GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore

DOCTORAL DISSERTATION

Fast Computing in Networks with Limited Bandwidth

PHD PROGRAM IN COMPUTER SCIENCE

Ph.D. candidate:

Dennis OLIVETTI
dennis.olivetti@gssi.it

Supervisor:

Prof. PIERRE FRAIGNIAUD
pierre.fraigniaud@irif.fr

December 2017

GSSI Gran Sasso Science Institute
Viale Francesco Crispi, 7 - 67100 L'Aquila - Italy

Supervising professor
Professor Pierre Fraigniaud
Institut de Recherche en Informatique Fondamentale (IRIF),
CNRS and University Paris Diderot, France

Thesis advisor
Professor Pierre Fraigniaud
Institut de Recherche en Informatique Fondamentale (IRIF),
CNRS and University Paris Diderot, France

Preliminary examiners
Professor Keren Censor-Hillel – *Technion Israel Institute of Technology*
Professor Fabian Kuhn – *Albert-Ludwigs-Universität Freiburg*

During my doctoral research, I shared my time between Gran Sasso Science Institute (GSSI), Italy, and Institut de Recherche en Informatique Fondamentale (IRIF), France.

Declaration

The author declares that most of the contents of this thesis are based on contents published in co-authored papers.

More precisely, the content showed in Chapter 3 is a merge of the contents of two articles. One has been published at SPAA 2017 [39] and invited to a special issue of TOPC 2017. The other has been published at DISC 2017 [32].

The content of Chapter 4 is still unpublished.

The content of Chapters 5 and 6 are based on an article published in the proceeding of SIROCCO 2016 [12].

The remaining of the thesis is a reelaboration and extension of the content published in the aforementioned papers.

Abstract

The CONGEST model for distributed network computing is well suited for analyzing the impact of limiting the throughput of a network on its capacity to solve tasks efficiently. Many problems that are trivial when the bandwidth is not constrained become hard when imposing such constraints. For example, even distributedly detecting the presence of a cycle of length 4 may require a lot of bandwidth, and in fact, if the bandwidth is of $O(\log n)$ bits, it requires $\widetilde{\Omega}(\sqrt{n})$ rounds of computation. In this thesis we study problems related to congestion in distributed computing.

At first, we address problems related to subgraph detection, by showing efficient algorithnms that are able to distributedly detect fixed patterns in the communication graph. We initially show how to detect any fixed tree T of constant size in a constant number of rounds. We then apply this result to *distributed property testing*, by showing that for a wide category of graph patterns H we can test in constant time whether the graph is *far* from being H-free.

After, we further examine the role of the bandwidth in distributed computing, by investigating how much the speed of a distributed algorithm can scale while changing the amount of allowed bandwidth. First, we show that different problems can benefit differently from having more bandwidth. We adapt existing distributed algorithms designed to use messages of size $O(\log n)$ to being time efficient when messages are bigger. Then, we show that, in some limit cases, having more bandwidth can not help at all.

Then, we study problems related to the *Core-Periphery* model. First, we give tradeoffs between the number of edges and the number of rounds required to emulate the Congested Clique model, by characterizing graphs that are able to emulate the clique communication efficiently. Then, we show an efficient way to solve the Minimum Spanning Tree construction problem in this model.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Pierre Fraigniaud, for all his help and continuous support. I would like to gratefully acknowledge his guidance and encouragement during my Ph.D., as without them, this thesis would not have been possible.

A special thanks goes to Michele Flammini and Zvi Lotker, for all their help and support during these years.

Thanks to Rocco De Nicola for all his support.

Thanks are due to all my colleagues and researchers at GSSI and IRIF: Alex Bredariol Grilo, Feliciano Collela, Simon Collet, Gianlorenzo D'Angelo, Laurent Feuilloley, Bruno Guillon, Juho Hirvonen, Maria Rita Iacò, Emilio Incerto, Bruno Karelovic, Alessandro Luongo, Fabian Reiter, Pablo Rotondo, Anna Carla Russo, Gian Luca Scoccia, Catia Trubiani, and Cosimo Vinci.

Thanks to Keren Censor-Hillel and Fabian Kuhn for reviewing my thesis.

I would like to thank my family for always supporting me.

Finally, my deepest thanks to Alkida Balliu.

Contents

Abstract **1**

Acknowledgements **3**

List of Figures **7**

1 Introduction **9**
1.1 Setting . . . 10
1.2 Subgraph Detection . . . 14
1.3 Tradeoffs Between Bandwidth and Time . . . 16
1.4 Clique Emulation . . . 18
1.5 Minimum Spanning Tree Construction . . . 19

2 Model and Definitions **21**
2.1 The CONGEST Model . . . 21
2.2 The Congested Clique . . . 22
2.3 Property Testing . . . 23
2.4 Distributed Property Testing . . . 25
2.5 Core-Periphery Networks . . . 26
2.6 Minimum Spanning Tree . . . 27
2.7 Single Source Shortest Path . . . 29
2.8 All Pairs Shortest Paths . . . 30

3 Subgraph Detection **31**
3.1 Introduction . . . 31
3.2 Our Goal . . . 34
3.3 Results . . . 36
3.4 Detecting the Presence of Trees . . . 39
3.5 Distributed Property Testing . . . 45
3.6 Conclusions . . . 49

4 Tradeoffs Between Bandwidth and Time **51**
4.1 Introduction . . . 51
4.2 Our Goal . . . 52
4.3 Results . . . 53
4.4 All Pairs Shortest Paths . . . 55

4.5 Minimum Spanning Tree 56
4.6 Single Source Shortest Path 59
4.7 $Distance_k$ 62
4.8 Conclusions 66

5 Clique Emulation **67**
5.1 Introduction 67
5.2 Our Goal 68
5.3 Results 69
5.4 Related Work 69
5.5 Deterministic Construction 70
5.6 Randomized Construction 74
5.7 Conclusions 78

6 MST In Core-Periphery Networks **79**
6.1 Introduction 79
6.2 Results 80
6.3 MST Construction 80
6.4 Conclusions 87

7 Conclusions and Open Problems **89**

List of Figures

2.1 Example of ϵ-farness 24
2.2 Example of a Core-Periphery network 27
2.3 Example of an MST 28

3.1 A lollipop graph 37
3.2 Example of graphs composed by a tree, an edge, and arbitrary connections between them 39

4.1 Complexity of MST as a function of the available bandwidth . . . 59
4.2 Example of an instance of the $Distance_k$ problem 63
4.3 Example of reduction from $Pointer_k$ to $Distance_k$ 65
4.4 Complexity of $Distance_k$ as a function of the available bandwidth 65

5.1 Axiom 2 of Core Periphery networks 68
5.2 Example of Johnson graph 71
5.3 Emulation of a removed edge 71
5.4 Emulation of K_9 with $K_{3,6}$ 72
5.5 Number of edges necessary to emulate the clique communication 74

6.1 Graphs that satisfy only 2 axioms of Core-Periphery networks . . 80

Chapter 1

Introduction

A distributed system is composed by entities that cooperate to achieve a common goal. Usually, an entity is assumed to be a computing device that has a processor and its private memory. However, distributed systems are studied in various areas, such as biology, where entities may be cells or insects. The field that studies how entities can solve problems in distributed systems is called *distributed computing*.

Due to the fact that these networks may be huge (think about the Internet or about social networks), determining whether the network satisfies some specific property may be challenging. An example is the problem of *pattern detection*, where we want to distributedly detect if the network contains or not some specific pattern. This problem has many applications in the real world. For example, in the context of networking, a pattern may reveal some anomaly in the network, like a cycle in the routing tables. In the context of social networks, a pattern may give useful information about the population, for example the presence of a clique reveals a group of friends that all know each other. In chemistry, detecting patterns allows to find similarities between chemical compounds. In this thesis, we study pattern detection in the context of *distributed property testing*, where we are allowed to lose precision on the result for gaining computational time. In this context, we first show how to efficiently detect trees of any constant size. Then, we present a distributed algorithm able to detect any pattern composed by a couple of nodes connected to a fixed tree in an arbitrary manner. Although this family of patterns may look artificial, notice that this is

not the case. In fact, it includes important patterns like cycles, the clique of size four, and complete bipartite graphs $K_{2,k}$ for any $k \geq 1$.

An important aspect of a network is its ability to allow entities to communicate efficiently. By allowing all entities to communicate with each other, and thus by supporting an efficient all-to-all communication, we can design simple and efficient algorithms that ignore all communication issues and delegate them to the network itself. If the communication graph is a clique, then all nodes can communicate with each other easily, but, unfortunately, if the communication graph is sparse, the all-to-all communication becomes challenging. Notice that, in real world networks, connecting all entities to each other may be too costly, since the number of links would be quadratic in the number of entities. For this reason, in this thesis we study tradeoffs between the number of links and the time required to emulate the all-to-all communication. This task becomes challenging when we consider communication links of limited capacity. In fact, our challenge consists of carefully choosing routing paths that avoid bottlenecks.

In distributed computing, the amount of available bandwidth plays an important role on the ability to solve tasks efficiently. Unfortunately, for many problems our current knowledge is limited to specific cases where the bandwidth is either logarithmic in the size of the network, or it is unbounded. We make a step forward in understanding the influence of the bandwidth, by establishing tradeoffs between the bandwidth and the time required to solve a task. For this purpose, we study different important distributed problems and analyze how the bandwidth affects their running time. We show that different problems are differently influenced by the amount of bandwidth allowed: there are problems for which their running time fully scales with the amount of bandwidth, there are problems that scale but not linearly, and there are tasks that do not benefit at all from having more bandwidth.

1.1 Setting

This thesis studies problems related to *distributed computing*, where the nodes of a network cooperate to solve a problem by executing a distributed algorithm. We consider the CONGEST model [80], a synchronous model where nodes are

considered to be fault-free and capable of doing an arbitrary amount of computation at each round. The main restriction imposed by the CONGEST model is to limit the amount of data that can be transferred by the nodes at each round. This model is well suited for analyzing the impact of the bandwidth on the time required to solve a task, while ignoring other problems related to synchronization and faults. Another well studied model in distributed computing is the so-called LOCAL model, that is, the CONGEST model with no restriction on the size of the messages [80].

In the general distributed synchronous model, a network can be represented as a graph, where nodes are machines and edges are connections between them. Two machines are considered to be neighbors if they have a communication link connecting them.

Each machine executes the same algorithm and has its own private memory, and in order to share data between different machines it is necessary to transmit messages. We assume that the computation starts at the same time and proceeds in synchronous rounds, where at each round each machine can send a different message to each neighbor. In other words, machines have access to a global clock, messages are always delivered in a fixed amount of time, and the computation that each machine performs between rounds requires a fixed amount of time. We do not restrict the amount of memory or the amount of computation allowed to each machine and we assume that the machines never crash or exhibit Byzantine faults. Machines have unique identifiers and may have access to private or shared random bits.

In the CONGEST model, the only added restriction is on the size of the messages. Typically messages are chosen to be bounded to $O(\log n)$ bits, where n is the number of nodes of the network. This seems to be the minimum reasonable amount required to solve tasks, because it is enough for transmitting, in a single round, a constant number of identifiers, or weights of edges. By abstracting away all the problems that a distributed network could exhibit and by considering only the problem related to congestion, this model is well suited for analyzing the amount of bandwidth required to solve a task distributedly.

To each node may be provided some input, that could be for example a labeling, or a weight for each of its incident edges. The complexity of a task is measured by the number of rounds required to solve it. Notice that, if the input provided

to each node has a reasonable size, in $O(n^2)$ rounds it is possible to solve any problem by just broadcasting the input and the graph structure, gather everything on some fixed node, and finally solve the task locally.

Typically, in the CONGEST model, the time complexity of problems is analyzed for the case where the bandwidth is set to be $O(\log n)$. Apart from designing algorithms for this model, a lot of effort is put on providing lower bounds, usually by reducing some communication complexity problem to the required distributed task. Often in this case the lower bounds directly depend on the bandwidth limit. For example, for the Minimum Spanning Tree construction task is proved a lower bound of $\Omega(\sqrt{\frac{n}{B}})$ [22], where B is the bandwidth constraint.

Another model worth of interest is the Congested Clique one, where problems related to *distance* are abstracted away. In this model it is assumed to have a fully connected network, thus allowing each node to communicate to each other node directly in one round.

A similar model is the Core-Periphery one, where there are two sets of nodes, the Core and the Periphery, and it is assumed that three axioms hold. The *clique emulation* axiom ensures that the Core nodes can communicate efficiently between them. The *Periphery-Core convergecast* axiom ensures that Periphery nodes can communicate efficiently with the Core nodes. Then, the *core boundary* axiom gives some constraint on the size of the Core. This model allows to design efficient distributed algorithms, while (differently from the Congested Clique) keeping the total number of edges small.

Usually, problems are classified as *global* or *local*. Global problems are related to global properties of a graph, that is, properties that require a time proportional to the diameter of the graph in order to be checked. For example, distributedly knowing if the communication graph is a tree, is a task that requires global knowledge. Other global problems are *Minimum Spanning Tree construction*, *Shortest Paths* and *Maximum Matching*.

Local problems are related to properties that depend only on the local neighborhood of a node. Although local problems seem easier than global ones, they become hard when imposing a limit on the bandwidth. For example, detecting a cycle of length 4 is trivial if the bandwidth is unbounded (in two rounds each node can gather all the required information), but in the CONGEST model, using

$O(\log n)$ bits per message, it requires $\Omega(\frac{\sqrt{n}}{B})$ rounds [24]. Another example of a local problem is *Maximal Matching*.

Another useful classification is the one between *construction* and *decision* problems. In construction problems we are given a graph, possibly some input to each node, and the nodes have to build a new distributed structure. For example, we may want that the nodes distributely construct a Minimum Spanning Tree of the communication network, or find a maximal matching between them.

In decision problems, instead, the nodes should cooperate to decide if the graph satisfies some property. Also in this case, nodes may have some input labeling. Typically, it is required that if the graph satisfies the property, *all* nodes output "accept", while if the graph does not satisfy the property, *at least one* node outputs "reject". Usually decision problems are hard, as they may require a lot of bandwidth or some global knowledge of the graph. Thus, a natural relaxation arises: instead of requiring to distinguish whether a graph satisfies or not a property, it is just required to distinguish whether a graph satisfies a property, or if it is *far* from satisfying it. This relaxation is called *property testing*. For example, detecting whether a graph is a tree or not requires global knowledge, while detecting a cycle in a graph that contains a large number of cycles is easier and can be performed in logarithmic time [17, 32]. Another example is detecting cycles of length 4, that, even if it is a local problem, requires polynomial time in the CONGEST model [24]. If we relax the problem and we want that nodes reject only when there is a big number (proportional to the number of edges of the graph) of cycles of length 4, the problem becomes solvable in constant time [40, 39].

In this thesis we study different aspects of distributed computing related to the CONGEST model.

- In Chapter 3 we study problems related to subgraph detection, providing algorithms able to detect the presence of any fixed tree as a subgraph of the communication graph, in the classical CONGEST model. Also, it is presented an algorithm able to detect the presence of more complex subgraphs, in the context of *property testing*. This chapter is based on results published in [39] and [32].
- In Chapter 4 we address some questions related to time complexity in the CONGEST model. More precisely, since often upper bounds are given

only for the case where the bandwidth is constrained to be $O(\log n)$, we analyze how the time complexity of existing algorithms can *scale* when more bandwidth is allowed. We show that the complexity of different problems can scale in different ways, and that in some limit cases, up to some point, more bandwidth does not help at all. This chapter is based on [76].

- In Chapters 5 and 6 we study problems related to the *Core-Periphery* model. First, we give tradeoffs between the number of edges and the number of rounds required to emulate the Congested Clique model, by characterizing graphs that are able to emulate the clique communication efficiently. Then, we show an efficient way to solve the Minimum Spanning Tree construction problem in this model. This chapter is based on results published in [12].

Other works done during my doctoral research regard distributed verification in the LOCAL model [9] and algorithmic game theory [10, 11].

1.2 Subgraph Detection

Consider a fixed graph $H = (V(H), E(H))$ and a bigger communication network represented by a graph $G = (V(G), E(G))$. In this thesis we investigate techniques that can be used to distributedly decide if G is H-free, i.e., if it does not contain H as a subgraph. This problem has been investigated in many frameworks, like classical sequential computing [2] or property testing [3]. For example, in sequential computing, for deciding whether a graph H is a subgraph of G, where both H and G are part of the input, the best known bound is exponential [84]. This bound becomes polynomial when H is fixed and only G is part of the input, and even linear if G is planar [29]. If H is a path of length k, the problem is fixed-parameter tractable, having a complexity of $O(nk!)$ [72].

In the context of distributed computing, deciding H-freeness means that the nodes of a network should cooperate to decide whether H is a subgraph of G, satisfying the following constraint:

- if G is H-free then every node outputs accept;

- otherwise, at least one node outputs reject.

H-freeness has been widely studied in the CONGEST model, and even for very simple graph patterns H like C_4, it has been observed that it may require a lot of bandwidth. In fact, Drucker et al. [24] showed a bound of $\widetilde{\Theta}(\sqrt{n})$[1] rounds in n-node networks. The intuition behind this lower bound is that the bandwidth limitation prevents the nodes with high degree to send their list of neighbors on a single communication link, unless consuming a lot of rounds.

When a lower bound is provided, a natural research direction is to find a way to overcome the hardness result. Usually, when the problem is relaxed by restricting the input instances or allowing approximated results, much more efficient solutions arise. In the sequential setting, a relaxation of the subgraph detection problem requires to distinguish whether G is H-free, or if G is *far* from being H-free. This relaxation, called *property testing*, allows to detect some subgraphs in sublinear time. The measure of *farness* is given by the number of edges that should be added or removed from G in order to make it H-free.

The property of H-freeness has been widely studied in the centralized property testing. In dense graphs, by exploiting the *graph removal lemma*, it is possible to show that we can decide H-freeness or induced H-freeness in constant time for any subgraph H of constant size. On sparse graphs, subgraph detection is harder. In fact, even detecting triangles requires $\Omega(n^{1/3})$ queries, and the best known upper bound is $O(n^{6/7})$ queries [4].

Distributed property testing has been introduced by Brakerski et al. [14] and fully formalized for the CONGEST model by Censor-Hillel et al. [17]. They show that, any tester designed for dense graphs for the centralized setting, can be distributedly emulated with just a quadratic slowdown, if the property being tested is non-disjointed. Notice that this allows to distributedly test H-freeness in constant time for any H of constant size, in *dense* graphs. They then provide testers for *sparse* graphs, and, among the various results, they show that triangle-freeness can be distributedly tested in constant time. Then, Fraigniaud et al. [40] extended this work, by showing that for every connected graph H of four vertices, H-freeness can be tested in constant time. However, the same paper shows that the techniques used for testing H-freeness for 4-node graphs H

[1] We use $\widetilde{\Omega}(\cdot)$ to hide polylogarithmic factors in n.

fail to test C_k-freeness or K_k-freeness in a constant number of rounds, whenever $k \geq 5$.

In this thesis we investigate for which patterns H it is possible to efficiently and distributedly decide if a graph is H-free. Since detecting cycles is proven to require a lot of time, we first focus on trees and try to answer the following question:

> For which trees T is it possible to decide T-freeness efficiently in the CONGEST model, that is, in a number of rounds independent from the size n of the underlying network?

At a first glance, deciding T freeness may look simpler than detecting a cycle, since we do not have to find a path having the constraint of starting and ending at the *same* node. However, even deciding P_k-freeness, where P_k is a path of length k, requires to overcome many obstacles. First, finding a longest simple path in a graph is NP-hard, which suggests that it is unlikely that an algorithm deciding P_k-freeness exists in the CONGEST model, with running time polynomial in k at every node. Second, and more importantly, there exists potentially up to $\Theta(n^k)$ paths of length k in a network, which makes it impossible to maintain all of them in partial solutions, as the overall bandwidth of n-node networks is at most $O(n^2 \log n)$ in the CONGEST model.

We then study a relaxation of this problem, by trying to understand for which patterns H we can test H-freeness in the context of distributed property testing. In fact, Fraigniaud et al. [40] proved that their techniques can not work for testing H-freeness for graphs with more than 4 nodes. Thus, we address the following question:

> For which graph patterns H is it possible to test H-freeness efficiently in the context of distributed property testing?

1.3 Tradeoffs Between Bandwidth and Time

Typically, in the CONGEST model, $O(\log n)$ is chosen to be the maximum amount of bits that two neighbors can exchange in a single round. In real networks the

amount of available bandwidth may be different, and in some cases one may prefer to send less and bigger messages, since the latency could heavily impact the running time. Thus, it is reasonable to ask whether the existing algorithms designed for the CONGEST model can be adapted to use messages of different sizes, and what is their performance in this case.

For example, one famous result in the CONGEST model is a minimum-weight spanning tree construction algorithm that performs in $O(D + \sqrt{n}\log^* n)$ rounds in diameter-D n-node networks [62]. It is known that the speed of this algorithm can not scale linearly with the size of the messages, since there exists a lower bound of $\Omega(D + \sqrt{\frac{n}{B}})$, where B is the maximum size of a message. Also, for many problems there is a similar issue: lower bounds depend on the bandwidth parameter B, but the algorithms are analyzed only for the case where $B = O(\log n)$ bits. For example Becker et al. [13] showed how to find a $(1+\epsilon)$ approximation for the Single Source Shortest Path problem in $\widetilde{O}(\epsilon^{-O(1)}\,(n^{\frac{1}{2}} + D))$[2] rounds, using messages of size $O(\log n)$, while the lower bound depends on B, and is $\Omega(D + \sqrt{\frac{n}{B}})$ [22]. Similarly, for the All Pairs Shortest Path problem, we can find a solution in $O(n)$ rounds [55, 81, 68], while the lower bound is B-dependant, and is $\Omega(\frac{n}{B})$ [43].

