorithm 2. This allows us to present another sequential approximation algorithm for MWM in the following section, with a worse approximation ratio of $2 + \epsilon$. 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 8.2.** Let $w, w_r$ and $\bar{w}_r$ be weight functions and $e \in E$ an edge, such that

$$\bar{w}_r[e'] = \begin{cases} 
  w[e] & e' = e; \\
  \alpha w[e] \text{ or } w[e] & e' \in N(e); \\
  0 & \text{otherwise},
\end{cases} \quad (8.1)$$

and $w_r = w - \bar{w}_r$; the choice between $w[e]$ and $\alpha w[e]$ can be arbitrary.

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

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

Next, we note that if the optimal solution for the reduced 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 reduced 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 8.3.** Let $w, w_r$ and $\bar{w}_r$ be weight functions satisfying $w = w_r + \bar{w}_r$ and $w_r[e] \leq w[e]$ for all $e \in E$. Let $x_r$ be a $\beta$-approximate solution for $w_r$.

If $x_r w$ is at least $p$ times larger than any matching in $\bar{w}_r$, then $x_r$ is a $(\beta + 1/p)$-approximate solution for $w$.

**Proof.** Let $x^*, x^*_r$ and $\bar{x}^*_r$ be matchings of maximum weights in $w, w_r$ and $\bar{w}_r$ respectively.

The assumptions imply $x^*_r w_r \leq \beta x_r w_r$ and $p\bar{x}^*_r \bar{w}_r \leq x_r w_r$, so

\[
\begin{align*}
x^* w &= x^*_r w_r + x^*_r \bar{w}_r \\
&\leq x^*_r w_r + \bar{x}^*_r \bar{w}_r \\
&\leq \beta x_r w_r + (1/p) x_r w_r \\
&= (\beta + 1/p) x_r w_r \\
&\leq (\beta + 1/p) x_r w,
\end{align*}
\]

where the last inequality follows from the fact that $w_r[e] \leq w[e]$ for all $e \in E$. \qed
Let \( w_1 \) be a weight vector for the MWM problem, and consider an iterative splitting of \( w_i \) into \( w_{i+1} \) and \( \bar{w}_{i+1} \) 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 8.4.** Let \( G = (V, E, w_1) \) a graph, and \( w_2, \ldots, w_{k+1} \) and \( \bar{w}_2, \ldots, \bar{w}_{k+1} \) sequences of reduced and residual weight functions for \( (V, E) \), respectively.

Assume that we generate a sequence of solutions \( x_{k+1}, \ldots, x_1 \), 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 \( w_i \) 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 \( \bar{w}_{i+1} \); or
   
   (b) \( x_i w_{i+1} \geq (\gamma/\beta_{i+1})x^*\bar{w}_{i+1} \) 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 8.1), \( x_i \) 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 8.3 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:

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

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

### 8.4 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 2; 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 8.4 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 8.4 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 \( O(n \text{ polylog } n) \) edges. The new algorithm does not discard edges from the stack, but only replaces them with the \( \perp \) 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.

### 8.4.1 Algorithm $\text{MWM-\text{-seq}}$

Algorithm $\text{MWM-\text{-seq}}$ (Algorithm 3) 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 \( \perp \) in its place in the stack.

In the second phase, the algorithm unwinds the stack, adding edges greedily to the matching while ignoring the \( \perp \) symbol. The usage of the \( \perp \) 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 \textsc{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 \( c_i(u) - c_j(u) \).

**Lemma 8.5.** 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

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

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 \( c_i(u) = c_{i-1}(u) + 1 \). \( \Box \)

Consider the sequences of reduced and residual edge-weight functions, \( w_2, \ldots, w_{k+1} \) and \( \widehat{w}_2, \ldots, \widehat{w}_{k+1} \), 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 \( w_i \)
Algorithm 8.2: MWM-seq($V, E, w$). A sequential approximation algorithm for MWM