In this thesis we investigate tradeoffs between the round complexity for solving a task and the bandwidth of the links. First, we investigate the round complexity of existing problems, such as Minimum Spanning Tree, Single Source Shortest Path and All Pairs Shortest Paths, by establishing tradeoffs between the size of the messages B and the number of rounds, by providing algorithms for the CONGEST model, having round complexities that are parametric on B. Thus, we address the following question:

> How well existing algorithms designed for the Minimum Spanning Tree construction, Single Source Shortest Path and All Pairs Shortest Paths problems behave when more bandwidth is allowed?

Then, we further investigate the role of B in the CONGEST model, and try to figure out if B has always some impact on the round complexity. For example, consider two nodes that are at a distance that is equal to the diameter of

[2] We use $\widetilde{O}(\cdot)$ to hide polylogarithmic factors in n.

the graph. Trivially, if they want to share just one bit, they need to wait a time proportional to the diameter of the graph, and even by allowing more bandwidth they can not solve the problem faster. We want to better understand this phenomenon and address the following question:

> Are there problems that can not benefit from the presence of more bandwidth, that are unrelated to trivial bounds related to distances?

1.4 Clique Emulation

A model similar to the CONGEST is the Congested Clique, where problems related to distances are abstracted away. In this model, the communication graph is a clique, i.e., all nodes can communicate with every other node in a single round. This abstraction allows to solve many tasks very efficiently, for example a Minimum Spanning Tree can be constructed in $O(\log \log n)$ rounds deterministically [69], and $O(1)$ rounds [59] if randomization is allowed (notice that even for diameter 3 graphs there is a polynomial lower bound [70]).

Since the number of edges is quadratic in the number of nodes, one may think that this model is too powerful and of limited practical utility. Contrarily, in [52] it is shown that, under some restrictions, fast algorithms for the Congested Clique model can be translated into fast algorithms in the MapReduce framework. Also, Avin et al. [8] proposed a novel network architecture for parallel and distributed computing, called Core-Periphery networks, that resembles the Congested Clique, but it is easier to use it in practice, since in this model the total number of edges is linear in the number of nodes. A part of this network, called *core*, is capable to emulate algorithms designed for the Congested Clique model. Thus, algorithms designed for the Congested Clique model can in fact have practical utility.

Core-Periphery networks are described implicitly by providing three *axioms*. One of these axioms requires that the *core* C satisfies the following:

Clique emulation: the core can emulate the clique in a constant number of rounds in the CONGEST model. That is, there is a communication protocol running in a constant number of rounds in the CONGEST model such

> that, assuming that each node $v \in C$ has a message $M_{v,w}$ on $O(\log n)$ bits for every $w \in C$, then, after $O(1)$ rounds, every $w \in C$ has received all messages $M_{v,w}$, for all $v \in C$. In other words, the *all-to-all* communication pattern can be implemented in a constant number of rounds.

Given this axiomatic description, it is not specified how to actually build a graph that can satisfy the requirements. In other words, if we want to build a Core-Periphery network, we need to find a graph and a routing schema associated to this graph, that is able to satisfy the axiom efficiently.

Due to the existence of very efficient algorithms designed for graphs that allow a fast all-to-all communication, and their applicability to different models, like the Core-Periphery one, it is a natural question to ask which graphs are good candidates to emulate the clique. In this thesis, we aim at establishing tradeoffs between the number of edges of a graph, and the capability of emulating the clique. Thus, we address the following questions:

> What is the minimum number of edges that a graph must have in order to be able to emulate the clique communication in k rounds?
>
> Which graphs are capable to emulate the clique communication efficiently?

1.5 Minimum Spanning Tree Construction

The Minimum Spanning Tree construction problem is well studied in many frameworks. In this problem we are given a graph G, a function w that maps edges to weights, and we want to find a tree containing all the nodes while having minimum weight. The weight of a tree is given by the sum of the weights of all its edges. In the distributed setting each node knows only its incident edges and their weights, and the nodes should collaborate to construct a Minimum Spanning Tree efficiently. At the end of the computation each node should know which of its incident edges are part of the Minimum Spanning Tree.

In the classical CONGEST model, it is possible to solve this problem in $O(D + \sqrt{n}\log^* n)$ rounds [62] and this bound is essentially tight, due to the existence of an $\Omega(\sqrt{\frac{n}{\log n}})$ lower bound [22]. In the Congested Clique model this problem can be solved much faster, due to the presence of sublogarithmic algorithms [69, 47, 59].

Avin et al. [8] showed that Core-Periphery networks can efficiently solve many problems, one of which is the Minimum Spanning Tree construction task. In fact, they provided a randomized algorithm that constructs an MST in $O(\log^2 n)$ rounds. They also showed that if any one of the three axioms that characterize the Core-Periphery networks does not hold, then it is possible to prove polynomial lower bounds. Notably, they do not provide lower bounds showing that their algorithm is optimal. Thus, in this thesis we address the following question:

> How fast can an MST be constructed in Core-Periphery networks?

Chapter 2

Model and Definitions

2.1 The CONGEST Model

We consider the classical CONGEST model for distributed computing [80]. The network is modeled as a connected simple graph (no self-loops, and no parallel edges) where nodes are computing entities that can exchange messages along the edges of the graph. Nodes may have assigned distinct identifiers in a range polynomial in n, the size of the network. Hence, every identifier can be stored on $O(\log n)$ bits.

This model is synchronous, that is, all nodes start the computation simultaneously and execute the same algorithm in a sequence of *rounds*. At each round, each node:

- performs some individual computation,
- sends messages to neighbors in the network, and
- receives messages sent by neighbors.

The main constraint imposed by this model is a restriction on the bandwidth of the links. At each round it is possible to transfer a limited amount of data between neighbors. The model in which the messages are limited to B bits is called CONGEST_B. Typically B is chosen to be $O(\log n)$, and CONGEST is typically used to refer to the $\text{CONGEST}_{O(\log n)}$ model.

The $O(\log n)$-bit bound seems to be the minimum reasonable amount required to solve tasks, because it is needed for transmit, in a single round, a constant number of identifiers.

This model is well suited for analyzing the impact of limiting the throughput of a network on its capacity to solve tasks efficiently. In fact, all other possible issues are abstracted away: nodes have unlimited memory and computational power, do not crash and do not exhibit a Byzantine behavior. Also, all synchronization issues are ignored. The complexity of a distributed algorithm in the CONGEST model is expressed in number of rounds.

Many famous problems have been investigated in the CONGEST model. Different global problems share a common peculiarity: a lower bound of $\Omega(D + \frac{\sqrt{n}}{\log n})$ rounds, even for just verifying if a given solution is correct, or for finding an approximation [22]. Some problems of this type are the Minimum Spanning Tree Construction task, for which there exists a matching upper bound of $O(D + \sqrt{n}\log^* n)$ [62], and Single Source Shortest Path, for which it is possible to find a $(1+\epsilon)$ approximation in $\widetilde{O}(\epsilon^{-O(1)}\,(n^{\frac{1}{2}} + D))$ rounds [13].

A problem that seems much more local is C_4 detection. This problem, in fact, can be solved in constant time if each node can see its $2-$hop neighborhood. Unfortunately, without this assumption, an $\Omega(\sqrt{n})$ lower bound and a matching upper bound hold [24]. This shows that also local problems can heavily suffer from bandwidth limitations.

2.2 The Congested Clique

A model similar to the CONGEST is the Congested Clique, where the communication graph is a clique and the same restrictions of the CONGEST model hold, i.e., all nodes can communicate with all the other nodes in one round using messages of size B. A close but more restrictive model is the Broadcast Congested Clique, where each node is restricted to send the *same* message to all the other nodes.

Various results are known for this model. Lenzen [65] investigated the routing and sorting problems, showing a deterministic algorithm that, if each node is the sender and receiver of at most n messages, allows to route all the messages

in $O(1)$ rounds in a clique of size n using messages of size $O(\log n)$ bits. He also showed an algorithm that allows to sort n^2 keys in constant time. Drucker et al. [24] proved that the Congested Clique is powerful enough to emulate certain classes of bounded depth circuits, which shows how difficult it is to find lower bounds for the Congested Clique. In the case where each node can only broadcast, [24] gives upper and lower bounds for the problem of detecting some types of subgraphs. Hegeman et al. [53] investigated the metric facility location problem providing a $O(1)$ approximation algorithm that runs in expected $O(\log\log\log n)$ rounds. They also showed how to compute a 3-ruling set in the Congested Clique. In [52] it is shown that, under some restrictions, fast algorithms for the Congested Clique model can be translated into fast algorithms in the MapReduce framework. Censor-Hillel et al. [18] showed that matrix multiplication on Congested Clique can be computed in $O(n^{1-2/\omega})$ rounds, where $\omega < 2.3728639$ is the exponent of matrix multiplication. Also, they showed how to use matrix multiplication to solve a variety of graph related problems. In [69] Lotker et al. provided a deterministic Minimum Spanning Tree construction algorithm that runs in $O(\log\log n)$ rounds in the Congested Clique. Then, Hegeman et al. [51] showed that in this context randomization can help, giving a randomized algorithm that requires $O(\log\log\log n)$ rounds. Then, this complexity was even reduced further to $O(\log^* n)$ in [47], and finally to $O(1)$ [59].

2.3 Property Testing

In the sequential setting, *property testing* aims to provide efficient mechanisms able to decide whether a data structure satisfies a given property. In the context of graphs, a *tester* is a centralized algorithm $\mathcal{A}$ that, given the ability to perform queries on the graph, where a query can ask what is the degree of a node, or ask the $i-$th neighbor of a node, must decide whether or not the graph satisfies the given property. The complexity is measured by the number of queries that the tester must perform before providing the result.

In the context of property testing, the requirements are relaxed: the tester should distinguish between instances satisfying the property, and instances that are *far* from satisfying that property. Different notions of farness have been considered:

- In the *dense* model, given any $\epsilon \in (0,1)$, an n-node m-edge network G is said to be ϵ-far from satisfying a graph property $\mathcal{P}$ if adding and/or removing at most ϵn^2 edges to/from G cannot result in a network satisfying $\mathcal{P}$.

- In the *sparse* model, given any $\epsilon \in (0,1)$, an n-node m-edge network G is said to be ϵ-far from satisfying a graph property $\mathcal{P}$ if adding and/or removing at most ϵm edges to/from G cannot result in a network satisfying $\mathcal{P}$.

In Figure 2.1 it is shown an example of a triangle-free graph, a graph that is far from being triangle-free, and a graph that is neither triangle-free, nor far from being triangle-free.

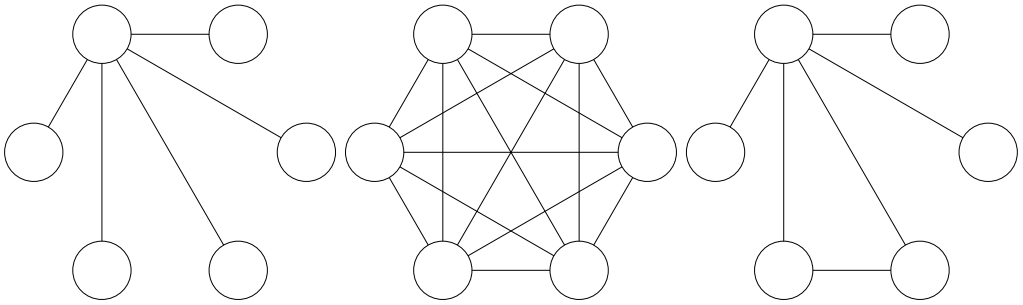

FIGURE 2.1: (left) A triangle-free graph. (center) A graph that is far from being triangle-free. (right) A non-triangle-free graph.

A tester for a graph property $\mathcal{P}$ is a randomized algorithm $\mathcal{A}$ that is required to accept or reject any given network instance, under the following two constraints:

- G satisfies $\mathcal{P} \implies \Pr[\mathcal{A} \text{ accepts } G] \geq 2/3$;

- G is ϵ-far from satisfying $\mathcal{P} \implies \Pr[\mathcal{A} \text{ rejects } G] \geq 2/3$.

If the graph does not satisfy a property, but it is not far from satisfying it, the algorithm can both accept or reject. Notice that the success guarantee $2/3$ is arbitrary, since it is possible to boost any success guarantee by repetition.

Hence, a tester for $\mathcal{P}$ is a mechanism enabling to detect degraded instances (i.e., instances that are far from satisfying a desired property $\mathcal{P}$) with arbitrarily large probability, while correct instances are accepted also with arbitrarily large probability.

A tester has 1-sided error if:

- G satisfies $\mathcal{P} \implies \Pr[\mathcal{A} \text{ accepts } G] = 1$;
- G is ϵ-far from satisfying $\mathcal{P} \implies \Pr[\mathcal{A} \text{ rejects } G] \geq 2/3$.

A well studied problem is H-freeness, where we want to know if a graph G contains H or not as a subgraph. In the *dense* model, using the so called *graph removal lemma*, it is possible to test H-freeness in constant time for any H of constant size. Essentially, this lemma says that if a graph is far from being H-free, than it must contain a very high number of copies of H [30, 3, 5, 20]. In the *sparse* model, subgraph detection is harder. Even detecting triangles requires $\Omega(n^{1/3})$ queries, and the best known upper bound is $O(n^{6/7})$ queries [4].

2.4 Distributed Property Testing

Let $\mathcal{P}$ be a graph property like, e.g., planarity, cycle-freeness, bipartiteness, C_k-freeness, etc. Let $\epsilon \in (0, 1)$.

A distributed property testing algorithm for $\mathcal{P}$ is a randomized algorithm, running in the CONGEST model, which performs as follows. Initially, every node is only given its ID as input. After a certain number of rounds, every node must output a value in $\{\mathsf{accept}, \mathsf{reject}\}$. The algorithm is correct if and only if the following two conditions are satisfied:

- G satisfies $\mathcal{P} \implies \Pr[\text{every node outputs } \mathsf{accept}] = 1$;
- G is ϵ-far from satisfying $\mathcal{P} \implies \Pr[\text{at least one node outputs } \mathsf{reject}] \geq 2/3$.

Like in centralized property testing, in the distributed setting H-freeness is easy in the *dense* model. In fact, it is possible to test H-freeness in constant time for any H of constant size [17]. Although no lower bounds are known, H-freeness in the sparse model seems harder, and only results regarding graphs H of at most four nodes are known [17, 40].

2.5 Core-Periphery Networks

Core-Periphery is a novel network architecture for parallel and distributed computing, proposed by Avin, Borokhovich, Lotker, and Peleg [8]. This architecture is not described explicitly, but rather implicitly by providing three *axioms*. Specifically, a Core-Periphery network $G = (V, E)$ has its node set partitioned into a *core* C and a *periphery* P, and the three properties to be satisfied are then the following:

1. **Core boundary:** For every node $v \in C$, $\deg_C(v) \simeq {}^1 \deg_P(v)$, where, for $S \subseteq V$ and $v \in V$, $\deg_S(v)$ denotes the number of neighbors of v in S.
2. **Clique emulation:** the core can emulate the clique in a constant number of rounds in the CONGEST model. That is, there is a communication protocol running in a constant number of rounds in the CONGEST model such that, assuming that each node $v \in C$ has a message $M_{v,w}$ on $O(\log n)$ bits for every $w \in C$, then, after $O(1)$ rounds, every $w \in C$ has received all messages $M_{v,w}$, for all $v \in C$. In other words, the *all-to-all* communication pattern can be implemented in a constant number of rounds.
3. **Periphery-core convergecast:** there is a communication protocol running in a constant number of rounds in the CONGEST model such that, assuming that each node $v \in P$ has a message M_v on $O(\log n)$ bits, then, after $O(1)$ rounds, for every $v \in P$, at least one node in the core has received M_v.

Figure 2.2 provides an example of a Core-Periphery network, i.e., a graph satisfying the three axioms.

As shown in [8], these three axioms allow to design efficient distributed algorithms in the CONGEST model for classical problems such as matrix multiplication and Minimum Spanning Tree construction. Interestingly, it is shown that if only two out of three axioms were satisfied, then the round complexity of all the considered problems would increase quite significantly. For example, it is provided an algorithm for the Minimum Spanning Tree construction problem, that, in graphs that satisfy these three axioms, solves the task in $O(\log^2 n)$

[1] We assume $a \simeq b$ iff $\frac{a}{b+1} = \Theta(1)$

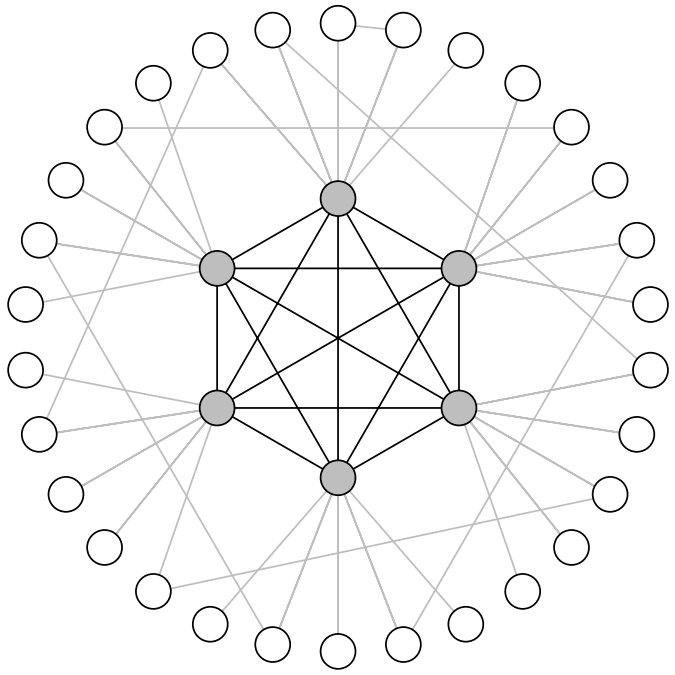

FIGURE 2.2: *Example of a Core-sPeriphery network, where the core (gray nodes) is a clique, and the periphery (white nodes) is a sparse graph.*

rounds, while, if only two out of three axioms were satisfied, then the Minimum Spanning Tree construction would require at least $\tilde{\Omega}(n^{\frac{1}{4}})$ rounds.

The Core-Periphery model provides an attractive alternative to the Congested Clique model. Indeed, the n-node Congested Clique has $\binom{n}{2}$ edges, while, assuming a core with, e.g., $O(\sqrt{n})$ nodes, even connecting all nodes in the core as a clique would only result in $O(n)$ edges in the core, a number that is much more manageable in practice, while still allowing efficient computation.

2.6 Minimum Spanning Tree

For the distributed Minimum Spanning Tree (MST) construction task, every node is given as input the weight $w(e)$ of each of its incident edges e. These weights are supposed to be of values polynomial in the size n of the network $G = (V, E, w)$, and thus each weight can be stored on $O(\log n)$ bits. The output of every node is a set of incident edges, such that the collection of all outputs forms an MST of the network. An MST is a subset of edges $T \subseteq E$ such that (V, T) is connected and $\sum_{e \in T} w(e)$ is minimum. At the end of the computation

each node must know which of its adjacent edges belong to the MST. In Figure 2.3 is depicted an example of MST.

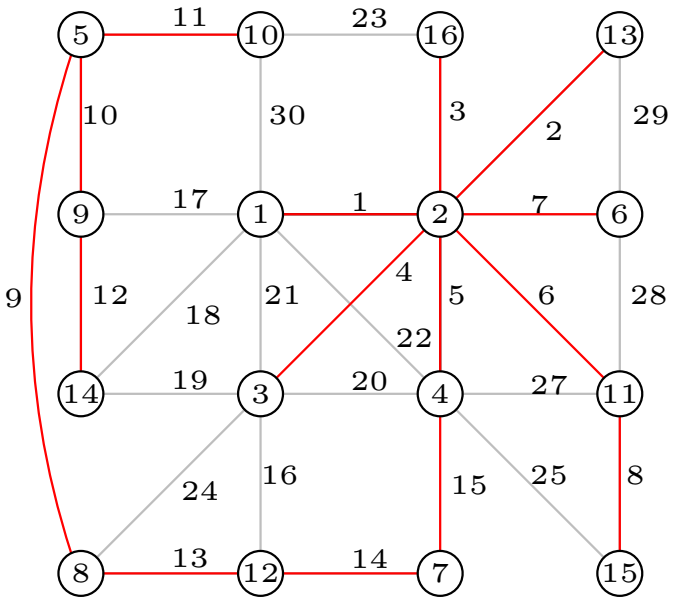

FIGURE 2.3: Example of a graph and its MST (in red).