1 $S \leftarrow $ empty stack
2 $w_1 \leftarrow w; \phi_0 \leftarrow 0; c_0 \leftarrow 0$
3 /* $c_i$ is only used for the proof */
4 $\forall v \in V : E_0(v) \leftarrow $ empty queue
5 $i \leftarrow 1$
6 foreach $e = (u, u') \in E$ do
7     if $w_1[e] \leq \alpha (\phi_{i-1}(u) + \phi_{i-1}(u'))$ then
8         continue /* Implicit: $\bar{w}_{j+1}[e] \leftarrow \alpha w_j[e_j]$ for every $e_j \in N(e)$ */
9     $e_i \leftarrow e$
10     $S$.push($e_i$
11     $w_1[e_i] \leftarrow w_1[e_i] - (\phi_{i-1}(u) + \phi_{i-1}(u'))$
12     /* Implicit: $\bar{w}_{j+1}[e_i] \leftarrow w_j[e_j]$ 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         $\phi_i(v) \leftarrow \phi_{i-1}(v) + w_1[e_i]$
18         if $(\alpha - 1)\alpha |E_i(v)|^{-2} > 2\alpha \gamma$ then
19             $e_j \leftarrow E_i(v).dequeue()$
20             $S[j] \leftarrow \bot$
21     $i \leftarrow i + 1$
22     $k \leftarrow |S|$
23     $x_{k+1} \leftarrow 0$
24     for $i \leftarrow k$ down to 1 do
25         $x_i \leftarrow x_{i+1}$
26         $e_i \leftarrow S[i]$
27         if $e_i = \bot$ 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 $x_1$

into $w_{i+1}$ and $\bar{w}_{i+1}$, the latter defined by

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

(8.2)
and the former by \( w_{i+1} = w_i - \bar{w}_{i+1} \). The length \( k \) is the number of heavy edges encountered in the first phase. Note that \( \bar{w} \) 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 8.6.** 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

\[
\begin{align*}
w_{j+1}(e_i) & \geq w_i(e_i) \\
& \geq (\alpha - 1)\phi_{i-1}(u) \\
& \geq (\alpha - 1)\alpha^{c_i(u)-c_j(u)}\phi_j(u) \\
& \geq (\alpha - 1)\alpha^{c_i(u)-c_j(u)}w_j[e_j] \\
& = (\alpha - 1)\alpha^{c_i(u)-c_j(u)-1}w_j[e_j] \quad (u \in e_i \text{ implies } c_{i-1}(u) = c_i(u) - 1)
\end{align*}
\]

as desired. \( \square \)

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 8.7.** 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 8.4 holds. We show that condition 2(b) of Lemma 8.4 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 \( (\alpha - 1)\alpha^{c_{i'}(v)-c_i(v)-2} > 2\alpha\gamma \) 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, \( (\alpha - 1)\alpha^{c_{i'}(v)-c_i(v)-1} \geq 2\alpha\gamma \).

Using this inequality and Lemma 8.6, we have

\[
w_{i+1}[e_{i'}] \geq (\alpha - 1)\alpha^{c_{i'}(v)-c_i(v)-1}w_i[e_i] \geq 2\alpha\gamma w_i[e_i].
\]

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 $\bar{w}_{i+1}$ guarantees it has the following structure:

$$
\bar{w}_{i+1}[e'] = \begin{cases} 
  w_i[e_i] & e' = e_i; \\
  \alpha w_i[e_i] & e' \in N(e_i); \\
  0 & \text{otherwise}.
\end{cases}
$$

(8.3)

Thus, any solution $x^*$ for $\bar{w}_{i+1}$ contains at most two edges, of weight at most $\alpha w_i[e_i]$, i.e. $2\alpha w_i[e_i] \geq x^*\bar{w}_{i+1}$. The last two inequalities guarantee any solution $x^*$ satisfies

$$(\beta_{i+1}/\gamma)x_{i+1}w_{i+1} \geq 2\alpha w_i[e_i] \geq x^*\bar{w}_{i+1}$$

so $x_iw_{i+1} = x_{i+1}w_{i+1} \geq (\gamma/\beta_{i+1})x^*\bar{w}_{i+1}$, and condition 2(b) of Lemma 8.4 holds.

Lemma 8.8. If $x_{i+1}$ is a $\beta_{i+1}$-approximation for $w_{i+1}$ and the condition on line 27 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 $\bar{w}_{i+1}[e_i] = w_i[e_i]$ and $w_{i+1} = w_i - \bar{w}_{i+1}$, 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 8.4 holds. By Lemma 8.2, $x_i$ is a $(2\alpha)$-approximate solution for $\bar{w}_{i+1}$, and because $2\alpha \leq \beta_{i+1}$ it is also a $\beta_{i+1}$-approximate solution to $\bar{w}_{i+1}$ and condition 2(a) of Lemma 8.4 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 8.4. The condition in Line 27 does not hold, so $x_{i+1}[e'] \neq 0$ for some $e' \in N^+[e_i]$, and Lemma 8.2 promises $x_i$ is a $(2\alpha)$-approximation for $\bar{w}_{i+1}$. As before, $2\alpha \leq \beta_{i+1}$ proves condition 2(a) of Lemma 8.4 holds.

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

Lemma 8.9. 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 $\bar{w}_{i+1}[e_i] = w_i[e_i]$ and $w_{i+1} = w_i - \bar{w}_{i+1}$ imply $w_{i+1}[e_i] = 0$. The monotonicity of $w_i[e_i]$ completes the proof.

If $e = (u, u')$ is a light edge considered in iteration $i$, then $w_i[e] \leq \alpha(\phi_{i-1}(u) + \phi_{i-1}(u'))$. Line 15 guarantees

$$\phi_{i-1}(u) = \sum_{\{e_j | j \leq i-1 \text{ and } e_j \in N(e)\}} w_j[e_j],$$

and a similar claim holds for $u'$. On the other hand, $w_{j+1} = w_j - \bar{w}_{j+1}$ and $\bar{w}_{j+1}[e] = \alpha w_j[e_j]$ 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

$$w_i[e] = w_1[e] - \alpha \sum_{\{e_j | e_j \in N(e) \text{ and } j \leq i-1 \}} w_j[e_j].$$

139
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 \( w_1[e] \leq \alpha(\phi_{i-1}(u) + \phi_{i-1}(u')) \) implies \( w_i[e] \leq 0 \) for all \( e \in E \), and the monotonicity of \( w_i[e] \) completes the proof.

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

**Theorem 8.10.** Algorithm \( \text{MWM-seq} \) returns a \((2 + \epsilon)\)-approximation for the MWM problem.

**Proof.** By Lemma 8.9, the first loop ends when \( w_{k+1} \leq 0 \), so \( x_{k+1} = \vec{0} \) is indeed an optimal solution for \( w_{k+1} \).

Assume \( x_{i+1} \) is a \( \beta_{i+1} \)-approximate solution for \( w_{i+1} \). From Lemmas 8.7 and 8.8 we conclude that in all cases the conditions of Lemma 8.4 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

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

The desired approximation ratio is achieved.

\[\square\]

8.4.2 Implementing Algorithm \( \text{MWM-seq} \) in the Semi-Streaming Model

In the previous section we showed that Algorithm \( \text{MWM-seq} \) 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 \( \perp \) 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 \( \text{MWM-semi} \) (Algorithm 4), an implementation of Algorithm \( \text{MWM-seq} \) in the semi-streaming model. The correctness of Algorithm \( \text{MWM-semi} \) is derived directly from the correctness of Algorithm \( \text{MWM-seq} \), so we only need to prove it obeys the space constraints.

After omitting notations and auxiliary variables from Algorithm \( \text{MWM-seq} \), we are only left with three types of data structures in Algorithm \( \text{MWM-semi} \): \( M \) 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 \( v \).
Algorithm 8.3: \texttt{MWM-semi}(V, E, w). A Semi-Streaming approximation algorithm for MWM

\begin{algorithm}
\begin{algorithmic}[1]
\State $S \leftarrow$ empty stack
\State $\phi \leftarrow \vec{0}$
\State $\forall v \in V : E(v) \leftarrow$ empty queue
\ForEach{$e = (u, u') \in E$}
\If{$w[e] \leq \alpha (\phi(u) + \phi(u'))$}
\State continue
\EndIf
\State $S$.push($e$)
\State $w'[e] \leftarrow w[e] - (\phi(u) + \phi(u'))$
\ForEach{$v \in e$}
\State $E(v)$.enqueue($e$)
\State $\phi(v) \leftarrow \phi(v) + w'[e]$
\If{$(\alpha - 1)\alpha |E(v)|^{-2} > 2\alpha \gamma$}
\State $e' \leftarrow E(v).dequeue()$
\State remove $e'$ from $S$
\EndIf
\EndForEach
\EndFor
\State $M \leftarrow \emptyset$
\While{$S \neq \emptyset$}
\State $e \leftarrow S.pop()$
\If{$M \cap N(e) = \emptyset$}
\State $M \leftarrow M \cup \{e\}$
\EndIf
\EndWhile
\State return $M$
\end{algorithmic}
\end{algorithm}
Lemma 8.11. During the execution of Algorithm \texttt{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 $(\alpha - 1)\alpha^{\log|E(v)|} \leq 2\alpha\gamma$ 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

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

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

From Lemma 8.11 we conclude that for a constant $\epsilon$, Algorithm \texttt{MWM-semi} maintains at most $O(n \log n)$ edges, each represented by $O(\log n)$ bits. 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 8.12. 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 9

Conclusions

This thesis studies a wide range of problems in the field of distributed graph algorithms and related topics.