The distributed MST construction problem has been widely studied. In the distributed asynchronous context, Gallager, Humblet and Spira [44] show an algorithm with time complexity of $O(n \log n)$ that uses $O(|E| + n \log n)$-bit messages, which is essentially optimal (under some assumptions we can actually break the $\Omega(|E|)$ barrier [60]). In the synchronous setting, the first sublinear algorithm was given by Garay et al. in [45]. Its running time is of $O(D + n^{\frac{\ln 3}{\ln 6}} \log^* n)$, that is approximately $O(D + n^{0.61} \log^* n)$ rounds, where D is the diameter of the graph. This complexity was later improved to $O(D + \sqrt{n} \log^* n)$ in [62]. Then, Peleg et al. [82] showed that this latter complexity is nearly optimal, giving an $\Omega(D + \frac{\sqrt{n}}{\log n})$ lower bound, which was improved by Das Sarma et al. [22] to $\Omega(D + \sqrt{\frac{n}{\log n}})$ and then by Ookawa et al. [77] to $\Omega(D + \sqrt{n})$. All these lower bounds hold for graphs with diameter $\Omega(\log n)$. For constant diameter graphs, there is a bound of $\widetilde{\Omega}(n^{1/3})$ rounds for diameter 4, a bound of $\widetilde{\Omega}(n^{1/4})$ rounds for diameter 3, and a bound of $O(\log n)$ rounds for diameter 2 (see [70]). Then, Elkin [25] showed that, if termination detection is not required, the diameter of the graph is not a lower bound, and that there exists an algorithm that requires $\widetilde{O}(\mu + \sqrt{n})$ rounds, where μ is the so-called MST-radius of the graph. Then, Pandurangan et al. [78] showed a randomized algorithm that is able to construct a MST in $\widetilde{O}(D + \sqrt{n})$ rounds while using an optimal number of messages, $\widetilde{O}(|E|)$. Elkin [27] showed how to achieve the same result

deterministically. Notice that all the aforementioned algorithms use messages of size $B = O(\log n)$, while some of the lower bounds have been explicitly stated as a function of B. Among the various lower bounds, the best one that depends on B is the one of Das Sarma et al. [22], that is $\Omega(D + \sqrt{\frac{n}{B}})$. In the Congested Clique, the MST problem can be solved much faster. In [69] Lotker et al. provided a deterministic algorithm that runs in $O(\log \log n)$ rounds in the Congested Clique. Then, Hegeman et al. [51] gave a randomized algorithm that requires $O(\log \log \log n)$ rounds. This complexity was even reduced further to $O(\log^* n)$ in [47]. They also showed that, by using slightly bigger messages, the complexity becomes $O(1)$ rounds. Finally, Jurdzinski and Nowicki gave a constant time algorithm [59].

2.7 Single Source Shortest Path

In the distributed Single Source Shortest Path (SSSP) problem, given a node v, all nodes of the graph have to find their distance from v.

For this problem, a linear-time exact solution can be found using the Bellman & Ford Algorithm. The problem of finding an exact solution in sublinear time remained opened for years, until when Elkin [26] gave an algorithm that is able to solve the problem in $O((n \log n)^{\frac{5}{6}})$ rounds for $D = O(\sqrt{n \log n})$ and $O(D^{\frac{1}{3}} \cdot (n \log n)^{\frac{2}{3}})$ rounds for larger diameters. The problem of finding an approximate solution seems easier. In fact, Lenzen et al. [66] showed how to find an $O(\frac{1}{\epsilon})$ approximation in $\widetilde{O}(n^{\frac{1}{2}+\epsilon} + D)$ rounds. Then, Nanongkai [73] improved this result providing a randomized algorithm that finds a $(1+\epsilon)$-approximation in $\widetilde{O}(n^{\frac{1}{2}} \cdot D^{\frac{1}{4}} + D)$ rounds. Finally, Henzinger et al. [54] removed the multiplicative dependence on D, by giving a deterministic algorithm that solves the problem in $O(n^{\frac{1}{2}+o(1)} + D^{1+o(1)})$ rounds finding a $(1+o(1))-$approximation. Becker et al. [13] improved this complexity by showing how to find a $(1+\epsilon)$ approximation in $\widetilde{O}(\epsilon^{-O(1)} \cdot (n^{\frac{1}{2}} + D))$ rounds. This last algorithm matches the lower bound given by Das Sarma et al. [22], that is $\Omega(D + \sqrt{\frac{n}{B}})$, but all the aforementioned algorithms are for $B = O(\log n)$.

2.8 All Pairs Shortest Paths

In the distributed All Pairs Shortest Paths (APSP) problem, each node of the network needs to find its distance from all the other nodes.

Frischknecht et al. [43] showed that, in dense graphs, the diameter can not be computed in sublinear time (using small messages), by providing a lower bound of $\Omega(\frac{n}{B})$ rounds. This result implies a lower bound for the APSP problem as well. Then, Abboud et al. [1] provided an $\widetilde{\Omega}(n)$ lower bound for $B = O(\log n)$, even for sparse networks. Concerning upper bounds, Holzer et al. [55] showed how to solve the APSP problem deterministically, in $O(n)$ rounds, in unweighted graphs. Nanongkai [73] presented a randomized algorithm that finds a $(1 + o(1))$-approximation in $\widetilde{O}(n)$ rounds in the weighted case. Lenzen et al. [67] showed that a $(1 + \epsilon)$-approximate solution can be deterministically found in $O(\epsilon^{-2} \cdot n \log n)$ rounds.

Chapter 3

Subgraph Detection

In this chapter we study problems related to subgraph detection, providing algorithms able to detect the presence of any fixed tree as a subgraph of the communication graph, in the classical CONGEST model. Also, it is presented an algorithm able to detect the presence of more complex subgraphs, in the context of *property testing*. This chapter is based on results published in [39], where we show how to test the presence of any cycle C_k of constant size k, and results published in [32], where we push further the techniques of [39], by showing how to test the presence of patterns composed by a couple of nodes connected to a fixed tree in an arbitrary manner.

3.1 Introduction

Consider a fixed graph $H = (V(H), E(H))$, that could be for example a triangle or a clique of four nodes. A graph $G = (V(G), E(G))$ is said to be H-free if it does not contain H as a subgraph, where H is a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Detecting the presence of subgraphs, or deciding subgraph freeness, has been investigated in many frameworks, like classical sequential computing [2], parametrized complexity [72], streaming [16], property-testing [3], communication complexity [58] and quantum computing [7]. In the classical sequential computing, for the general problem of detecting whether a graph H is a subgraph of G, where both H and G are part of the input, the best know bound is exponential [84]. Faster algorithms for

special cases of graphs H and G are known. For example, if H is a k-node tree, and G is an n-node tree, then there is an $O(\frac{k^{3/2}}{\log k}n)$-time algorithm for deciding whether H is a subgraph of G [83]. Subgraph detection becomes solvable in polynomial time if H is fixed, and only G is part of the input. Moreover, for any fixed H, subgraph detection can be solved in linear time in planar graphs [29]. In the case of general graphs, but where $H = P_k$, the path of length k, subgraph detection can be solved in time $O(nk!)$ [72].

In the context of distributed computing, deciding H-freeness means that the nodes of a network should cooperate to decide whether H is a subgraph of G, satisfying the following constraint:

- if G is H-free then every node outputs accept;
- otherwise, at least one node outputs reject.

That is, G is H-free if and only if all nodes output accept.

H freeness has been widely studied in the CONGEST model, for various types of graph patterns (see, e.g., [17, 18, 23, 24, 57, 39, 40]). In particular, even for very simple graph patterns H, it has been observed that deciding H freeness may require a lot of bandwidth. For example, it has been shown in [24] that deciding C_4-freeness requires $\widetilde{\Omega}(\sqrt{n})$ rounds in n-node networks in the CONGEST model. The intuition behind this lower bound is that the bandwidth limitation prevents the nodes with high degree to send their list of neighbors on a single communication link, unless consuming a lot of rounds. The lower bound for C_4-freeness can be extended to larger cycles C_k, $k \geq 4$, obtaining a lower bound of $\Omega(\text{poly}(n))$ rounds, where the exponent of the polynomial in n depends on k [24]. Similar bounds hold also in the Broadcast Congested Clique model, and also for detecting cliques. Hence, not only "global" tasks such as Minimum-weight Spanning Tree [22, 62, 77], diameter [1, 43], and All-Pairs Shortest Paths [55, 67, 73] are bandwidth demanding, but also "local" tasks such as deciding H-freeness are bandwidth demanding, at least for some graph patterns H. Also triangle related problems seem hard: Izumi et al. [57] provided randomized algorithms for triangle detection and triangle listing in the CONGEST model, with round complexity $\widetilde{O}(n^{2/3})$ and $\widetilde{O}(n^{3/4})$, respectively, and established a lower bound of $\widetilde{\Omega}(n^{1/3})$ on the round complexity of triangle listing.

Subgraph detection has also been investigated in the Congested Clique model, a variant of the CONGEST model which separates the communication network (assumed to be a complete graph) from the input graph G. In [23], it is shown that, for every k-node graph H, deciding whether H is a subgraph of an n-node input graph G can be achieved in $\widetilde{O}(n^{1-2/k})$ rounds. Using an efficient implementation of parallel matrix multiplication algorithms in the Congested Clique, [18] improved the results in [23] for triangle detection (as well as for C_4-detection), via an algorithm running in $O(n^{0.158})$ rounds.

As shown before, subgraph detection in the distributed setting is hard, since there exist polynomial lower bounds based on communication complexity results. One way to overcome this difficulty is to relax the requirements. Instead of requiring that at least one node rejects if G contains a copy of H, we require that at least one node rejects if G is far from being H-free. This relaxation is inspired from the notion of property testing, that is used in the centralized setting to solve decision tasks using a sublinear number of queries on the input.

The property of H-freeness has been the subject of a lot of investigation in classical (i.e., sequential) property testing. In the *dense* model, most solutions exploit the *graph removal lemma*, which essentially states that, for every k-node graph H, and every $\epsilon > 0$, there exists $\delta > 0$ such that every n-node graph containing at most δn^k (induced) copies of H can be transformed into an (induced) H-free graph by deleting at most ϵn^2 edges. This lemma was first proved for the case $k = 3$, and later generalized to subgraphs H of any size [30], and further to induced subgraphs [3]. It is possible to exploit this lemma for testing the presence of any (induced or not) subgraph of constant size, in constant time. Notice that δ is a fast growing function of ϵ and k. The growth of the function was later improved in [5] under some assumptions. For more details on the graph removal lemma, see [20, 3, 30]. In the *sparse* model, subgraph detection is harder. Even detecting triangles requires $\Omega(n^{1/3})$ queries, and the best known upper bound is $O(n^{6/7})$ queries [4] (the $\Omega(n^{1/3})$ lower bound holds even for 2-sided error algorithms, and for detecting any non bipartite subgraph). There exists a faster tester for cycle-detection in graphs of constant degree, as cycle-freeness can be tested with a constant number of queries by a 2-sided error algorithm [48]. However, testing cycle-freeness using 1-sided error algorithms requires $\Omega(\sqrt{n})$ queries [21].

Distributed property testing has been introduced in [14], where authors propose a constant-time distributed algorithm for finding a linear-size ϵ-near clique, under the assumption that the graph contains a linear-size ϵ^3-near clique (an ϵ-near clique is a set of nodes where all but an ϵ fraction of pairs of nodes have edges between them). Then, [17] fully formalized the notion of distributed property testing in the CONGEST model. They show that, in the dense model, any tester for a *non-disjointed* property can be emulated in the distributed setting with just a quadratic slowdown, i.e., if a sequential tester makes q queries, then it can be converted into a distributed tester that performs in $O(q^2)$ rounds. This simulation exploits the fact that any dense tester can be converted to a tester that first chooses some nodes uniformly at random, gathers their edges, and then performs centralized analysis of the obtained data (see [49]). The same paper also provides distributed testers for triangle-freeness, cycle-freeness, and bipartiteness, in the sparse model, running in $O(1)$, $O(\log n)$, and $O(\text{polylog}\, n)$ rounds, respectively. Then, Fraigniaud et al. [40] extended this work, by showing that for every connected graph H of four vertices, H-freeness can be tested in constant time. However, the same paper shows that the techniques used for testing H-freeness for 4-node graphs H fail to test C_k-freeness or K_k-freeness in a constant number of rounds, whenever $k \geq 5$.

Distributed property testing fits into the larger framework of *distributed decision*. The seminal paper [74] was perhaps the first to identify the connection between the ability to locally check the correctness of a solution in a distributed manner, and the ability to design an efficient deterministic distributed algorithm for constructing a correct solution. Since then, there have been a huge amount of contributions aiming at studying variants of distributed decision, in the deterministic setting (see, e.g., [38]), the anonymous setting (see, e.g., [28]), the probabilistic setting (see, e.g., [34, 37]), the non-deterministic setting (see, e.g., [50, 61]), and even beyond (see, e.g., [9, 36]). We refer to [35] for a survey on distributed decision.

3.2 Our Goal

The main objective of this work is to better understand for which patterns H it is possible to efficiently decide if a graph is H-free. We focus our attention

to a generic set of H-freeness decision tasks which includes several instances deserving full interest on their own right. In particular, deciding P_k-freeness, where P_k denotes the k-node path, is directly related to the NP-hard problem of computing the longest path in a graph. Also, detecting the presence of large complete binary trees, or of large binomial trees, is of interest for implementing classical techniques used in the design of efficient parallel algorithms (see, e.g., [63]). Similarly, detecting large Polytrees in a Bayesian network might be used to check fast belief propagation [79]. Finally, as it will be shown in this work, detecting the presence of various forms of trees can be used to test the presence of graph patterns of interest in the framework of distributed property testing [17]. Hence, this work addresses the following question:

> For which trees T is it possible to decide T-freeness efficiently in the CONGEST model, that is, in a number of rounds independent from the size n of the underlying network?

At a first glance, deciding T-freeness for some given tree T may look simpler than detecting cycles, or even just deciding C_4-freeness. Indeed, the absence of cycles enables us to ignore the issue of checking that a path starts and ends at the *same* node. This constraint is bandwidth consuming because it requires maintaining all possible partial solutions corresponding to growing paths from all starting nodes. Indeed, discarding even just a few starting nodes may result in missing the unique cycle including these nodes. However, even deciding P_k-freeness requires us to overcome many obstacles. First, as mentioned before, finding a longest simple path in a graph is NP-hard, which implies that it is unlikely that an algorithm deciding P_k-freeness exists in the CONGEST model, with running time polynomial in k at every node. Second, and more importantly, there exists potentially up to $\Theta(n^k)$ paths of length k in a network, which makes impossible to maintain all of them in partial solutions, as the overall bandwidth of n-node networks is at most $O(n^2 \log n)$ in the CONGEST model.

We then study a relaxation of this problem, by trying to understand for which patterns H we can test H-freeness in the context of distributed property testing. It has been shown in [17] that, in the classical CONGEST model for distributed computing [80], there exists a distributed property testing algorithm for triangle-freeness performing in $O(1/\epsilon^2)$ rounds. This result has been extended in [40] where it is proved that there exists a distributed property testing

algorithm for C_4-freeness performing in $O(1/\epsilon^2)$ rounds as well. Perhaps surprisingly, the techniques in [17, 40] do not extend to larger cycles. Indeed, using explicit constructions of so-called Behrend graphs, it was proved in [40] that these techniques fail for most values of $k \geq 5$. That is, these techniques cannot result in a tester whose runtime is constant in all graphs, even if the constant is allowed to be a function of $1/\epsilon$. The existence of distributed property testing algorithms which decide H-freeness in a constant number of rounds was left open for graphs having more than 4 nodes. Hence, we address the following question:

> For which graph patterns H is it possible to test H-freeness efficiently in the context of distributed property testing?

3.3 Results

We show that, in contrast to C_k-freeness, P_k-freeness can be decided in a constant number of rounds, for any $k \geq 1$. In fact, our main result is far more general, as it applies to *any* tree. Stated informally, we prove the following:

Theorem 3.1. *For every tree T, there exists a deterministic algorithm for* deciding *T-freeness in a constant number of rounds under the* CONGEST *model.*

Notice that, if we do not restrict the amount of bandwidth, then the problem becomes easily solvable by just gathering the h-radius neighborhood, where h is the height of T, and then checking everything locally. A different approach could be the following. Each node v can start broadcasting, for every possible leaf l of T, that v is a potential candidate to be the node l. Then, at each round, each node considers all the possible subtrees of T, and sees which of them can be constructed by merging some subtrees received from its neighbors at the previous round (notice that they must be disjoint), and setting itself to be the root of the subtree. At last, each node sends all the possible valid subtrees that was able to construct. By repeating this process a number of rounds proportional to the height of T, nodes can detect the presence of T.

The main obstacle for implementing this algorithm in the CONGEST model is that even collecting the identities of the nodes at distance 2 from a given node

u might be impossible to achieve in $o(n)$ rounds in n-node network. Indeed, u may have constant degree, with $\Omega(n)$ neighbors at distance 2 (e.g., node 7 in the lollipop graph of Figure 3.1). To overcome this difficulty, we proceed by

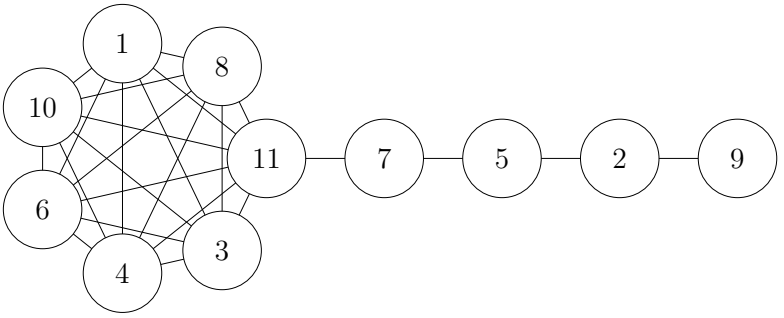

FIGURE 3.1: A lollipop graph.

pruning the set of information transmitted between nodes. This pruning is at the risk of discarding candidates that would have turned out to be actual trees. Nevertheless, our pruning mechanism guarantees that at least one actual tree remains in the current set of candidates throughout the execution of the algorithm. In fact, we present a distributed implementation of a pruning technique based on a combinatorial result due to Erdős et al. [31] that roughly states the following. Let $k > t > 0$. For any set V of n elements, and any collection F of subsets of V, all with cardinality at most t, let us define a *witness* of F as a collection $\widehat{F} \subseteq F$ of subsets of V such that, for any $X \subseteq V$ with $|X| \leq k - t$, the following holds:

$$\big(\exists Y \in F : Y \cap X = \emptyset\big) \implies \big(\exists \widehat{Y} \in \widehat{F} : \widehat{Y} \cap X = \emptyset\big).$$

Of course, every F is a witness of itself. However, Erdős et al. have shown that, for every k, t, and F, there exists a *compact* witness $\widehat{F}$ of F, that is, a witness whose cardinality depends on k and t only, and hence is independent of n. To see why this result is important for detecting a tree T in a network G, consider V as the set of nodes of G, k as the number of nodes in T, and F as a collection of subtrees Y of size at most t, each isomorphic to some subtree of T. The existence of compact witnesses allows an algorithm to keep track of only a small subset $\widehat{F}$ of F. Indeed, if F contains a partial solution Y that can be extended into a global solution isomorphic to T using a set of nodes X, then there is a *representative* $\widehat{Y} \in \widehat{F}$ of the partial solution $Y \in F$ that can also be extended into a global solution isomorphic to T using the same set X of nodes. Therefore, there is no need to keep track of all partial solutions $Y \in F$, it is sufficient to keep track of

just the partial solutions $\widehat{Y} \in \widehat{F}$. This pruning technique has been successfully used for designing fixed-parameter tractable (FPT) algorithms for the longest path problem [72].

Theorem 3.1, which establishes the existence of distributed algorithms for detecting the presence of trees, has important consequences on the ability to *test* the presence of more complex graph patterns in the context of *distributed property-testing*. Recall that, for $\epsilon \in (0,1)$, a graph G is ϵ-far from being H-free if removing less than a fraction ϵ of its edges cannot result in an H-free graph. In fact, we obtain the following result.

Theorem 3.2. *For every graph pattern H composed of an edge and a tree with arbitrary connections between them, there exists a (randomized) distributed algorithm for* testing *H-freeness in a constant number of rounds under the* CONGEST *model.*

At a first glance, the family of graph patterns H composed of an edge and a tree with arbitrary connections between them (like, e.g., the graph depicted on the top-left corner of Fig. 3.2) may look quite specific and artificial. This is not the case. For instance, every cycle C_k for $k \geq 3$ is a "tree plus one edge". This also holds for 4-node complete graph K_4. In fact, all known results about testing H-freeness for some graph H in [17, 39, 40] are just direct consequence of Theorem 3.2. Moreover, Theorem 3.2 enables us to test the presence of other graph patterns, like the cycle C_k of length k, the complete bipartite graph $K_{2,k}$ with $k+2$ nodes, for every $k \geq 1$, or the graph pattern depicted on the top-right corner of Fig. 3.2, in $O(1)$ rounds. It also enables us to test the presence of connected 1-factors as a subgraph in $O(1)$ rounds. (Recall that a graph H is a 1-factor if its edges can be directed so that every node has out-degree 1). In fact, our algorithm is 1-sided, that is, if G is H-free, then all nodes output accept with probability 1.

Also, by carefully combining the previous results we can easily obtain the following:

Theorem 3.3. *For every graph pattern H composed of a node and a tree with arbitrary connections between them, there exists a distributed algorithm for* deciding *H-freeness performing in $O(n)$ rounds under the* CONGEST *model.*

All our results are summarized on Table 3.1, together with some of the previous work in the literature. These results appeared in [39] and [32]. In the former

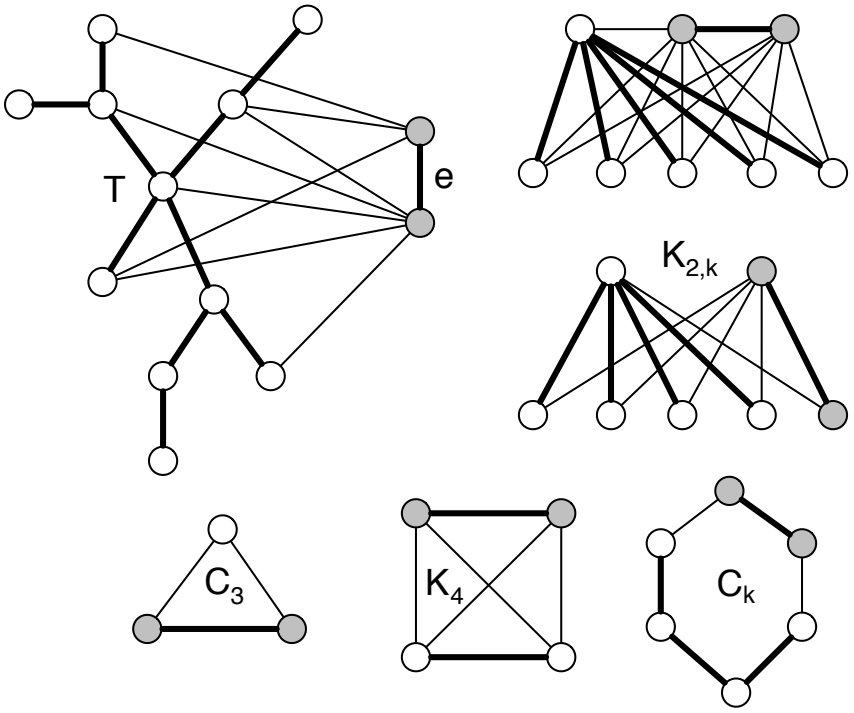

FIGURE 3.2: All these graphs are composed of a tree T and edge e with arbitrary connections between them.

we presented a tester for C_k freeness, for any constant $k \geq 3$, by providing a distributed implementation of a result of Erdős et al. [31]. In the latter we extended the result of [39], by showing the results stated in Theorems 3.1 and 3.2.