The first topic studied is distance computation algorithms in graphs. We define a shortest lightest tree, and present a distributed algorithm that builds many such trees in parallel. While we show an application of this algorithm for spanner construction, we believe that it can find other applications to distance-related problems such as routing table construction. Moreover, we hope this algorithm could help in designing faster algorithm for one of the central problems in our field: computing all-pair-shortest-paths.

We then turn to study the distributed construction of spanners. We devise distributed algorithms for the construction of several families of purely additive spanners, and present a lower bounds for one of these families — pairwise spanners. Our lower bounds are not limited to purely additive spanners, but they also apply to pairwise $(\alpha, \beta)$-spanners in general. The proof of these lower bounds uses a new communication complexity problem, which has to do with computing a relation and not a function, a technique we that believe could be of independent interest. Yet, finding the complexity of building spanners is left as an interesting question for future research; of example, even in the case for pairwise spanners, it is not clear if the complexity should be a function of the number of pairs, as in the lower bound, or the number of nodes appearing in these pairs, as in the algorithms.

We further study lower bounds for distributed graph algorithms, and use a new lower bound graph construction in order to present the first super-linear lower bounds for distributed graph algorithms in the CONGEST model. Specifically, we present almost-quadratic lower bounds for the minimum vertex cover, maximum independent set and chromatic number computation problems. While all these problems are NP-hard, we also present lower bounds for problems in P, and for one of them we show an $\Omega(n^2)$ lower bound for deterministic algorithms and a $O(D)$ rounds randomized algorithm, achieving the largest possible gap for a global problem. To conclude the part dealing with lower bounds for the CONGEST model, we discuss the limitations of current techniques in improving the state-of-the-art lower bounds for the APSP problem. Following our lower bounds, we wonder if there are natural problems with time complexity $\Theta(n^\delta)$ for $1 < \delta < 2$, and more
generally, we suggest a round-complexity based hierarchy of problems in the CONGEST model as an interesting venue for future research.

In an attempt to gain better understanding of distributed algorithms for computing APSP, we study it in the related CLIQUE model. In this model, we are able to compute APSP fast using matrix multiplication algorithms, which is the standard technique for computing APSP in the sequential setting. We further use our distributed implementation of matrix multiplication in order to solve several other graph problems in the CLIQUE, providing faster algorithms for problem such as triangle and 4-cycle counting. The rapid computation of matrix multiplication in the CONGEST model is beyond the scope of our current techniques; however, we hope that our work in the CLIQUE model can be used for adapting fast matrix multiplication algorithms to the CONGEST model as well, thus helping in the design of faster algorithms for the APSP problem in this model.

In addition to computing graph parameters and building subgraphs, we are also interested in verifying the results of previous computations. To this end, we consider proof-labeling schemes, which consist of assignment of labels to graph nodes in a way that allows the fast verification of a given claim. We prove lower bounds on the label sizes, using lower bounds for nondeterministic communication complexity. Then, we suggest that these bounds can be circumvented by compromising on an approximated verification, define the relevant notion of approximate proof-labeling schemes, and present such schemes for the problems of computing the diameter and a maximum weight matching.

Finally, we discuss the semi-streaming model of computation, in which we present a \((2 + \epsilon)\)-approximation algorithm for the maximum weight matching problem. This algorithm improves over the approximation ratio of previous algorithms for the problem, and matches the best known algorithms for the unweighted case of the problem. Further improving the approximation ratio for the weighted or unweighted cases of the problem, or proving matching lower bounds on the approximation ratio, are all interesting questions for future research.
Bibliography


fuer Informatik, 2016.


[MP14] Dániel Marx and Michal Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science, STACS*, pages 542–553, 2014.


[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest


[Tis99] Alexandre Tiskin. The Design and Analysis of Bulk-Synchronous Parallel Algorithms.


In the methods of fast matrix multiplication, we present an algorithm that solves the problem of computing all distances in a graph much faster than before, and also additional algorithms for solving other problems in graphs, such as the number of triangles in a graph. This technique is also used to solve the problem of computing the shortest path. 