3.4 Detecting the Presence of Trees

In this section we establish our main result, i.e., Theorem 3.1, stated formally below as Theorem 3.5. As a warm up, we first show a simple and elegant randomized algorithm for deciding T-freeness, for every given tree T, running in $O(1)$ rounds under the CONGEST model. Next, we show an algorithm that achieves the same runtime, but deterministically.

	Distributed detection	Distr. property testing
Cycles C_k	$\Omega(\text{poly}(n))$ for $k \geq 4$ [24] O(n) **[this work]**	$O(1)$ [17, 40] for $k \leq 4$ O(1) **[this work]**
Cliques K_k	$\widetilde{O}(n^{2/3})$ for $K_3 = C_3$ [57] open for $k \geq 4$	$O(1)$ for K_3 & K_4 [17, 40] open for $k \geq 5$
Trees	$O(1)$ **[this work]**	$O(1)$ **[this work]**
Trees-plus-one-node	$O(n)$ **[this work]**	$O(1)$ **[this work]**
Trees-plus-one-edge	$\widetilde{\Theta}(\sqrt{n})$ for C_4 [24]	$O(1)$ **[this work]**
Large pseudo-cliques	open	$O(1)$ [14]

TABLE 3.1: Number of rounds for deciding H-freeness in the CONGEST model

A simple randomized algorithm

Theorem 3.4. *For every tree T of constant size, there exists a 1-sided error randomized algorithm performing in $\mathcal{O}(1)$ rounds in the* CONGEST *model, which correctly detects if the given input network contains T as a subgraph, with probability at least $2/3$.*

Proof. The algorithm performs in a sequence of phases. Algorithm 1 displays a phase of the algorithm.

Let k be the number of vertices of tree T, i.e., $k = |V(T)|$. Pick an arbitrary vertex of T, and root T at that node. The root is labeled k. Then, label the rest of the nodes of T in decreasing order according to the order obtained from a BFS traversal starting from the root. For $i \in [1, k]$, let T_i be the subtree of T rooted at the node labeled i. Let $\text{child}(i)$ denote the labels of all the nodes adjacent to i in T_i (i.e., the labels of all the children of i in T). We use the color coding technique introduced in [6] in the context of (classical) property testing. Each vertex u of G picks a *color* in $[1, k]$ uniformly at random. We say that G is *well colored* if at

least one of the subgraphs T' of G that is isomorphic to T satisfies that the colors of T' correspond to the labels of the nodes in T. (Note that if G is T-free then G is well colored, no matter the coloring).

In the verification algorithm, every vertex u is either *active* or *inactive*, which is represented by a variable active$(u) \in \{\text{true}, \text{false}\}$. Initially, every node u is inactive (i.e., active$(u) = $ false). Intuitively, a node u becomes active if it has detected that the graph contains the tree T_c as subgraph, rooted at u, where c is the color of u. More precisely, once every node has picked a color in $[1, k]$ u.a.r., all nodes exchange their colors between neighbors. Then Algorithm 1 performs k rounds. At the beginning of each round, every node v communicates active(v) to all its neighbors. In round c, $1 \leq c \leq k$, each node u with color c checks whether, for each color c' of its children, some neighbor v is colored c' and is active. If that is the case, it becomes active, otherwise it remains inactive.

Algorithm 1 Randomized tree-detection, for a given tree T. Algorithm executed by node u.

send ID(u) to all neighbors, and receive ID(v) from every neighbor v
let $k = |V(T)|$, and pick color$(u) \in [k]$ uniformly at random
send color(u) to all neighbors, and receive color(v) from every neighbor v
for every $c \in [1, k]$, let $N_c(u) = \{v \in N(u) \mid \text{color}(v) = c\}$
active$(u) \leftarrow$ false
for $c = 1$ **to** k **do**
    send active(u) to all neighbors, and receive active(v) from every neighbor v
    compute $A(u) = \{v \in N(u) \mid \text{active}(v) = \text{true}\}$
    if color$(u) = c$ **and** $\left(\forall c' \in \text{child}(c), N_{c'}(u) \cap A(u) \neq \emptyset\right)$ **then**
        active$(u) \leftarrow$ true
    end if
end for
if color$(u) = k$ **and** active$(u) =$ true **then**
    output reject
else
    output accept
end if

We claim that a well colored graph G contains T as a subgraph if and only if a vertex colored k becomes active at round k. To establish that claim, note first that, if $c \in [1, k]$ is a leaf of T, then the tree T_c is detected on round c, by every node colored c. Suppose now that, for every $c' < c$, the fact that a node u colored c' becomes active at round c' means that u has detected $T_{c'}$. Let $c_1, \ldots, c_r$

be children of c in T_c. A node u colored c becomes active at round c if and only, for every $i, 1 \leq i \leq r$, it holds that u has an active neighbor colored c_i. From the construction of the labels of T, and from the induction hypothesis, this implies that u becomes active at round c if and only if u has detected T_c. We conclude that a node colored k becomes active at round k if and only if T is detected in G, as $T = T_k$.

Now, if G contains T as a subgraph, then the probability that G is well colored is at least $(1/k)^k$. Therefore, we run $\mathcal{O}(k^k)$ independent iterations of Algorithm 1, which yields that, with probability at least $2/3$, G is well colored for at least one iteration. □

Deterministic algorithm

In this section, we establish our main result:

Theorem 3.5. *For every tree T of constant size, there exists an algorithm performing in $O(1)$ rounds in the* CONGEST *model for detecting whether the given input network contains T as a subgraph.*

Proof. Let k be the number of nodes in tree T. The nodes of T are labeled arbitrarily by k distinct integers in $[1, k]$. We arbitrarily choose a vertex $r \in [1, k]$ of T, and view T as rooted in r. For any vertex $\ell \in V(T)$, let T_ℓ be the subtree of T rooted in ℓ. We say that T_ℓ is a *shape* of T. Our algorithm deciding T-freeness proceeds in $\text{depth}(T_r) + 1$ rounds. At round t, every node u of G constructs, for each shape T_ℓ of depth at most t, a set of subtrees of G all rooted at u, denoted by $\text{SOS}_u(T_\ell)$, such that each subtree in $\text{SOS}_u(T_\ell)$ is isomorphic to the shape T_ℓ. The isomorphism is considered in the sense of rooted trees, i.e., it maps u to ℓ. If we were in the LOCAL model, we could afford to construct the set of all such subtrees of G. However, we cannot do that in the CONGEST model because there are too many such subtrees. Therefore, the algorithm acts in a way which guarantees that:

1. the set $\text{SOS}_u(T_\ell)$ is of constant size, for every node u of G, and every node ℓ of T;

2. for every set $C \subseteq V$ of size at most $k - |V(T_\ell)|$, if there is some subtree W of G rooted at u that is isomorphic to T_ℓ, and that is not intersecting C, then $\text{SOS}_u(T_l)$ contains at least one such subtree W' not intersecting C. (Note that W' might be different from W).

The intuition for the second condition is the following. Assume that there exists some subtree W of G rooted at u, corresponding to some shape T_ℓ, which can be extended into a subtree isomorphic to T by adding the vertices of a set C. The algorithm may well not keep the subtree W in $\text{SOS}_u(T_l)$. However, we systematically keep at least one subtree W' of G, also rooted at u and isomorphic to T_ℓ, that is also extendable to T by adding the vertices of C. Therefore the sets $\text{SOS}_u(T_\ell)$, over all shapes T_ℓ of depth at most t, are sufficient to ensure that the algorithm can detect a copy of T in G, if it exists. Our approach is described in Algorithm 2. (Observe that, in this algorithm, if we omit Lines 17 to 19, which prune the set $\text{SOS}_u(T_\ell)$, we obtain a simple algorithm detecting T in the LOCAL model, where no bandwidth restriction is imposed). Implementing the pruning of the sets $\text{SOS}_u(T_\ell)$ for keeping them compact, we make use of the following combinatorial lemma, which has been rediscovered several times, under various forms (see, e.g., [72]).

Lemma 3.6 (Erdős, Hajnal, Moon [31])**.** *Let V be a set of size n, and consider two integer parameters p and q. For any set $F \subseteq \mathcal{P}(V)$ of subsets of size at most p of V, there exists a* compact (p, q)-representation *of F, i.e., a subset $\hat{F}$ of F satisfying:*

1. *For each set $C \subseteq V$ of size at most q, if there is a set $L \in F$ such that $L \cap C = \emptyset$, then there also exists $\hat{L} \in \hat{F}$ such that $\hat{L} \cap C = \emptyset$;*
2. *The cardinality of $\hat{F}$ is at most $\binom{p+q}{p}$, for any $n \geq p + q$.*

By Lemma 3.6, the sets $\text{SOS}_u(T_\ell)$ can be reduced to constant size (i.e., independent of n), for every shape T_ℓ and every node u of G. Moreover, the number of shapes is at most k, and, for each shape T_ℓ, each element of $\text{SOS}_u(T_\ell)$ can be encoded on $k \log n$ bits. Therefore each vertex communicates only $O(\log n)$ bits per round along each of its incident edges. So, the algorithm does perform in $O(1)$ rounds in the CONGEST model[1].

[1]We may assume that, for compacting a set $\text{SOS}_u(T_\ell)$ in Lines 17-19, every node u applies Lemma 3.6 by brute force (e.g., by testing all candidates $\hat{F}$). In [72], an algorithmic version of Lemma 3.6 is proposed, producing a set $\hat{F}$ of size at most $\sum_{i=1}^{q} p^i$ in time $O((p+q)! \cdot n^3)$, i.e., in time $\text{poly}(n)$ for fixed p and q.

Algorithm 2 Tree-detection, for a given tree T. Algorithm executed by node u.

for each leaf ℓ of T **do**
  let $\text{SOS}_u(T_\ell)$ be the unique tree with single vertex u
  exchange the sets SOS with all neighbors
end for
for $t = 1$ **to** $\text{depth}(T)$ **do**
  for each node ℓ of T with $\text{depth}(T_\ell) = t$ **do**
    $\text{SOS}_u(T_\ell) \leftarrow \emptyset$
    let $j_1, \ldots, j_s$ be the children of ℓ in T
    for every s-uple $(v_1, \ldots, v_s)$ of nodes in $N(u)$ **do**
      for every $(W_1, \ldots, W_s) \in \text{SOS}_{v_1}(T_{j_1}) \times \cdots \times \text{SOS}_{v_s}(T_{j_s})$ **do**
        if $\{u\}$ and $W_1, \ldots, W_s$ are pairwise disjoint **then**
          let W be the tree with root u, and subtrees $W_1, \ldots, W_s$
          add W to $\text{SOS}_u(T_\ell)$ ▷ *each W_i is glued to u by its root*
        end if
      end for
    end for
    let $F = \{V(W) \mid W \in \text{SOS}_u(T_l)\}$ ▷ *collection of vertex sets for trees in* $\text{SOS}_u(T_l)$
    construct a $(|V(T_\ell)|, k - |V(T_\ell)|)$-compact representation $\hat{F} \subseteq F$ ▷ *cf. Lemma 3.6*
    remove from $\text{SOS}_u(T_\ell)$ all trees W with vertex set not in $\hat{F}$
    exchange $\text{SOS}_u(T_\ell)$ with all neighbors
  end for
end for
if $\text{SOS}_u(T_r) = \emptyset$ **then** ▷ *r denotes the root of T*
  accept
else
  reject
end if

Proof of correctness. First, observe that if $\text{SOS}_u(T_\ell)$ contains a graph W, then W is indeed a tree rooted at u, and isomorphic to T_ℓ. This is indeed the case at round $t = 0$, and we can proceed by induction on t. Let T_ℓ be a shape of depth ℓ. Each graph W added to $\text{SOS}_u(T_\ell)$ is obtained by gluing vertex-disjoint trees at the root u. These latter trees are isomorphic to the shapes $T_{j_1}, \ldots, T_{j_s}$, where $j_1, \ldots, j_s$ are the children of node j in T. Therefore W is isomorphic to T_ℓ. In particular, if the algorithm rejects at some node u, it means that there exists a subtree of G isomorphic to T.

We now show that if G contains a subgraph W isomorphic to T, then the algorithm rejects in at least one node. For this purpose, we prove a stronger statement:

Lemma 3.7. *Let u be a node of G, T_ℓ be a shape of T, and C be a subset of vertices of G, with $|C| \leq k - |V(T_u)|$. Let us assume that there exists a subgraph W_u of G, satisfying the following two conditions: (1) W_u is isomorphic to T_ℓ, and the isomorphism maps u on ℓ, and (2) W_u does not contain any vertex of C. Then $\text{SOS}_u(T_\ell)$ contains a tree W'_u satisfying these two conditions.*

We prove the lemma by induction on the depth of T_ℓ. If $\text{depth}(T_\ell) = 0$ then ℓ is a leaf of T_ℓ, and $\text{SOS}_u(T_\ell)$ just contains the tree formed by the unique vertex u. Il particular, it satisfies the claim. Assume now that the claim is true for any node of T whose subtree has depth at most $t-1$, and let ℓ be a node of depth t. Let $j_1, \ldots, j_s$ be the children of ℓ in T. For every i, $1 \leq i \leq s$, let v_i be the vertex of W_u mapped on j_i. By induction hypothesis, $\text{SOS}_{v_1}(T_{j_1})$ contains some tree W'_{v_1} isomorphic to T_{j_1} and avoiding the nodes in $C \cup \{u\}$, as well as all the nodes of $W_{v_2}, \ldots W_{v_s}$. Using the same arguments, we proceed by increasing values of $i = 2, \ldots, s$, and we choose a tree $W'_{v_i} \in \text{SOS}_{v_i}(T_{j_i})$ isomorphic to T_{j_i} that avoids $C \cup \{u\}$, as well as all the nodes in $W'_{v_1}, \ldots, W'_{v_{i-1}}$ and the nodes of $W_{v_{i+1}}, \ldots, W_{v_s}$. Now, observe that the tree W'' obtained from gluing u to $W'_{v_1}, \ldots, W'_{v_s}$ has been added to $\text{SOS}_u(T_\ell)$ before compacting this set, by Line 12 of Algorithm 2. Since W'' does not intersect C, we get that, by compacting the set $\text{SOS}_u(T_\ell)$ using Lemma 3.6, the algorithm keeps a representative subtree W' of G that is isomorphic to T_l and not intersecting C. This completes the proof of the lemma. ◇

To complete the proof of Theorem 3.5, let us assume there exists a subtree W of G isomorphic to T, and let u be the vertex that is mapped to the root r of T by this isomorphism. By Lemma 3.7, $\text{SOS}_u(T_r) \neq \emptyset$, and thus the algorithm rejects at node u. □

3.5 Distributed Property Testing

In this section, we show how to construct a distributed tester for H-freeness in the sparse model, based on Algorithm 2. This tester is able to test the presence of every graph pattern H composed of an edge e and a tree T connected in an arbitrary manner, by distinguishing graphs that include H from graphs that are ϵ-far from being H-free.

Specifically, we consider the set $\mathcal{H}$ of all graph patterns H with node-set $V(H) = \{x, y, z_1, \ldots, z_k\}$ for $k \geq 1$, and edge-set $E(H) = \{f\} \cup E(T) \cup \mathcal{E}$, where $f = \{x, y\}$, T is a tree with node set $\{z_1, \ldots, z_k\}$, and $\mathcal{E}$ is some non-empty set of edges with one end-point equal to x or y, and the other end-point z_i for $i \in \{1, \ldots, k\}$. Hence, a graph $H \in \mathcal{H}$ can be described by a triple $(f, T, \mathcal{E})$ where $\mathcal{E}$ is a set of edges connecting a node in T with a node in f.

We now establish our second main result, i.e., Theorem 3.2, stated formally below as follows:

Theorem 3.8. *For every graph pattern $H \in \mathcal{H}$, i.e., composed of an edge and a tree of constant size connected in an arbitrary manner, there exists a randomized 1-sided error distributed property testing algorithm for H-freeness performing in $O(1/\epsilon)$ rounds in the* CONGEST *model.*

Proof. Let $H = (f, T, \mathcal{E})$, with $f = \{x, y\}$. Let us assume that there are ν copies of H in G, and let us call these copies $H_1 = (f_1, T_1, \mathcal{E}_1), \ldots, H_\nu = (f_\nu, T_\nu, \mathcal{E}_\nu))$. Let $\mathbf{E} = \{f_1, \ldots, f_\nu\}$. Our tester algorithm for H-freeness is composed by the following two phases:

1. determine a candidate edge e susceptible to belong to $\mathbf{E}$;
2. checking the existence of a tree T connected to e in the desired way.

In order to find the candidate edge, we exploit the following lemma:

Lemma 3.9 ([40])**.** *Let H be any graph. Let G be an m-edge graph that is ϵ-far from being H-free. Then G contains at least $\epsilon m/|E(H)|$ edge-disjoint copies of H.*

Hence, if the actual m-edge graph G is ϵ-far from being H-free, we have $|\mathbf{E}| \geq \epsilon m/|E(H)|$. Thus, by randomly choosing an edge e and applying Lemma 3.9, $e \in \mathbf{E}$ with probability at least $\epsilon/|E(H)|$.

The first phase can be computed in the following way. First, every edge is assigned to the endpoint having the smallest identifier. Then, every node picks a random integer $r(e) \in [1, m^2]$ for each edge e assigned to it. The candidate edge of Phase 1 is the edge e_{min} with minimum rank, and indeed $\Pr[e_{min} \in \mathbf{E}] \geq \epsilon/|E(H)|$.

It might be the case that e_{min} is not unique though. However, the following Lemma holds.

Lemma 3.10. $\Pr[e_{min}$ *is unique*$] \geq 1/e^2$, *where e denotes here the basis of the natural logarithm.*

Proof. The probability that there are no collisions while choosing for each edge a random number from $[1, m^2]$ is

$$\frac{m^2-1}{m^2} \times \ldots \times \frac{m^2-m}{m^2} \geq \left(\frac{m^2-m}{m^2}\right)^m$$

$$= \left(1-\frac{1}{m}\right)^m \geq \left(e^{\frac{-2}{m}}\right)^m = \frac{1}{e^2}$$

where the last inequality holds whenever $m \geq 2$. □

Also, every node picks, for every edge $e = \{v_1, v_2\}$ assigned to it, a random bit b. Assume, w.l.o.g., that $\mathrm{ID}(v_1) < \mathrm{ID}(v_2)$. If $b = 0$, then the algorithm will start Phase 2 for testing the presence of H with $(x, y) = (v_1, v_2)$, and if $b = 1$, then the algorithm will start Phase 2 for testing the presence of H with $(x, y) = (v_2, v_1)$. We have $\Pr[e_{min}$ is considered in the right order$] \geq 1/2$. It follows that the probability e_{min} is unique, considered in the right order, and part of $\mathbf{E}$ is at least $\frac{\epsilon}{2|E(H)|e^2}$.

Using a deterministic search based on Algorithm 2, H will be found with probability at least $\frac{\epsilon}{2|E(H)|e^2}$. To boost the probability of detecting H in a graph that is ϵ-far from being H-free, we repeat the search $2e^2|E(H)|\ln 3/\epsilon$ times. In this way, the probability that H is detected in at least one search is at least $2/3$ as desired.

During the second phase, the ideal scenario would be that all the nodes of G search for $H = (f, T, \mathcal{E})$ by considering only the edge e_{min} as candidate for f, to avoid congestion. Obviously, making all nodes aware of e_{min} would require diameter time. However, there is no needs to do so. Indeed, the tree-detection algorithm used in the proof of Theorem 3.5 runs in $depth(T)$ rounds. Hence, since only the nodes at distance at most $depth(T) + 1$ from the endpoints of e_{min} are able to detect T, it is enough to broadcast e_{min} at distance up to $2\,(depth(T) + 1)$

rounds. This guarantees that all nodes participating to the execution of the algorithm for e_{min} will see the same messages, and will perform the same operations that they would perform by executing the algorithm for e_{min} on the full graph.

So, every node broadcasts its candidate edges with the minimum rank, at distance $2\,(depth(T)+1)$. Two contending broadcasts, for two candidate edges e and e' for f, resolve contention by discarding the broadcast corresponding to the edge e or e' with largest rank. (If e and e' have the same rank, then both broadcast are discarded). After this is done, every node is assigned to one specific candidate edge, and starts searching for T. Similarly to the broadcast phase, two contending searches, for two candidate edges e and e', resolve contention by aborting the search corresponding to the edge e or e' with largest rank. From now on, one can assume that a single search in running, for the candidate edge e_{min}.

It remains to show how to adapt Algorithm 2 for checking the presence of a tree T connected to a *fixed* edge $e = \{x, y\} \in E(G)$ as specified in $\mathcal{E}$. Let us consider Instruction 6 of Algorithm 2, that is: "**for** each node ℓ of T with $\text{depth}(T_\ell) = t$ **do**". At each step of this for-loop, node u tries to construct a tree W that is isomorphic to the subtree of T rooted at ℓ. In order for u to add W to $\text{SOS}_u(T_\ell)$, we add the condition that:

- if $\{\ell, x\} \in E(H)$ then $\{u, x\} \in E(G)$, and
- if $\{\ell, y\} \in E(H)$ then $\{u, y\} \in E(G)$.

Note that this condition can be checked by every node u. If this condition is not satisfied, then u sets $\text{SOS}_u(T_\ell) = \emptyset$.

This modification enables us to test H-freeness. Indeed, if the actual graph G is H-free, then, since at each step of the modified algorithm, the set $\text{SOS}_u(T_\ell)$ is a subset of the set $\text{SOS}_u(T_\ell)$ generated by the original algorithm, the acceptance of the modified algorithm is guaranteed from the correctness of the original algorithm.

Conversely, let us show that, in a graph G that is ϵ far of being H-free, the algorithm rejects G as desired. In the first phase of the algorithm, it holds that $e_{min} \in \mathbf{E}$ happens in at least one search whenever G is ϵ-far from being H-free, with probability at least $2/3$. Following the same reasoning of the proof of

Lemma 3.7, since the images of the isomorphism satisfy the condition of being linked to nodes $\{x, y\}$ in the desired way, the node of G that is mapped to the root of T correctly detects T, and rejects, as desired. □

3.6 Conclusions

In this work, we have proposed a generic construction for designing deterministic distributed algorithms detecting the presence of any given tree T as a subgraph of the input network, performing in a constant number of rounds in the CONGEST model. Therefore, there is a clear dichotomy between cycles and trees, as far as efficiently solving H-freeness is concerned: while every cycle of at least four nodes requires at least a polynomial number of rounds to be detected, every tree can be detected in a constant number of rounds. It is not clear whether one can provide a simple characterization of the graph patterns H for which H-freeness can be decided in $O(1)$ rounds in the CONGEST model. An intriguing question is to determine the round-complexity of deciding K_k-freeness in the CONGEST model for $k \geq 3$, and in particular to determine the exact round-complexity of deciding C_3-freeness.

Our construction also provides randomized algorithms for testing H-freeness (i.e., for distinguishing H-free graphs from graphs that are far from being H-free), for every graph pattern H that can be decomposed into an edge and a tree, with arbitrary connections between them, also running in $O(1)$ rounds in the CONGEST model. This generalizes the results in [17, 40], where algorithms for testing H-freeness for every H of at most 4 nodes were provided. Interestingly, K_5 is the smallest graph pattern H for which it is not known whether testing H-freeness can be done in $O(1)$ rounds, and this is also the smallest graph pattern that cannot be decomposed into a tree plus an edge. We do not know whether this is just coincidental or not.

Chapter 4

Tradeoffs Between Bandwidth and Time

In this chapter we address some questions related to time complexity in the CONGEST model. More precisely, since often upper bounds are given only for the case where the bandwidth is constrained to be $O(\log n)$, we analyze how the time complexity of existing algorithms can *scale* when more bandwidth is allowed. We show that the complexity of different problems can scale in different ways, and that in some limit cases, up to some point, more bandwidth does not help at all. This chapter is based on currently unpublished results present in [76].

4.1 Introduction

The links of networks typically have limited bandwidth. The CONGEST model for distributed network computing captures this constraint. In this model, an algorithm proceeds in synchronous rounds. At each round, every node can send B bits to each of its neighbors in the network (these B bits do not need to be the same for all neighbors). A typical value for B is $O(\log n)$ in n-node networks. This value is sufficient to transmit an integer in a polynomial range, like the identifier of a node, or the weight of an edge.

One celebrated result in this context is a Minimum-weight Spanning Tree (MST) construction algorithm that performs in $O(D + \sqrt{n}\log^* n)$ rounds in diameter-D n-node networks [62]. This complexity is optimal for $B = O(\log n)$ [77]. On the other hand, all (computable) tasks can be solved in $O(D)$ rounds in diameter-D networks whenever there is no limitation on the bandwidth, by gathering all data at one node, computing the solution at that node, and broadcasting that solution to all nodes.

The aim of this work is to investigate tradeoffs between the round complexity for solving a task, and the bandwidth of the links. So far, results are known only for $B = O(\log n)$ in the classical CONGEST model, and $B = +\infty$ in the so-called LOCAL model. A specific case that deserves particular interest is to determine, given a task, the minimum value for B such that the task is solvable in $O(D)$ rounds in diameter-D networks with links of bandwidth B. For instance, in the case of the MST construction task, what is the minimum value of B such that MST can be constructed in $O(D)$ rounds? It is known that $B = O(\log n)$ is not sufficient, as $\Omega(D + \sqrt{n})$ rounds is a lower bound on the number of rounds for such a value of B [77].

4.2 Our Goal

In this work, our objective is twofold. First, we are aiming at establishing tradeoffs between the size of the messages B and the number of rounds, by obtaining algorithms for the CONGEST$_B$ model, having round complexities that are parametric on B. Our second objective is to better understand the role of B in the CONGEST$_B$ model, and try to figure out if B always has some impact on the round complexity.

We start by analyzing the round complexity of various problems, namely MST, SSSP, and APSP. The upper bounds for these problems have been studied only in the case where the size of the messages is $B = O(\log n)$, while lower bounds have been typically given as a function of B. Thus, our goal is to better understand what is the time complexity of these problems on a wider spectrum of bandwidths. In fact, we provide new time complexities that depend on the bandwidth parameter B.

Then, we try to understand if all problems can benefit from the presence of more bandwidth. Consider two nodes that are at a distance that is equal to the diameter of the graph. Trivially, if they want to share just a bit, they need to wait a time proportional to the diameter of the graph, and even by allowing more bandwidth they can not solve the problem faster. We want to better understand this phenomenon and address the following question: are there problems that can not benefit from the presence of more bandwidth, possibly unrelated to trivial bounds related to distances?

4.3 Results

We notice that different problems exhibit a different behavior when analyzing how their time complexity scale with the allowed bandwidth.

For global problems, a lower bound of $\Omega(D)$ holds, where D is the diameter of the network. Thus, a problem can be typically solved faster using more bandwidth, up to some point, where other parameters, like D, start to dominate the time complexity. The goal is to measure how much the speed of an algorithm can scale before reaching this point. In order to do so, we consider $T_P(\mathcal{X})$, defined to be the exact round complexity of a problem P using messages of size $B = O(\mathcal{X} \log n)$, and we define the speedup as $S_P(\mathcal{X}) = \lim_{n\to+\infty} \frac{T_P(1)}{T_P(\mathcal{X})}$. In this way, we treat all the other parameters as constants and we measure the speedup as a function of n. For example, for the complexity $\Theta(D + \sqrt{\frac{n}{\mathcal{X}}})$, we obtain $S_P(\mathcal{X}) = \sqrt{\mathcal{X}}$, suggesting that the speed of the algorithm depends on the square root of the bandwidth. Notice that, when D is not constant, this definition still makes sense: we have the same speedup, but we stop gaining speed when $\sqrt{\frac{n}{\mathcal{X}}}$ becomes as small as D.

Depending on S_P, we define three classes of problems and, for each of these classes, we provide examples of problems belonging to them.

- *bandwidth efficient*: the class of all problems P such that $S_P(\mathcal{X}) = \Theta(\mathcal{X})$, i.e., problems having a round complexity that fully scales with the bandwidth.

- *bandwidth sensitive*: this class contains all problems P such that $S_P(\mathcal{X}) = o(\mathcal{X})$ and $S_P(\mathcal{X}) = \omega(1)$, i.e., those problems that have a round complexity that scales with the bandwidth, but not linearly.
- *bandwidth insensitive*: the class of all problems P such that $S_P(\mathcal{X}) = \Theta(1)$, i.e., whose complexity does not depend on the size of the messages.

First, we investigate the round complexity of the All Pairs Shortest Paths (APSP) problem in unweighted graphs, in the CONGEST$_B$ model. We know that for this problem there is a lower bound of $\Omega(D + \frac{n}{B})$ rounds [43] and an algorithm performing in $O(n)$ rounds [55]. We show that the APSP problem is bandwidth efficient, i.e., that $S_{APSP}(\mathcal{X}) = \Theta(\mathcal{X})$, by modifying the existing algorithm to run in time $O(D + \frac{n \log n}{B}) = O(D + \frac{n}{\mathcal{X}})$.

We then investigate the round complexity of two well studied problems, namely Minimum Spanning Tree (MST) and Single Source Shortest Path (SSSP). For both these problems there is a lower bound of $\Omega(D + \sqrt{\frac{n}{B}})$ rounds [22]. On the other hand, these two problems have been studied only when $B = O(\log n)$. In this case, there exists an algorithm that solves the MST construction problem in $O(D + \sqrt{n} \log^* n)$ rounds [62], that is a round complexity that nearly matches the lower bound. For the SSSP problem, no sublinear algorithm that matches the lower bound and finds an exact solution is known. Conversely, we can find a $(1+\epsilon)$-approximation of the solution in $\widetilde{O}(\epsilon^{-O(1)}(\sqrt{n} + D))$ rounds, that is near-optimal. We show that the round complexity of both these problems scales with B, that is, if it is possible to send messages of size B at each round,

- there exists an algorithm that constructs a MST in $\widetilde{O}(D + \sqrt{\frac{n}{B}})$ rounds;
- there esists an algorithm that finds a $(1+\epsilon)$-approximation of the SSSP problem in $\widetilde{O}(\epsilon^{-O(1)}(\sqrt{\frac{n}{B}} + D))$ rounds.

Notice that these two round complexities match their respective lower bounds, up to polylogarithmic factors, for any value of B. Also, for both these problems, $S(\mathcal{X}) = \Theta(\sqrt{\mathcal{X}})$, i.e., these two problems are *bandwidth sensitive*.

We then show that there are problems for which, by increasing the bandwidth of the links, the round complexity does not change. In order to reduce the round complexity, one needs to increase the size of the messages exponentially. More

specifically, we show that there is a problem, $Distance_k$, solvable in $O(\log n)$ rounds using messages of size $B = O(\log n)$, but such that if we want to solve the problem in less than $\frac{\log n}{2}$ rounds, then we need to use messages of size at least $\Omega(\frac{n}{\log^3 n})$. In other words, $Distance_k$ is a *bandwidth insensitive* problem for bandwidths in the range from $\Omega(\log n)$ to $O(\frac{n}{\log^3 n})$ bits, since in this range $S(\mathcal{X}) = \Theta(1)$.

4.4 All Pairs Shortest Paths

In this section we show how to modify the APSP algorithm of [55, 81] and reduce the round complexity when the bandwidth increases, solving the problem in $O(D + \frac{n \log n}{B})$ rounds. Notice that a more elaborated proof allows to prove a bound of $O(D + \frac{n}{B})$ [56].

Recall that, in the APSP problem, each node needs to find its distance from all the other nodes. In [55, 81] it is shown how to deterministically find an exact solution of the APSP problem in $O(D + n)$ rounds in unweighted graphs. The procedure is the following:

1. construct a BFS tree;
2. perform a DFS visit on the tree;
3. at each step of the DFS visit, wait 1 round and then start a BFS from the current node (if it is visited for the first time).

All nodes will know their distances from all the other nodes by knowing their distance from the root of each BFS tree. In [55] it is shown that, by waiting one round before starting each visit, the breadth-first searches can be executed concurrently without congestion. In other words, at each round, at most one message passes on a fixed edge. The complexity comes by the fact that the DFS visit requires $O(n)$ rounds and that $O(D)$ is the time required to complete a BFS.

We show how to modify this procedure in order to solve the problem in $O(D + \frac{n \log n}{B})$ rounds.

Theorem 4.1. *There exists an algorithm for the* CONGEST$_B$ *model that solves the All Pairs Shortest Path problem in* $O(D + \frac{n \log n}{B})$ *rounds.*

Proof. Let s_i, $1 \leq i \leq n$, be the i-th node visited (for the first time) by a DFS performed on the BFS tree. Let t_i, $0 \leq t_i < 2n$, be the time at which node s_i is visited *for the first time* by the DFS. Consider the sequence $\mathcal{T} = (t_1, \ldots, t_n)$. We split $\mathcal{T}$ in $\lceil \frac{B}{\log n} \rceil$ subsequences $T_1, \ldots, T_{\lceil \frac{B}{\log n} \rceil}$ of length at most $\lceil \frac{2n \log n}{B} \rceil$ each, where T_j contains the values t_i such that $\lceil \frac{2n \log n}{B} \rceil (j-1) \leq t_i < \lceil \frac{2n \log n}{B} \rceil j$. Notice that, if each node s_i starts its BFS at time $2\,t_i$, there is no congestion between different breadth-first searches, for the same arguments showed in [55]. Now, in order to speed up the computation, we can visit all the sequences concurrently, by starting the BFS of node s_i with $t_i \in T_j$ at time $2(t_i - \lceil \frac{2n \log n}{B} \rceil (j-1))$. In other words, we are splitting the DFS visit in at most $\lceil \frac{B}{\log n} \rceil$ parts, each of them of length at most $\lceil \frac{2n \log n}{B} \rceil$. By doing this, there can be congestion for at most $\lceil \frac{B}{\log n} \rceil$ breadth-first searches at the same time (one for each sequence), and since the bandwidth is B, a constant number of rounds is enough to transmit on the same edge $\lceil \frac{B}{\log n} \rceil$ messages of size $O(\log n)$ belonging to different breadth-first searches. Since each sequence is of length at most $\lceil \frac{2n \log n}{B} \rceil$, the total time required to solve the problem is $O(D + \frac{n \log n}{B})$.

What remains to show is how each node s_i can compute t_i in $O(D)$ rounds. Notice that, by performing a convergecast, all nodes of the BFS tree can learn the size of the subtree rooted at each of its children. Now, starting from the root s_1 and setting $t_1 = 0$, each node s_i can assign an arbitrary order to its children, and send, to each child s_j, t_i and the sum σ of the number of nodes present in the subtrees rooted on the children before s_j. Then each child s_j can compute $t_j = 2\sigma + t_i + 1$ and repeat the same procedure. In total, $O(D)$ rounds are required. □

4.5 Minimum Spanning Tree

In this section we show how to modify the MST construction algorithm of [62] and reduce the round complexity when the bandwidth increases, by proving the following theorem.

Theorem 4.2. *There exists an algorithm for the* CONGEST$_B$ *model that constructs a Minimum Spanning Tree in* $\widetilde{O}(D + \sqrt{\frac{n}{B}})$ *rounds.*

Recall that, for the MST construction task, every node is given as input the weight $w(e)$ of each of its incident edges e. These weights are supposed to be of values polynomial in the size n of the network $G = (V, E, w)$, and thus each weight can be stored on $O(\log n)$ bits. The output of every node is a set of incident edges, such that the collection of all outputs forms an MST of the network. A Minimum Spanning Tree is a subset of edges $T \subseteq E$ such that (V, T) is connected and $\sum_{e \in T} w(e)$ is minimum. At the end of the computation each node must know which of its adjacent edges belong to the Minimum Spanning Tree.

The algorithm of [62] is divided in two phases: FAST-DOM-G and PIPELINE. The first part, FAST-DOM-G, computes a set of sub-MST by applying a modified version of the algorithm of Gallager, Humblet and Spira (GHS) [44]. The second part, PIPELINE, finds the remaining edges by performing a convergecast, where nodes send edges in non-decreasing order of weight by ignoring edges that close some cycle. FAST-DOM-G runs in $O(k \log^* n)$ rounds and creates $\frac{n}{k+1}$ fragments, where k is a parameter to be fixed later, while PIPELINE runs in $O(D + \frac{n}{k})$ rounds. By setting $k = \sqrt{n}$, they obtain a complexity of $O(D + \sqrt{n} \log^* n)$ rounds.

We show that PIPELINE can be performed faster if more bandwidth is allowed. More specifically, we show that it can be performed in $O(D + \frac{n \log n}{k \cdot B})$ rounds. By choosing $k = \sqrt{\frac{n \log n}{B}}$, we obtain Theorem 4.2.

Lemma 4.3. PIPELINE *can be completed in* $O(D + \frac{n \log n}{k \cdot B})$ *rounds, where* B *is the size of the messages.*

The rest of the section is used to prove Lemma 4.3, that is, we now show how to adapt the PIPELINE procedure in such a way that if each node sends $O(\frac{B}{\log n})$ edges per round, it still runs in a fully pipelined way. This implies that it is possible to send $O(\frac{n}{k+1})$ edges in $O(D + \frac{n}{k+1} / \frac{B}{\log n}) = O(D + \frac{n \log n}{(k+1)B}) = O(D + k)$ rounds.

The existing PIPELINE algorithm, modified to send $\frac{B}{\log n}$ edges per round, does the following:

1. construct a BFS-tree of G;
2. each node knows, from the first phase (FAST-DOM-G), the edges that connect it to other fragments;
3. each node keeps a set of edges (in non-decreasing order of weight) not already sent to its parent;
4. leaves start at round 0, intermediate nodes start when they have received at least a message from all their neighbors;
5. at each step each node sends the $\frac{B}{\log n}$ lightest edges that does not close a cycle with the edges already sent (or the ones that it is sending) and repeats this step until there are no valid edges (since $\frac{B}{\log n}$ could be non-integral, we can use two rounds to complete this task);
6. at the end the root chooses the $\frac{n}{k+1}$ lightest received edges and broadcasts them.

We adapt the proof in [62] to show that the PIPELINE procedure described above correctly completes within the required time. We first show the main part of the proof of [62] and then our adaptation.

In [62] it is shown inductively that, at each step, there is at least one edge that can be upcasted. Let a node be *active* if it has sent an edge in the previous step. Assume that there is a node v that is active from m steps and it has at least one active child u. Notice that its active children are active from at least $m+1$ steps. Let U be the set of edges already sent from node v to its parent in the previous steps. We know that this set forms a forest of trees, that we call $U_1, ..., U_l$. Let $x_i = |U_i|$. Notice that $\sum x_i = m$, since node v sent an edge in all the previous m rounds. We also know that $|V(U_i)| = x_i + 1$, since U_i is a tree. Let C be the set of edges that v already received from its child u. We know that $|C| \geq m+1$, since each child is active from at least one step before its parent.

Now, suppose for the sake of contradiction that all the edges in C are not candidates that can be sent. It means that each edge, either it has already been sent, or that it closes a cycle. In both cases, the endpoints of each edge are part of some edge of U (the edges already sent). Since U forms a forest this also implies that there are no edges that connect two different trees U_i, U_j of U. It means that C can be partitioned depending on the trees to which the endpoints of each

edge belong to. We can say that for each i, each edge in C_i has its endpoints in $V(C_i)$. And since C is a forest, we obtain that $|C_i| \leq |V(U_i) - 1|$. It follows that $|C_i| \leq |V(U_i) - 1| = x_i$, $C \leq \sum x_i = m \leq |C| - 1$ that is a contradiction. This implies that C contains at least one candidate.

The above proof can be adapted for our purposes in the following way. Since at each step a child sends $\frac{B}{\log n}$ edges, we know that $\sum x_i = |U| = \frac{mB}{\log n}$. We know also that children sent edges for at least $m+1$ steps, thus $|C| \geq \frac{(m+1)B}{\log n}$. Again, for the sake of contradiction, assume that the number of candidate edges is less than $\frac{B}{\log n}$. This means that at least $|C| - \frac{B}{\log n} + 1$ edges have both their endpoints in U. In this case there are at least $|C| - \frac{B}{\log n} + 1$ edges that can be partitioned in sets $C_1, \ldots, C_l$ such that the endpoints of edges in C_i are only in $V(U_i)$. Hence, we obtain that $\sum |C_i| \leq \sum(|V(U_i)| - 1) = \sum x_i$. Since $\sum |C_i|$ is at least $|C| - \frac{B}{\log n} + 1$, we have that $|C| - \frac{B}{\log n} + 1 \leq \sum x_i$, implying that $|C| \leq \sum x_i + \frac{B}{\log n} - 1$. Thus, we obtain $\frac{(m+1)B}{\log n} \leq |C| \leq \sum x_i + \frac{B}{\log n} - 1 = \frac{mB}{\log n} + \frac{B}{\log n} - 1 = \frac{(m+1)B}{\log n} - 1$, which is a contradiction. This proves that C contains at least $\frac{B}{\log n}$ candidates.

Figure 4.1 depicts how the complexity of MST scales with the available bandwidth.