CONGEST

In fast methods for constructing shortest paths in graphs, we present an algorithm that solves the problem of computing all distances in a graph in a model-

Proof

In addition to the properties of graphs and the construction of subgraphs, there is also interest in the verification of claims. In order to do so, we consider methods of verifying properties, in which we mark a subset of vertices in the graph in a way that enables fast verification of the claims.

Labeling Schemes

In addition to the properties of graphs and the construction of subgraphs, there is also interest in the verification of claims. In order to do so, we consider methods of verifying properties, in which we mark a subset of vertices in the graph in a way that enables fast verification of the claims.

Semi-Streaming

In fast methods for constructing shortest paths in graphs, we present an algorithm that solves the problem of computing all distances in a graph in a model-

In fast methods for constructing shortest paths in graphs, we present an algorithm that solves the problem of computing all distances in a graph in a model-

Semi-Streaming

In fast methods for constructing shortest paths in graphs, we present an algorithm that solves the problem of computing all distances in a graph in a model-

ii

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-16 - 2017
A survey is conducted on a variety of problems related to computing distances on various networks, as well as a number of related issues. We use specific models in this chapter, namely the CONGEST model, which considers the network model, a congestion model, and a circuit model. The main purpose of this chapter is to present algorithms for computing distances in graphs and solving various problems related to navigation.

The chapter introduces the concept of graph spanners and presents an algorithm for constructing a graph spanner with linear growth factor. The chapter also presents a new algorithm for constructing a graph with bounded degree, and a lower bound for a family of graphs. The lower bound is not limited to graphs with bounded degree, but also applies to graphs with bounded degree and bounded diameter. The chapter concludes by presenting lower bounds for algorithms in the CONGEST model, and an algorithm for computing the distance in a graph with bounded degree.

In summary, this chapter presents algorithms and models for computing distances and solving various navigation problems. The chapter introduces new concepts and algorithms, and presents lower bounds for algorithms in various models.
תודות
למודי הדוקטורט של לערכו בחנוכית של פרופסור קרן צנזור-הלל בפקולטה למדעי מחשב.
בכתכינו. קרן הציעה ליBern במחקר בביות השירה האידיאלים, ידע כלון אחר מחד.
REDENTIAL אנשי המחקר את החזון העניין של פאסי, ותמכה עשתה את ביטחון המבוך.
למדה את מחזור זה בבעית הפסקאות עם לערכו בברית עם עריכתי.
רחבתי.
ל ancor תמית, קרן עודדה אתיל לשכת פועלו עם ויקיר גאראים, ומחנה בקתשלו
ומתינה יות מפתקרים ייחודי. אני מודה לפסקאות הפסקאות והפקצה בברית, sólo מחמדית.
 Öncekiיה.
הפקולטה למדעי המחשב פסכה היהת לי בית שלימודי כלכלי, יוהת תרגום מוגזים
לפקולטה, למדתי בכל הפוסקים ו لمدة ניסיון באופן נואם, אך בחרתי גוות. אם בכל, הרכה או ברהום או יומאים שלברחים.
ביוליט ונספור ארחות פרווזים, הפסקת את, משקק בילדראד ושוחף מוסדרון.
ותרוי תוספי נמל גיאית показ עניין בבר - הווה מתוך. וMouseEventי בכרוחיות דרור אופי
ומדי.
הפקולטה פסכה בבהמת טבע, בל окруג לע החבורה והחרים שברוך. על כל תנועה בל תכנית.
אשתית הומך לאחר הגידול של לגור饮水 והתמקם, והackBar את באישול וدادגון של מסלול
החיים האקדיום. של תייגוב אבר וחובל, ובאשר לחון אינני הכניסי והנופים. על כל
הכישר הניצי את האベンגר, מבאר את בראוניות החבר, ושווה את vrouעה השבלית פנינו
התקן. של החבורה תמיתידוע לחהול קר באת הדוברי השצובים באית בחיים, והנהו והז.
מוקדשתeland.

אני מודה לחבר הקירות הראויים לפחד (מענק מספר 1696/14) לטכניקאי על התמיכה המכלים בנידובה.
בשםleftrightarrow.
числен מרחקים מבודר
ובעיות קשורות

חיבר על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

עמי פז

הנהלת הטכניון – מכון טכנולוגי לישראל
הנהלת הטכניון – מכון טכנולוגי לישראל

2017

2017

הנהלת הטכניון – מכון טכנולוגי לישראל
הנהלת הטכניון – מכון טכנולוגי לישראל

2017
חישוב מרחקים מבודד
ובsteen קשורות

עופי פז