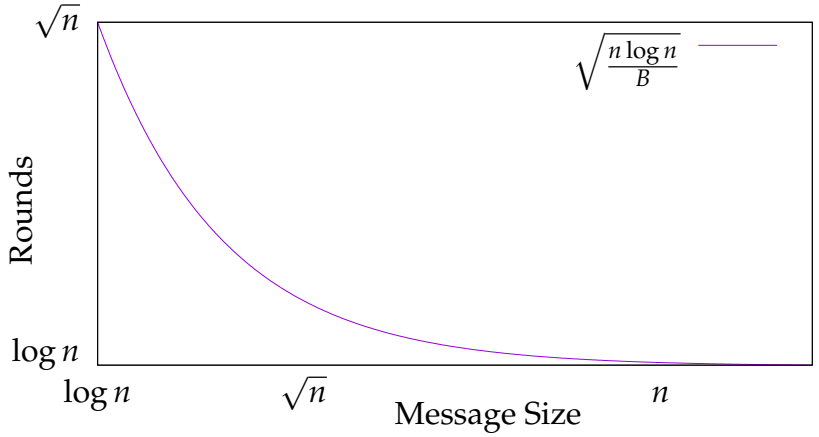

FIGURE 4.1: Round complexity of MST as a function of B

4.6 Single Source Shortest Path

In this section we show how to modify the SSSP algorithm of [13] and reduce the round complexity when the bandwidth increases. Recall that in the SSSP

problem, given a node v, all nodes of the graph have to find their distance from v. In [13] it is shown how to deterministically find a $(1+\epsilon)$ approximate solution of the SSSP problem using $\widetilde{O}(\epsilon^{-O(1)}(\sqrt{n}+D))$ rounds, where D is the (hop) diameter of the network. First, it exploits a Theorem of [54], that states the following:

Theorem 4.4. *([54]) Given any weighted undirected network $G = (V, E, w)$ and a source node $s \in V$, there is a $\widetilde{O}(\sqrt{n})$-round deterministic distributed algorithm in the broadcast* CONGEST *model that computes an overlay network $G' = (V', E', w')$ with edge weights $w' : E' \to \{1, \ldots, poly(n)\}$ and some additional information for every node with the following properties:*

- $|V'| = \widetilde{O}(\epsilon^{-1}\sqrt{n})$ *and* $s \in V'$.
- *For $\epsilon' := \epsilon/7$, each node $v \in V$ can infer a $(1+\epsilon)$-approximation of its distance to s from $(1+\epsilon')$-approximations of the distances between s and each $t \in V'$.*

By applying Theorem 4.4, the SSSP problem is reduced to a graph of roughly $\sqrt{n}$ nodes (*skeleton* nodes). Then, in [13] the authors provide an algorithm that solves the SSSP problem in the Broadcast Congested Clique and that runs in $\epsilon^{-9}polylog(n)$ rounds. They also show how to emulate this algorithm in the CONGEST model, with a slowdown proportional to D and to the number of nodes of the clique. Since the problem is reduced to an instance with $\sqrt{n}$ nodes, the algorithm can be emulated in $\epsilon^{-9}polylog(n) \cdot O(D+\sqrt{n})$ rounds. At this point the skeleton nodes can broadcast their distance from s to all the nodes of the original graph in $O(D+\sqrt{n})$ rounds, and all the nodes can compute their approximate distance from s (Theorem 4.4).

As stated in [54], Theorem 4.4 is a deterministic version of a more general theorem stated in [73], where $|V'|$ can be of size $\widetilde{O}(\alpha)$ and the running time of their algorithm is $\widetilde{O}(\alpha + \frac{n}{\alpha} + D)$ and succeeds with high probability. The round complexity is proved by providing an algorithm that solves the bounded-hop multi-source shortest path problem in $\widetilde{O}(|V'| + h + D)$ rounds, where the sources are nodes in V' and the number of hops h is $\frac{n \log n}{\alpha}$.

We now show that the aforementioned algorithm can complete in $\widetilde{O}(\frac{\alpha}{B} + \frac{n}{\alpha} + D)$ rounds, and that a round of communication of the broadcast congested clique with $|V'|$ nodes can be emulated in $\widetilde{O}(\frac{|V'|}{B} + D)$ rounds in the CONGEST$_B$ model, by proving the following lemmas.

Lemma 4.5. *A round of communication of the Broadcast Congested Clique can be emulated in the* CONGEST_B *model in* $\widetilde{O}(\frac{|V'|}{B} + D)$ *rounds.*

Proof. As in [13], we can solve the problem by using pipelining on a BFS tree. As in the MST case (Lemma 4.3), the speed of the pipelining linearly depends on $\frac{1}{B}$. □

Lemma 4.6. *There is a distributed algorithm that runs in* $\widetilde{O}(\frac{|V'|}{B} + h + D)$ *rounds in the* CONGEST_B *model that solves the bounded-hop multi-source shortest path problem.*

Proof. The idea of the algorithm in [73] is to execute many bounded-hop single-source shortest path algorithms in parallel, one for each source node, and to randomly delay the starting time of each execution in order to avoid congestion. In [73] it is shown that the execution of a single bounded-hop single-source shortest path algorithm requires $O(h + D)$ rounds using messages of logarithmic size, and by choosing a random delay from the interval of numbers from 0 to $|V'| \log n$, $|V'|$ executions can be performed in parallel without much congestion, obtaining a round complexity of $\widetilde{O}(|V'| + h + D)$. We show that, by increasing the bandwidth (thus allowing more congestion), it is possible to reduce the size of the interval from where the random delay is chosen.

Let $k = |V'|$. Let $\mathcal{M}_{i,u}$ be the set of messages sent by node u while executing the bounded-hop single-source shortest path algorithm for source s_i. In [73] (Lemma 3.7) it is shown that $\mathcal{M}_{i,u} \leq c \log n$ for some constant c. They then show that, if the delay is randomly chosen from the integers from 0 to $k \log n$, then the probability that there exists a time t, a node u and a set $\mathcal{M} \subseteq \bigcup_i \mathcal{M}_{i,u}$ such that $|\mathcal{M}| \geq \log n$ and all messages in $\mathcal{M}$ are broadcasted by u at time $t = O(k + h + D)$, is $O(\frac{1}{n^2})$. We extend this result showing in the remaining part of the section that, if the delay is randomly chosen from the integers from 0 to $\frac{k \log^2 n}{B}$, then the probability that $\mathcal{M} \geq B$ is also $O(\frac{1}{n^2})$, that implies the lemma.

Fix any node u, time t and set $\mathcal{M}$ as above. Since u at each round sends at most one message for each s_i, we can assume that $|\mathcal{M} \cap \mathcal{M}_{i,u}| \leq 1$. This implies that $|\mathcal{M}| \leq k$. As in the original proof, we can bound the number of possible sets $\mathcal{M}$ of size m by $\binom{k}{m}(c \log n)^m$. This holds since each set $\mathcal{M}$ can be constructed by picking m different sets $\mathcal{M}_{i,u}$ and picking one message out of $c \log n$ messages from each $\mathcal{M}_{i,u}$. Then, the probability that all the messages of a set are sent at the same round is at most $(\frac{B}{k \log^2 n})^{|M|}$. Thus, for fixed u and t, the probability

that there exists $\mathcal{M}$ such that $|\mathcal{M}| \geq B$ and all messages in $\mathcal{M}$ are sent by u at time t, is at most

$$\sum_{m=B}^{k} \binom{k}{m} (c \log n)^m \left(\frac{B}{k \log^2 n}\right)^m \leq$$

$$\sum_{m=B}^{k} \left(\frac{ke}{m}\right)^m (c \log n)^m \left(\frac{B}{k \log^2 n}\right)^m =$$

$$\sum_{m=B}^{k} \left(\frac{ecB}{m \log n}\right)^m \leq \sum_{m=B}^{k} \left(\frac{ecB}{B \log n}\right)^B =$$

$$\sum_{m=B}^{k} \left(\frac{ec}{\log n}\right)^B \leq k \left(\frac{ec}{\log n}\right)^B$$

Since $B \geq \log n$, for large n the above formula is at most $\frac{1}{n^4}$. Then the lemma follows by summing this probability over all nodes u and time steps t. □

Now we have all the ingredients to prove the following theorem.

Theorem 4.7. *There is a distributed algorithm that runs in $\widetilde{O}(\epsilon^{-O(1)}(\sqrt{\frac{n}{B}} + D))$ rounds in the* CONGEST$_B$ *model that solves the $(1+\epsilon)$ approximate SSSP problem.*

Proof. Let $|V'|$ be $\widetilde{O}(\epsilon^{-1}\sqrt{nB})$. By Theorem 4.4 and Lemma 4.6, we can reduce the problem to a SSSP instance on $|V'|$ nodes in $\widetilde{O}(\epsilon^{-1}\sqrt{\frac{n}{B}} + D)$ rounds. We can then simulate, using Lemma 4.5, the Broadcast Congested Clique algorithm of [13] in $\widetilde{O}(\epsilon^{-O(1)}(\sqrt{\frac{n}{B}} + D))$ rounds to solve the original problem. □

4.7 *Distance$_k$*

In this section we show a problem, called *Distance$_k$*, whose round complexity does not depend on the size B of the messages, for a large interval of values of B. Consider a graph $G = (V, E)$ that is the underlying communication graph, and consider a (possibly different) directed graph $G' = (V, E')$. To each node $v \in V$ is provided as input the set of its neighbors that it has in G' (its outgoing edges). Notice that the neighbors that v has in G and in G' could be different. The problem *Distance$_k$* consists in finding a node $w \in V$ that is at distance k from a fixed node u in G'. A node w is at distance k from u if there is a directed path starting from u that ends at w and that path is the shortest one. Notice that

both u and w are part of the input, that is, only a fixed node needs to find the distance. An example of an input instance of $Distance_k$ is shown in Figure 4.2, where, for example, a valid result for $Distance_2(1)$ is 7.

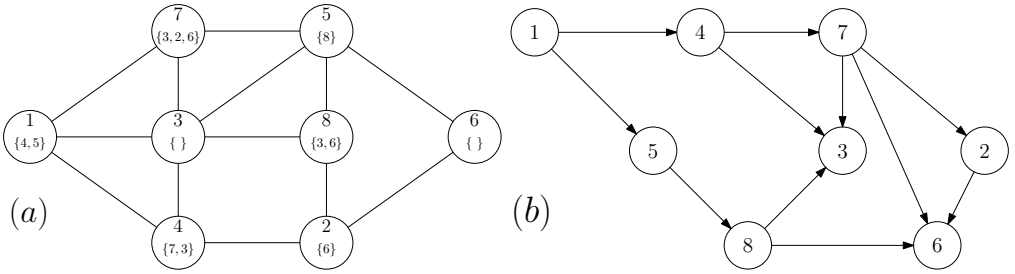

FIGURE 4.2: (a) represents G, the communication graph where the algorithm is executed and the input given to the nodes, while (b) is G', the directed graph given in input to the nodes.

Theorem 4.8. *There are instances of $Distance_k$ that can be solved in $O(k)$ rounds using messages of size $B = O(\log n)$, that, if we want to solve in $\lfloor \frac{k-1}{2} \rfloor$ rounds, we need to use messages of size $B = \Omega(\frac{n}{k^3})$.*

In the following we will prove Theorem 4.8. Consider instances of $Distance_k$ where the out-degree of G' is at most 1. In this case, in the CONGEST model, using messages of size $O(\log n)$ bits, this problem can be solved in $O(k \cdot D(G))$ rounds, where $D(G)$ is the diameter of the communication graph, using Algorithm 3.

Algorithm 3 $Distance_k(u)$

```
1:  if my Id is u then
2:      my distance is 0
3:  else
4:      my distance is +∞
5:  end if
6:  for i ← 0, k − 1 do
7:      if my distance is i then
8:          broadcast in G the Id of my neighbor in G'
9:      end if
10:     if I receive a broadcast containing my Id ∧ my distance is +∞ then
11:         my distance is i + 1
12:     end if
13: end for
14: if my distance is k then
15:     broadcast my Id
16: end if
```

The idea is the following: once a node knows that it is at distance i, it can perform a BFS on G in order to broadcast its outgoing edge, that it has in G', to all the other nodes. Since each broadcast requires $O(D)$ rounds, in total $O(k \cdot D)$ rounds are required. Notice that, if the diameter of G is constant, the complexity becomes $O(k)$.

We now show that, in order to solve the problem in $\lfloor \frac{k-1}{2} \rfloor$ rounds, we need messages of size at least $\Omega(\frac{n}{k^3})$. Notice that, by choosing $k = \log n$, we obtain the following: there are instances of $Distance_k$ that can be solved in $O(\log n)$ rounds using messages of size $O(\log n)$, while in order to solve the problem in less than a constant fraction of $\log n$ rounds, messages of size $\Omega(\frac{n}{\log^3 n})$ are needed, i.e., there is an exponential gap in the size of the messages.

In [75] the following problem, $Pointer_k$, is defined. There are two players, Alice and Bob, and to each of them is provided a list of n pointers, each pointing to the list of the other player. Their task is to follow these pointers, starting from some fixed pointer of Alice, and then find the k-th pointer. It is known that the communication complexity of this problem is $O(k \log n)$ bits: it is enough to send the i-th pointer at the i-th round, for a total of k rounds, where at each round $\log n$ bits are sent. They show that, in order to solve this problem in $k-1$ rounds, even if Las Vegas algorithms are allowed, the communication complexity of this problem becomes $\Omega(\frac{n}{k^2})$ bits. Notice that for $k = \log n$ there is an exponential gap between the two complexities.

We can reduce an instance of the problem $Pointer_k$ on p pointers to an instance of $Distance_k$ with $p+2$ nodes in the following way. Construct a graph $G = (V, E)$ where the nodes V are partitioned in two groups L and R of equal size, plus two special nodes l and r (see Figure 4.3). The nodes of each group are connected to a single special node and there is an edge connecting the two special nodes l and r. To each node of L is assigned a different ID from the set $\{1, \ldots, |L|\}$ and to each node of R is assigned a different ID from the set $\{|L|+1, \ldots, |L|+|R|\}$. We assign to node l the ID $|V|-1$ and to node r the ID $|V|$. The edges of $G' = (V, E')$ are defined in the following way. If the i-th pointer of Alice points to the j-th pointer of Bob, the node with ID i has an outgoing edge to the node with ID $|L|+j$. On the other hand, if the i-th pointer of Bob points to the j-th pointer of Alice, the node with ID $|L|+i$ has an outgoing edge to the node with ID j. Also, nodes $|V|-1$ and $|V|$ have an edge between themselves. An example of reduction is shown in Figure 4.3. Notice that Alice

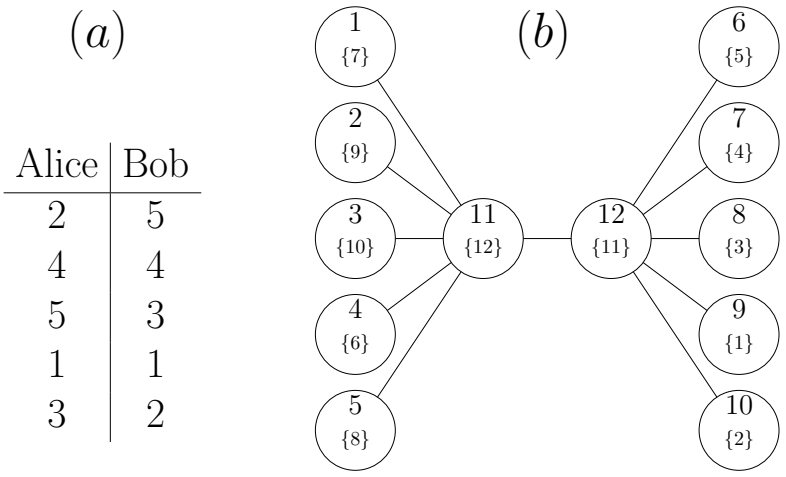

FIGURE 4.3: (a) shows the $Pointer_k$ instance, while (b) shows its $Distance_k$ version.

(resp. Bob) can construct the input of the nodes of L (resp. R) without communicating with Bob (resp. Alice). Assume that there is an algorithm for the CONGEST$_B$ model that solves the problem in $\lfloor \frac{k-1}{2} \rfloor$ rounds. Alice and Bob can simulate this algorithm in at most $k-1$ rounds by exchanging the B bits that are transmitted between the two special nodes at each round. Thus, a lower bound for $Pointer_k$ holds for $Distance_k$ as well.

In the CONGEST$_B$ model, we are allowed to send B bits on each edge at each round. If we want to solve $Distance_k$ in at most $\lfloor \frac{k-1}{2} \rfloor$ rounds, we need to transfer $\Omega(\frac{n}{k^2})$ bits, thus $\Omega(\frac{n}{Bk^2})$ rounds are required. In other words, we need to satisfy $\frac{n}{Bk^2} \leq \lfloor \frac{k-1}{2} \rfloor$, that gives $B = \Omega(\frac{n}{k^3})$.

Figure 4.4 depicts how the complexity of $Distance_k$ scales with the available bandwidth.

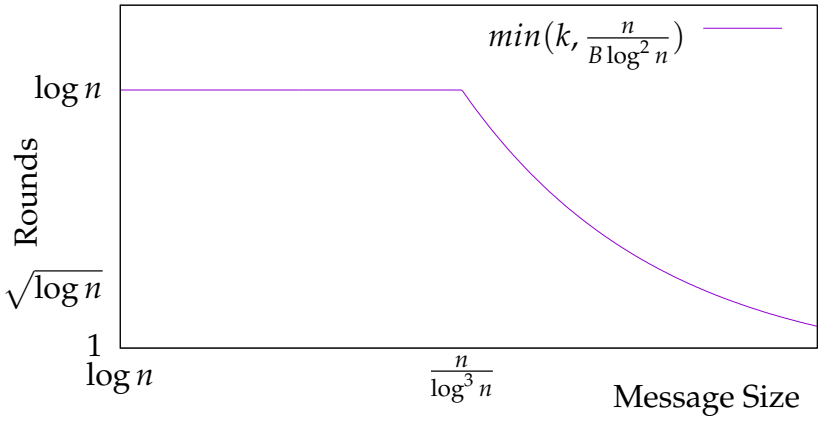

FIGURE 4.4: Round complexity of of $Distance_k$ as a function of B

4.8 Conclusions

We considered the problem of understanding how the bandwidth affects the round complexity in the CONGEST model. Our results show that in some cases the round complexity perfectly scales with the bandwidth, matching existing lower bounds. On the other hand we showed that there are problems in which we need to increase exponentially the size of the messages in order to speed up the round complexity. These results suggest that it is important to analyze algorithms in the CONGEST model by considering a wider spectrum of bandwidths. It would be interesting to study this tradeoff between bandwidth and round complexity for other problems present in the literature.

Chapter 5

Clique Emulation

In this chapter we investigate tradeoffs between the number of edges of a graph and its ability to allow an efficient all-to-all communication. We first provide graphs that optimally satisfy this tradeoff, and then we investigate the ability of Erdős-Rényi random graphs $\mathcal{G}_{n,p}$ to emulate the clique. This chapter is based on results published in [12].

5.1 Introduction

Avin, Borokhovich, Lotker, and Peleg [8] proposed a novel network architecture for parallel and distributed computing, called Core-Periphery networks. This architecture is described implicitly by providing three *axioms*. One of these axioms requires that the *core* C satisfies the following.

Clique emulation: the core can emulate the clique in a constant number of rounds in the CONGEST model. That is, there is a communication protocol running in a constant number of rounds in the CONGEST model such that, assuming that each node $v \in C$ has a message $M_{v,w}$ on $O(\log n)$ bits for every $w \in C$, then, after $O(1)$ rounds, every $w \in C$ has received all messages $M_{v,w}$, for all $v \in C$. In other words, the *all-to-all* communication pattern can be implemented in a constant number of rounds.

Given this axiomatic description, it is not specified how to actually build a graph that can satisfy the requirements. In other words, if we want to build

a Core-Periphery network, we need to find a graph and a routing schema associated to this graph, that is able to satisfy the axiom efficiently. In Figure 5.1 is depicted a graph that can emulate the clique communication in just three rounds.

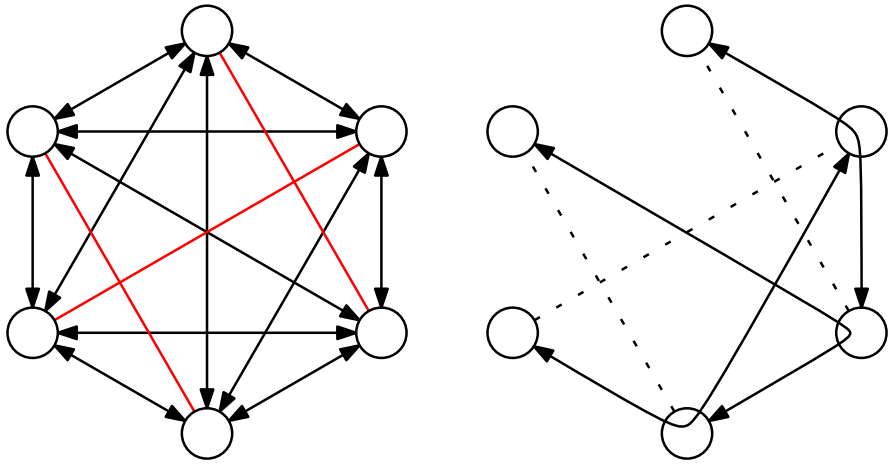

FIGURE 5.1: (left) Red edges are missing. (right) Emulation of missing edges

5.2 Our Goal

In this work, we are aiming at establishing tradeoffs between the number of edges, and the capability of emulating the clique. More precisely, we consider the all-to-all communication pattern:

- **Input:** every node v has a message $M_{v,w}$, for every node $w \neq v$;
- **Output:** every node w has received the message $M_{v,w}$, for every node $v \neq w$.

In the CONGEST model, assuming all messages are on $O(\log n)$ bits, all-to-all communication can be performed in just a single round if the graph is a clique. Our objective is to study the tradeoff between the number of edges, and the number of rounds for performing the all-to-all communication in the CONGEST model.

5.3 Results

We show that, in the CONGEST model, implementing the all-to-all communication in k rounds can be done in n-node networks with roughly n^2/k edges, and this bound is essentially tight because every node must have degree at least $(n-1)/k$ to receive $n-1$ messages in at most k rounds. Hence, sparsifying the clique beyond just saving a fraction of the edges requires to relax the constraint on the time required to simulate that clique.

We also investigate the ability of random graphs to emulate the clique. Let us define $\alpha = \sqrt{3e/(e-2)}$, where e is the basis of the natural logarithm. We show that, for $p \geq \alpha\sqrt{\ln n/n}$, a random graph in $\mathcal{G}_{n,p}$ can, w.h.p., perform an all-to-all communication in $O(\min\{\frac{1}{p^2}, np\})$ rounds.

5.4 Related Work

Emulating the clique in Core-Periphery networks gives the ability to emulate many algorithms designed for the Congested Clique, a different model that has been widely studied in the literature. For example, we can emulate the routing and sorting algorithms of Lenzen [65], which showed a deterministic algorithm that, if each node is the sender and receiver of at most n messages, allows to route all the messages in $O(1)$ rounds in a clique of size n using messages of size $O(\log n)$. Ghaffari et al. [46] showed how to solve a similar task in general graphs. More precisely, they show that, given several packet routing requests, such that each node v is the source and destination for at most $d(v)$ packets, where d is the degree of v, we can route all these packets from their sources to their destinations in $\tau_{mix} \cdot 2^{O(\sqrt{\log n \log\log n})}$ rounds, where τ_{mix} is the *mixing time* of the network.

A problem related to the clique emulation is broadcasting. Feige et al. [33] studied the broadcast problem in random graphs, where a single node has a message that has to be received by all the nodes of the graph. They show that rumor spreading (which propagates the message to a randomly chosen neighbor at each step) is an efficient way to solve the broadcast problem for random graphs. Censor-Hillel et al. [19] studied the broadcast problem in the context where every node is the source of a message and it is limited to send the *same*

message to each neighbor at each round. They give an efficient algorithm that solves the problem, also in case of failures.

Finally, it is worth mentioning that a problem related to ours, that is finding disjoint paths between pairs of nodes, has been largely investigated in expander graphs, which are sparse graphs that guarantee strong connectivity properties [15, 41, 64, 42].

5.5 Deterministic Construction

In this section we provide a deterministic construction yielding a perfect tradeoff between number of edges and number of rounds in clique emulation.

Theorem 5.1. *Let $n \geq 1$, and $k \geq 3$. There is an n-node graph with at most $\lceil \frac{k-2}{(k-1)^2} \ n^2 \rceil$ edges that can emulate the n-node clique in k rounds. Also, there is an n-node graph with at most $\frac{1}{3}n^2$ edges that can emulate the n-node clique in 2 rounds.*

Proof. First, we show that there is an n-node graph with at most $\frac{1}{3}n^2$ edges that can emulate the n-node clique in 2 rounds. For this purpose, recall that the so-called Johnson graph $J(n,r)$ has vertex set composed of all the r-element subsets of the set $\{1,\ldots,n\}$, and two vertices are adjacent if and only if they meet in a $(r-1)$-element set.

Claim 1. There exists an independent set I of size at least $\lceil \frac{1}{n}\binom{n}{3} \rceil$ in the Johnson graph $J(n,3)$.

To establish the claim, for every k, $0 \leq k < n$, let us consider the set

$$I_k = \{\{x,y,z\} \in V(J(n,3)) \mid x+y+z \equiv k \pmod{n}\}$$

Every set I_k is an independent set. Indeed, if two triples $\{x,y,z\}$ and $\{x,y,z'\}$ are both in I_k, then $x+y+z \equiv k \pmod{n}$ and $x+y+z' \equiv k \pmod{n}$. Therefore, $z \equiv z' \pmod{n}$, which implies $z = z'$, because $z, z' \in \{1,\ldots,n\}$. Observe that $\{I_0,\ldots,I_{n-1}\}$ is a partition of $V(J(n,3))$. Therefore, one of them has size at least $\lceil \frac{1}{n}\binom{n}{3} \rceil$, which establishes the claim. In Figure 5.2 it is shown K_5, $J(5,3)$, and $I = \{\{2,3,5\},\{1,4,5\}\}$.

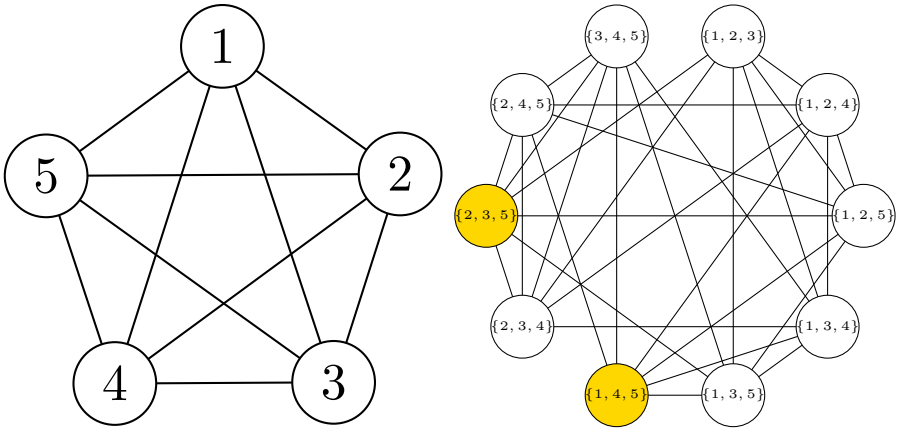

FIGURE 5.2: (left) The clique of size five. (right) The Johnson graph $J(5,3)$, and an independent set of size two.

Let I as in Claim 1. Note that for any $\{a,b,c\} \in I$, none of the edges $\{a,b\}$, $\{a,c\}$, $\{b,c\}$ are appearing in any other triples of I. Thus, the edge $\{a,b\}$ of the complete graph can be emulated by the path $\{a,c\},\{b,c\}$ without congestion resulting from the emulation of another edge $\{a',b'\}$. Moreover, the edge $\{a,b\}$ itself does not belong to any path used to emulate other edges. It follows that one can remove $|I|$ edges from K_n, one from each triple in the independent set I, and all removed edges can be emulated by edge-disjoint paths of length 2. Figure 5.3 shows how to emulate the six communications $x \to y$ for every ordered pair (x,y), $x \in \{a,b,c\}$, $y \in \{a,b,c\}$, $x \neq y$, in just 2 rounds. It follows that there is an n-node graph with at most $\frac{n^2}{3}$ edges that can emulate the n-node clique in 2 rounds.

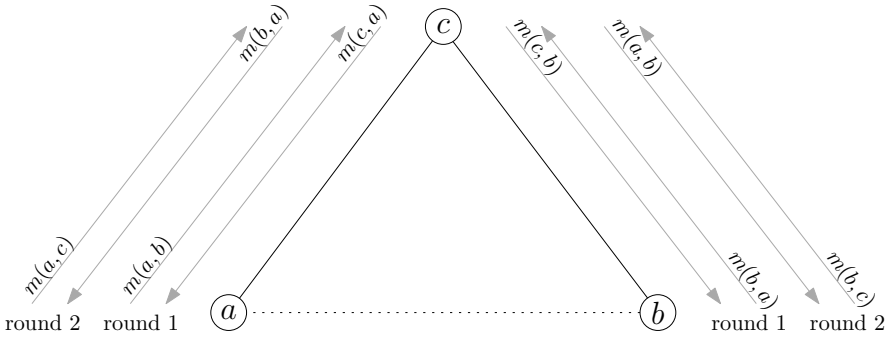

FIGURE 5.3: *Emulation of removed edge* $\{a,b\}$ *(*$m(x,y)$ *denotes the message from* x *to* y*).*

We now move on with the general case, that is, we show that there is an n-node graph with at most $\lceil \frac{n^2(k-2)}{(k-1)^2} \rceil$ edges that can emulate the n-node clique in k rounds.

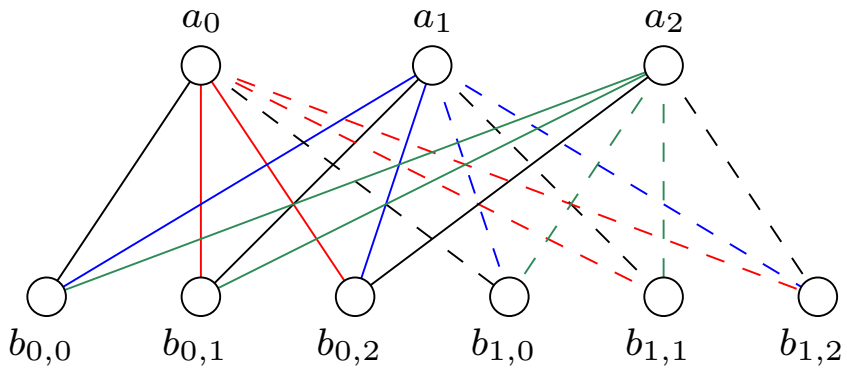

FIGURE 5.4: Emulating K_9 with $K_{3,6}$. The plain red path $(b_{0,1}, a_0, b_{0,2})$ is used at the 1st round for exchanging messages between $b_{0,1}$ and $b_{0,2}$, and, at the 2nd round, it is used for sending messages from $b_{0,1}$ to $b_{1,2}$, and from $b_{0,2}$ to $b_{1,1}$.

Claim 2. All-to-all communication between the nodes of the same part of the complete bipartite graph $K_{r,r}$ can be performed in 2 rounds.

Indeed, let A and B be the two parts of $K_{r,r}$, where the nodes in A and B are labeled $a_0, \ldots, a_{r-1}$ and $b_0, \ldots, b_{r-1}$, respectively. Let us consider $a_i \in A$, and its message for node $a_j \in A$. This message is routed via node $b_k \in B$ where $i + j + k \equiv 0 \pmod{r}$. This guarantees that each edge is used at most once in each direction, at each round. Indeed, sender a_i chooses different intermediate nodes to route messages to the different receivers a_j, $j \neq i$. Similarly, for the same receiver j, different senders a_i, $i \neq j$, choose different intermediate nodes. This proves Claim 2.

By performing the above routing scheme in parallel, we directly get the following:

Claim 3. Let A and B be the two parts of the complete bipartite graph $K_{r,kr}$, and let us partition the nodes of B into k groups $B_0, \ldots, B_{k-1}$ of r nodes each. The k all-to-all communication patterns between the nodes of B_i can be performed in parallel for all $i \in \{0, \ldots, k-1\}$, in 2 rounds, also in parallel to all-to-all communication between the nodes of A.

We have now all the ingredients to establish the general case of Theorem 5.1. Let $k \geq 1$, and let $K_{r,kr}$ be the n-node complete bipartite graph with $r = \frac{n}{k+1}$ nodes in the first part A, and $kr = \frac{nk}{k+1}$ nodes in the other part B. Note that $K_{r,kr}$ has $kr^2 = \frac{n^2 k}{(k+1)^2}$ edges. We show how to perform all-to-all in $K_{r,kr}$ in $k+2$ rounds. We divide the kr nodes of B into k groups $B_0, \ldots, B_{k-1}$ of r nodes each. For $i \in \{0, \ldots, k-1\}$, we set $B_i = \{b_{i,j}, 0 \leq j \leq r-1\}$ — cf. Figure 5.4. We describe

a routing scheme that allows the kr nodes of B to perform all-to-all, by relaying their messages using the r nodes of A. Routing is achieved by repeating k times the all to all routing protocol in Claim 3, where, at each phase $s = 1, \ldots, k$, nodes of B_i perform the communications with the nodes in $B_{j+s \bmod k}$. Importantly, the above routing scheme does not require $2k$ rounds but only $k+1$ rounds, because the kr nodes in B do not have to wait for receiving relayed messages in order to start sending new messages, and the phases can be pipelined. One more round is used to route the direct communication between every node in A and every node in B. Interestingly, during the $k+1$ rounds needed to perform all-to-all communications between the nodes in B, the edges are always used in both directions, except for the first and last round. We can use these two rounds to let the nodes in A perform their own all-to-all among them using the same routing pattern as in Claim 2. In total, in the $\frac{n^2k}{(k+1)^2}$-edge graph $K_{r,kr}$, all-to-all is performed in $k+2$ rounds. □

We complete the section by showing that the bounds in Theorem 5.1 provide an essentially optimal tradeoff between the number of rounds k performed in the emulation, and the number of edges m of the emulator. A trivial lower bound $\frac{1}{2}\frac{n(n-1)}{k}$ can be obtained by noticing that every node must have degree at least $\frac{n-1}{k}$ for receiving $n-1$ messages in k rounds. The following theorem improves this trivial bound by a factor 2, and matches with the bound in Theorem 5.1.

Property 1. Let $n \geq 1$, $k \in \{1, \ldots, n-1\}$, and let G be an n-node graph that can emulate the n-node clique in k rounds. Then G has at least $\frac{n(n-1)}{k+1}$ edges.

Proof. Let m be the number of edges of G. There are $\binom{n}{2}$ pairs of nodes in K_n, communicating $n(n-1)$ messages in total. In G, only m pairs of nodes are directly connected. All the other $\binom{n}{2} - m$ pairs of nodes are not directly connected, and they are at least at distance 2 in G. Thus, the number of mesages generated to route the messages corresponding to these pairs of nodes is at least $4(\binom{n}{2} - m)$. The total number of messages to be transferred is thus at least $2m + 4(\binom{n}{2} - m)$. Since one communication round in G can route at most $2m$ messages, it follows that any routing protocol requires at least $\frac{2m+4\binom{n}{2}-4m}{2m} = \frac{n(n-1)}{m} - 1$ rounds of communication. Thus, $k \geq \frac{n(n-1)}{m} - 1$, which implies $m \geq \frac{n(n-1)}{k+1}$. □

Figure 5.5 compares the provided lower and the upper bounds.

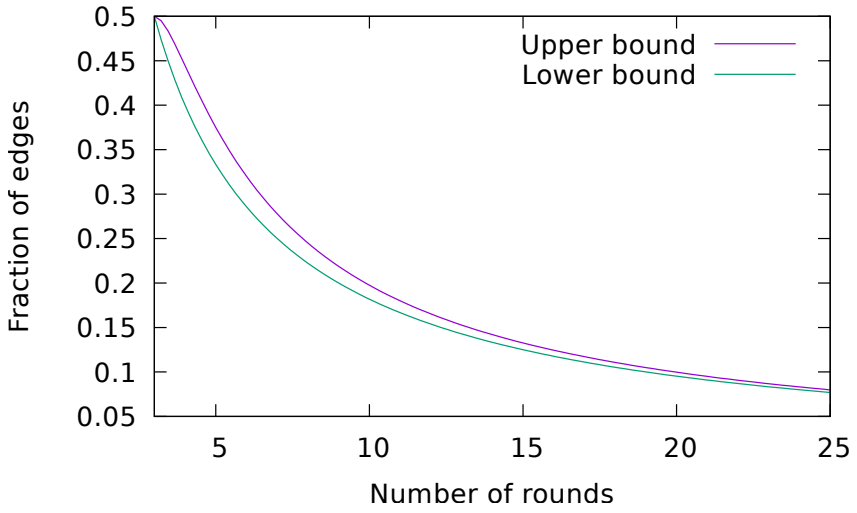

FIGURE 5.5: Number of edges necessary to emulate the clique communication.

5.6 Randomized Construction

In this section, we consider clique emulation by Erdős-Rényi random graphs $\mathcal{G}_{n,p}$. Our main result is the following.

Theorem 5.2. *Let $c \geq 0$, $n \geq 1$, $\alpha = \sqrt{(3+c)e/(e-2)}$ where e is the base of the natural logarithm, and $p \geq \alpha\sqrt{\ln n/n}$. For $G \in \mathcal{G}_{n,p}$,*

$$\Pr[G \text{ can emulate } K_n \text{ in } O(\min\{\tfrac{1}{p^2}, np\}) \text{ rounds}] \geq 1 - O(\tfrac{1}{n^{1+c}})$$

where the big-O notations hide the dependency in c.

Proof. Let $G \in \mathcal{G}_{n,p}$. The proof works as follows. For each missing edge in G between two nodes u and v, we route the messages between these nodes via an intermediate node w, i.e., along a path (u, w, v) of length 2. The intermediate node is picked at random among all nodes w such that $\{u, w\} \in E(G)$, and $\{w, v\} \in E(G)$. To analyze the load of the edges, we have to overcome two problems. First, the load of an edge is not necessarily independent from the load of another edge. Second, we are interested in the maximum, taken over all edges, of the load of the edges. As a consequence, an analysis based only on the expectation of the load of each edge may not yield accurate results. Instead, we base our analysis on a double application of a balls-into-bins protocol.

We aim at constructing a path for routing the messages between every pair of nodes that are not directly connected in G. As said before, the alternative paths

used to replace missing edges are of length 2, and the probability expressed in the statement of the theorem reflects the probability that such paths exist, without too much congestion. More specifically, let us consider a missing edge $\{i, j\}$ in G. Let $S_{i,j}$ be the set of common neighbors to i and j in G. The message from i to j is aimed at being routed via some intermediate node $k \in S_{i,j}$. The first question to address is thus: how large is $S_{i,j}$? To answer this question, let $\mathcal{E}_{i,j}$ be the event "there are at least $\frac{np^2}{e}$ different paths of length 2 between i and j", and let $\mathcal{E} = \bigcap_{\{i,j\} \notin E(G)} \mathcal{E}_{i,j}$.

Claim 4. Let $\alpha_c = \sqrt{(c+3)e/(e-2)}$, and $p \geq \alpha_c \sqrt{\ln n / n}$. Then

$$\Pr[\mathcal{E}] \geq 1 - \frac{1}{n^{c+1}}.$$

To establish the claim, let $X_{i,j,k}$ be the Bernoulli random variable, for $\{i, j\} \notin E(G)$, such that $X_{i,j,k} = 1$ iff $k \in S_{i,j}$, i.e., $\{i, k\} \in E(G)$ and $\{k, j\} \in E(G)$. Then let $X_{i,j} = \sum_{k=1}^{n} X_{i,j,k}$. We have $\Pr[X_{i,j,k} = 1] = p^2$, and, for a fixed pair i, j, the variables $X_{i,j,k}$, $k = 1, \ldots, n$, are mutually independent. Thus, using Chernoff bounds, we get:

$$\Pr[X_{i,j} \leq \frac{np^2}{e}] \leq e^{(\frac{2}{e}-1)np^2}.$$

By union bound, it follows that

$$\Pr[\bigcup_{\{i,j\} \notin E(G)} \overline{\mathcal{E}_{i,j}}] \leq n^2 e^{(\frac{2}{e}-1)np^2} \leq \frac{1}{n^{c+1}}$$

as desired, where the last inequality holds because $p \geq \alpha_c \sqrt{\ln n / n}$.

In addition to Claim 4, we will also use the following known result:

Lemma 5.3 ([71]). *Let $X_1, \ldots, X_n$ be a sequence of random variables in an arbitrary domain, and let $Y_1, \ldots, Y_n$ be a sequence of binary random variables, with the property that Y_i is a function of the variables $X_1, \ldots, X_{i-1}$. If, for every $i = 1, \ldots, n$, we have $\Pr[Y_i = 1 | X_1, \ldots, X_{i-1}] \leq q$ then $\Pr[\sum_{i=1}^{n} Y_i \geq k] \leq \Pr[B(n, q) \geq k]$ where $B(n, q)$ denotes the binomial distribution of parameters n and q.*

Our path construction algorithm for every missing edge $\{i, j\} \notin E(G)$ is sequential, and proceeds as follows. For every $\{i, j\} \notin E(G)$, the path from i to j is not necessarily the same as the path from j to i. We process all ordered pairs of nodes (i, j) in n phases, where Phase i, $i = 1, \ldots, n$, constructs all paths (i, j) for

$\{i,j\} \notin E(G)$, in increasing order of j. Assume already fixed a set of paths, corresponding to previously considered sender-receiver pairs, and consider now the pair (i,j) (of course corresponding to the missing edge $\{i,j\} \notin E(G)$). The previously constructed paths induce some load on each edge of G, corresponding to the number of paths using that edge. The choice of the path for (i,j) depends on this load, and is inspired from the power of two choices in balls-and-bins protocols. Precisely, for suitable parameters d and r, node i repeats r times the following: pick d incident edges $\{i,k\}$ uniformly at random, and select the least loaded one. Once this is done, node j picks the least loaded edge among the r edges selected by i.

Let $I_{i,j}$ be the node selected to route the message from sender i to receiver j. Messages from i to j will be routed along the path $P_{i,j} = (i, I_{i,j}, j)$. For $h \geq 0$, let $b_{i,h}(j)$ be the number of edges $\{i,k\}$ of load at least h after deciding the intermediate nodes $I_{i,1}, \ldots, I_{i,j}$ of the first j receivers for sender i. We define the following quantities:

$$x = \left\lceil \frac{e^{5+c}}{p^2} \right\rceil \text{ and } \beta = \frac{np^2}{e^{5+c}}.$$

Since $b_{i,x}(n) \leq n/x$, it follows from the above that $b_{i,x}(n) \leq \beta$. Now, let

$$\ell(j) = |\{j' \leq j : I_{i,j'} = I_{i,j}\}|.$$

We define the random variables $Z_{i,j}$ where

$$Z_{i,j} = \begin{cases} 1 & \text{if } \ell(j) \geq x+1 \\ 0 & \text{otherwise.} \end{cases}$$

Hence $Z_{i,j} = 1$ is the bad event that the edge between node i and the intermediate node $I_{i,j}$ used to route from i to j is heavily loaded by i. Conditioned on the fact that $\mathcal{E}$ holds (cf. Claim 4), we get that

$$\Pr[Z_{i,j} = 1] \leq r \left(\frac{\beta}{np^2/e} \right)^d.$$

We let q be the right hand side of the above equation. Let us now consider $Z_i = \sum_{j=1}^{n} Z_{i,j}$. Observe that $Z_{i,j}$ is a function of $I_{i,1}, \ldots, I_{i,j-1}$. Therefore, by Lemma 5.3 we get that

$$\Pr[Z_i \geq k] \leq \Pr[B(n,q) \geq k].$$

So, in particular, $\Pr[Z_i \geq 1] \leq \Pr[B(n,q) \geq 1]$. We now set $d = \ln n$, and $r \leq n$ (a suitable r will be specified thereafter). Thanks to this choice of d and r, we have $q \leq \frac{1}{n^{3+c}}$, and therefore

$$\Pr[Z_i \geq 1] \leq \Pr[B(n, \frac{1}{n^{3+c}}) \geq 1] \leq \mathbf{E}[B(n, \frac{1}{n^{3+c}})] \leq \frac{1}{n^{2+c}}.$$

Let $Z = \sum_{i=1}^{n} Z_i$. By union bound, we get $\Pr[Z \geq 1] \leq \frac{1}{n^{1+c}}$.

Using a similar analysis, from the perspective of the receiver, and defining the corresponding random variables $Z'_{i,j}$ capturing the load of the edges incident to a receiver j, and $Z'_j = \sum_{i=1}^{n} Z'_{i,j}$, we get

$$\Pr[Z'_j \geq 1] \leq \Pr[B(n,q') \geq 1]$$

where

$$q' = \left(1 - \left(1 - \frac{e\beta}{np^2}\right)^d\right)^r.$$

We get $q' \leq \frac{1}{n^{3+c}}$ by setting $d = \ln n$ and $r = (c+3)\ n^{\epsilon}\ \ln n$ for $\epsilon = -\ln(1 - \frac{1}{e^{4+c}})$. By this setting of d and r, we get that

$$\Pr[Z'_j \geq 1] \leq \Pr[B(n, \frac{1}{n^{3+c}}) \geq 1] \leq \mathbf{E}[B(n, \frac{1}{n^{3+c}})] \leq \frac{1}{n^{2+c}}.$$

Let $Z' = \sum_{j=1}^{n} Z'_j$. By union bound, we get $\Pr[Z' \geq 1] \leq \frac{1}{n^{1+c}}$.

Therefore, altogether, we get that

$$\Pr[Z = 0 \text{ and } Z' = 0 \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] \geq (1 - \frac{1}{n^{1+c}})^3 \geq 1 - \frac{3}{n^{1+c}}.$$

In other words, w.h.p., the load of all edges is no more than $x = O(1/p^2)$. On the other hand, with a similar argument as for proving that the degree is large, we have that, w.h.p., the degree of all nodes is at most enp, and therefore the load of an edge does not exceed enp. □

5.7 Conclusions

In this work, we first provided a deterministic graph construction yielding a perfect tradeoff between number of edges and number of rounds required to emulate the all to all communication. Then, we have shown how to emulate the clique by a random graph in $\mathcal{G}_{n,p}$ in $O(\min\{\frac{1}{p^2}, np\})$ rounds, w.h.p. Hence, on dense random graphs (i.e., $p = \Omega(1)$), our simulation performs in just a multiplicative constant factor away from the optimal, and, on sparse graphs (i.e., $p \simeq \sqrt{\log n/n}$), it performs just a $\log n$ factor away from the optimal. However, in general, whenever $p \gg \frac{1}{\sqrt[3]{n}}$, it performs in $O(\frac{1}{p^2})$ rounds, which is a factor $O(\frac{1}{p})$ away from the trivial lower bound $\Omega(\frac{1}{p})$. Ghaffari et al. [46] recently improved this result to $O(\frac{1}{p} + \log n)$. An intriguing question is whether the n-node clique can be simulated by $\mathcal{G}_{n,p}$ in just $O(\frac{1}{p})$ rounds. Also, our result shows that it is possible to emulate the clique on random graphs efficiently, but it does not provide a way to distributedly build the required routing schema. This is left as an open problem.

Chapter 6

MST In Core-Periphery Networks

In this chapter we investigate the MST construction problem in Core-Periphery networks. We provide a deterministic algorithm that solves the task in $O(\log n)$ rounds, improving an existing randomized algorithm that requires $O(\log^2 n)$ rounds. This chapter is based on results published in [12].

6.1 Introduction

In [8], Avin, Borokhovich, Lotker, and Peleg showed that Core-Periphery networks can be a powerful alternative to the Congested Clique model. They studied various problems, such as MST construction, median and mode finding, matrix transposition and multiplication. For many problems they provided algorithms that construct solutions efficiently. They also showed that if only *two* axioms hold, then it is possible to prove lower bounds that are much higher than the upper bounds that hold when the graph satisfies all the *three* axioms of the Core-Periphery networks (Figure 6.1 shows graphs that satisfy only two out of the three axioms). For the Minimum Spanning Tree construction problem, they provided a randomized algorithm running in $O(\log^2 n)$ rounds.

We revisit one of the main problems left open in [8], namely the complexity of MST construction in the core-periphery model. In fact, in [8], authors show a randomized algorithm that solves MST in $O(\log^2 n)$ rounds. Hence, an open question was to investigate if there is a more efficient deterministic algorithm that solves MST.

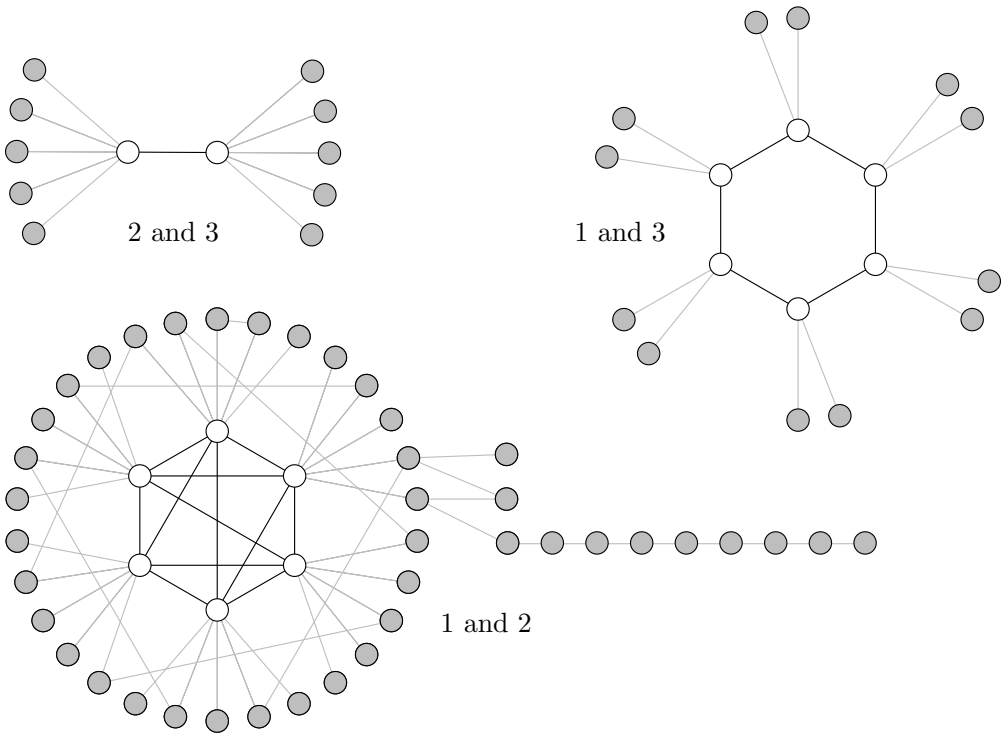

FIGURE 6.1: Graphs that satisfy only 2 axioms of Core-Periphery networks.

6.2 Results

We design a fast deterministic MST construction algorithm for Core-Periphery networks under the CONGEST model. We show that there exists a distributed MST construction algorithm performing in $O(\log n)$ rounds, improving the randomized algorithm in [8] by a factor of $\Theta(\log n)$.

6.3 MST Construction

In [8], a randomized algorithm for Minimum Spanning Tree (MST) construction is presented. It runs in $O(\log^2 n)$ rounds with high probability. We improve this result by describing a deterministic algorithm for MST construction that runs in just $O(\log n)$ rounds. Recall that, for the MST construction task, every node is given as input the weight $w(e)$ of each of its incident edges e. These weights are supposed to be of values polynomial in the size n of the network, and thus each weight can be stored on $O(\log n)$ bits. The output of every node is a set of incident edges, such that the collection of all outputs forms an MST of the network. Without loss of generality, all weights are supposed to be different (since,

otherwise, it is sufficient to add to each edge the identities of the extremities of that edge).

Theorem 6.1. *The MST construction task can be solved in $O(\log n)$ rounds in core-periphery networks under the* CONGEST *model.*

Proof. As usually in the distributed setting, the general idea of the algorithm is based on the sequential Borůvka's algorithm for MST construction, consisting in merging subtrees called *fragments*. Recall that, in Borůvka's algorithm, there are initially n fragments, where each node alone forms a fragment. Each fragment has an ID. Initially, the identity of each fragment is the ID of the single node in the fragment. Then the algorithm proceeds in at most $\lceil \log_2 n \rceil$ phases. At each phase, each fragment F computes the edge e_F of minimum weight incident to fragment F, and adds it to the MST. Fragments connected by such an edge merge, and a new phase begins. This procedure is repeated until there is only one fragment, which is the desired MST.

We first present a (deterministic) distributed algorithm running in $O(\log^2 n)$ rounds in core-periphery networks. This algorithm is composed of at most $\lceil \log_2 n \rceil$ phases, where each phase requires $O(\log n)$ rounds. Then, we show how to perform each phase in $O(1)$ rounds, obtaining the desired $O(\log n)$ rounds algorithm. Recall that a core-periphery network satisfies the three axioms listed in Section 2.5 where C and P denote the sets of nodes in the core and in the periphery, respectively.

The algorithm starts by an initialization phase, where each node in the periphery looks for a node in the core, which will be its *representative*. By Axiom 3 all nodes in the periphery can concurrently send messages to the core so that each message will be received by at least one node in the core after $O(1)$ rounds. So, each node in the periphery sends a request for a representative by sending its own ID to the core. Every node in the periphery then waits for an acknowledgment from nodes in the core that accepted its request. These acknowledgements follow the same route as the corresponding requests, backward. Hence, all acknowledgments are also received after $O(1)$ rounds. Every node takes as representative the core node whose acknowledgment reaches that node first. If a node receives several acknowledgments simultaneously, then it selects the one with the smallest ID. By Axiom 1, each node in the core can be the representative of at most $O(|C|)$ nodes in the periphery because its degree is at most

$O(|C|)$, and thus it can receive at most $O(|C|)$ messages in $O(1)$ rounds. Every node in the core is its own representative.

We assume that the nodes in the core are sorted according to their IDs (this operation can be done in $O(1)$ rounds using all-to-all and Axiom 2). For every node in the core, we denote by $\text{succ}(u)$ and $\text{pred}(u)$ the successor and the predecessor of u in this order, respectively.

We heavily use the protocols in [65]. Note that the *routing* protocol in [65] requires that each node is the the source and destination of at most n messages. However, it can be trivially adapted to be applied with $O(n)$ messages, still requiring $O(1)$ rounds. Similarly, the *sorting* protocol in [65] requires that each node receives at most n keys, but, again, it can be trivially modified for allowing each node to receive $O(n)$ keys, still requiring $O(1)$ rounds.

We now explain how every phase of Borůvka's algorithm is performed.

1. Every node sends the ID of its fragment to all its neighbors.
2. Let $r(v) \in C$ and $\text{id}(F)$ be the representative and the ID of the fragment F of node v, respectively. We denote by $e_F(v)$ the edge of minimum weight incident to v and connecting v to a node not in its fragment F. Each node v in the periphery sends $(e_F(v), w(e_F(v)), \text{id}(F), \text{id}(F'))$ to $r(v)$, where the tail of $e_F(v)$ belongs to F, and its head belongs to fragment $F' \neq F$. Observe that each node in the core receives $O(|C|)$ such messages.
3. Every node in the core, upon reception of 4-tuple

   $$(e_F(v), w(e_F(v)), \text{id}(F), \text{id}(F'))$$

   from the nodes that it represents (including itself), selects the ones with minimum weight for each fragment F. We denote by S_1 the set of the selected edges by all nodes in the core. Note that $|S_1| = O(|C|^2)$.
4. The algorithm assigns a *leader* to each fragment. The leaders are core nodes chosen in such a way that the fragments are equally distributed among leaders. Let

   $$x = \lceil |S_1|/|C| \rceil.$$

Note that $x = O(|C|)$. Given a fragment F, its leader is

$$\ell(F) = 1 + \left\lfloor \frac{|\{(u,v) \in S_1 : \mathrm{id}(F_u) < \mathrm{id}(F)\}|}{x} \right\rfloor$$

where F_u is the fragment of u. Note that $1 \leq \ell(F) \leq |C|$. For each fragment F, all edges incident to F in S_1 are sent to $\ell(F)$ by its representative holding such edges — we shall explain hereafter how this is implemented in core-periphery networks. In this way each leader can select the edge e_F of minimum weight incident to fragment F. Let S_2 be the set of all edges e_F, where F is a fragment.

5. The algorithm then aims at merging the fragments. We call *merge tree* a tree whose nodes are fragments F, and whose edges are the edges e_F connecting these fragments. Note that, in a merge tree, there are two adjacent fragments F and F' connected by two possibly distinct edges e_F and $e_{F'}$. The fragment with smallest ID that is extremity of such an edge is the root of the merge tree. The algorithm proceeds so that each leader $\ell(F)$ of a fragment F in the merge tree becomes aware of the root of the tree. The ID of this root will become the ID of the fragment resulting from merging all the fragments in the merge tree. It is possible to find the root of a tree of height h in $O(\log h)$ steps using *pointer jumping* — we shall explain hereafter how this is precisely implemented in core-periphery networks.

6. By the previous step, for every fragment F, its leader $\ell(F)$ knows the ID of the merge tree it belongs to. Moreover, for each edge (u,v) that was received by a leader from the representative $r(u)$ in step 4, the leader saved $\mathrm{id}(r(u))$. This allows leaders to notify the right representatives of the ID of the root of the merge tree.

7. Finally, the ID of every merged fragment is sent to every node v of the periphery from its representative $r(v)$ in the core.

It remains to explain how steps 4 and 5 are actually performed.

Step 4 in more details. First, observe that the parameter $x = \lceil |S_1|/|C| \rceil$ can be computed at each node of the core, as performing all-to-all communication in the core allows each core node to compute $|S_1|$. Now, we show how to distribute

the fragments among the leaders such that leader $\ell(F)$ becomes aware of the edges $e_F(v) \in S_1$ incident to F.

The edges $(u, v) \in S_1$ are sorted according to the ID of the fragment F_u its tail belongs to, and are then split into groups of x edges. Again, this operation can be done in $O(1)$ rounds using the sorting protocol in [65] because $x = O(|C|)$. The k-th group is assigned to the k-th node of the core.

Let us consider a core node u, and let $\mathcal{F}(u)$ be the set of fragments F such that $\ell(F) = u$. Let us denote by $\mathrm{id}_{max}(u)$ (resp., $\mathrm{id}_{min}(u)$) the maximum ID (resp., minimum ID) of the fragments $F \in \mathcal{F}(u)$. Having sorted the set S_1 guaranties that the leader u receives all the edges assigned to it, except perhaps some edges starting from fragment $\mathrm{id}_{max}(u)$ that could have been delivered to $\mathrm{succ}(u)$. However, there are at most $x - 1$ such edges, since the representatives kept at most one edge per fragment. So, every core node u can send $\mathrm{id}_{max}(u)$ to $\mathrm{succ}(u)$, in order to let that node know that the leader of the fragment with ID equal to $\mathrm{id}_{max}(u)$ should be u, and not $\mathrm{succ}(u)$. Since each node u has then at most $x - 1$ messages to transmit to $\mathrm{pred}(u)$, we can transmit these messages using the routing protocol in [65]. Now each leader u has all the outgoing edges of each fragment F with $\ell(F) = u$. Thus, u can compute e_F for each of these fragments. Finally, each node u in the core broadcasts the pair $(\mathrm{id}_{min}(u), \mathrm{id}_{max}(u))$ in the core so that every node in C learns the leader of each fragment.

Note that, while sorting and routing, every node keeps track of the ID of the representative nodes which originally received every edge that is manipulated by that node (this is needed in step 6).

Step 5 in more details. We show how to perform the first step of pointer jumping. Recall that, for every fragment F, the leader $\ell(F)$ knows e_F. This latter edge is the one leading toward the root of the merge tree. Assume that $e_F = (u, v)$, with $u \in F$ and $v \in F'$. The objective for the leader $\ell(F)$ is to learn to which fragment F'' is pointing the edge $e_{F'} = (u', v')$ with $u \in F'$ and $v' \in F''$. In other words, if p denotes the parent relation in a merge tree, the leader $\ell(F)$ of fragment F wants to learn the ID of $p(p(F))$. The bad news is that $\ell(F)$ cannot directly ask $\mathrm{id}(p(p(F)))$ to $\ell(p(F))$ because this could create a bottleneck at $\ell(p(F))$. Nevertheless this issue can be overcame as follows.

First, the edges in S_2 are sorted according to the IDs of the fragment of their heads, and grouped into groups whose heads belong to the same fragment. In this way, only one request is sent for each group (to the leader of the corresponding fragment). Since $x = \lceil |S_1|/|C| \rceil$, we have $x = O(|C|)$, and thus the number of requests that each leader has to make is at most $O(|C|)$.

Second, every leader does not receive more than $O(|C|)$ requests. Indeed, let $q_{u,v}$ be the number of different fragments for which a node u in the core has to send a request to leader v. Let $F_{i_1}, F_{i_2}, \ldots, F_{i_{q_{u,v}}}$ be these fragments, with $\ell(F_{i_1}) = \ell(F_{i_2}) = \cdots = \ell(F_{i_{q_{u,v}}}) = v$, and $i_1 < i_2 < \cdots < i_{q_{u,v}}$. Recall that the edges in S_2 are sorted according to the IDs of the fragment of their heads. Thus, if $q_{u,v} > 1$ then the fragments $F_{i_2}, \ldots, F_{i_{q_{u,v}}}$ do not appear in any list of fragments assigned to nodes with identity smaller than $\mathrm{id}(u)$. Therefore, leader v receives at least $\sum_{u \in C}(q_{u,v} - 1)$ requests for different fragments. On the other hand, every core node v is the leader of at most x fragments. Therefore $\sum_{u \in C}(q_{u,v} - 1) \leq x$. Hence the number of requests received by v is $\sum_{u \in C} q_{u,v} = O(|C|)$.

These two facts, allow the routing protocol in [65] to be used, for sending the requests to the leaders, and for receiving back their answers. Once this is done, every node u sends $\mathrm{id}(p(p(F)))$ to $\ell(F)$, for every $F \in \mathcal{F}(u)$ in a constant number of rounds, again using [65]. It follows that every leader u can learn the ID of $p(p(F))$ for every $F \in \mathcal{F}(u)$ in a constant number of rounds.

Time analysis. The initialization phase can be performed in a constant number of rounds thanks to Axiom 3. Step 1 trivially requires $O(1)$ rounds. Step 2 also requires $O(1)$ rounds thanks to Axiom 3. Step 3 is executed locally by each node, thus it does not require communication. Step 4 can be executed in $O(1)$ rounds using the sorting protocol in [65] because $x = O(|C|)$. Step 6 can also be performed in $O(1)$ rounds using the routing protocol in [65] because each leader handles $O(|C|)$ edges (for which it has to send a fragment ID), and each representative has to receive $O(|C|)$ messages (one for each edge it has to receive a new fragment ID). The last step is the inverse of step 2, and thus can still be executed in $O(1)$ rounds. Step 5 however requires $O(\log n)$ rounds because the merge tree might be of height $\Omega(n^\epsilon)$ for some $\epsilon > 0$. Since the number of phases is also $O(\log n)$, the total number of rounds of this algorithm is $O(\log^2 n)$.

A faster algorithm. Now, we describe how to modify the above algorithm so that it uses only $O(1)$ rounds for each phase, hence $O(\log n)$ rounds in total. Since the only step that requires a non constant number of rounds is Step 5, we show how to perform that step in $O(1)$ rounds.

The idea is to use a technique introduced first in [70], and also used in Avin et al. [8], called *amortized pointer jumping*. The reduction of long chains of pointers is deferred to later phases of Borůvka's algorithm, and only a constant number of pointer jumps are performed at each phase. This technique exploits the fact that, if a chain is long, it must contain many fragments. As a consequence, when pointer jumping completes, the resulting fragment is quite large, and other nodes involved in small fragments may continue building the MST in parallel, without waiting for large fragments to be constructed.

We show how to do a constant number of pointer jumping steps, then freezing the procedure, and resuming it later in the next phase of Borůvka's algorithm. At each step of pointer jumping, every leader u can know, for every $F \in \mathcal{F}(u)$, if the root of the merge tree has been reached. Suppose that the root has not been reached by u after a constant number of pointer jumping (i.e., the leader does not know yet the new ID of the merged fragment), and that u is currently pointing at fragment F'. In the following, node u adds a flag in its messages, which specifies that the fragment has not been resolved yet, and that it stopped at F'. This flag will be propagated to all nodes that proposed edges that start from unresolved fragments. At the next phase of Borůvka's algorithm, these nodes will propose again the same edges, by specifying also F'. Fragment F' will be used as if it was the destination fragment of the edge. In this way, for every fragment F in a merge tree whose merging has not yet been performed, the same edge e_F as before will be chosen, and other steps of pointer jumping will be performed. This insures that nodes belonging to fragments in such merge trees do not propose new edges, thus emulating a full execution of pointer jumping.

After having reduced the number of rounds for performing step 5 from $O(\log n)$ to $O(1)$, amortized, we get that the resulting algorithm just requires $O(\log n)$ rounds to construct a MST. □

6.4 Conclusions

In this work, we have proposed a deterministic MST construction algorithm for core-periphery networks that performs in $O(\log n)$ rounds, improving the previously known (randomized) algorithm by a factor $\Theta(\log n)$. Recent advances in the Congested Clique model demonstrate that ultra fast MST algorithms exist for this later model, namely, a $O(1)$-round randomized algorithm [59], and a $O(\log \log n)$-round deterministic algorithm [69]. Since no lower bounds have been proved for Core-Periphery networks, an intriguing question is whether such ultra fast algorithms exist for this model.

Chapter 7

Conclusions and Open Problems

In this thesis we studied different aspects of distributed computing. In Chapter 3 we focused on problems related to subgraph detection. We initially showed how to detect any fixed tree T of constant size in a constant number of rounds. We then applied this result to distributed property testing, by showing that for any pattern H composed by an edge e, a tree T, and arbitrary connections between the nodes of e and the nodes of T, we can test H-freeness in constant time in graphs that are far from being H-free. This result allows to distributedly detect the presence of important patterns, like cycles C_k for any $k \geq 3$, the clique of four nodes K_4, complete bipartite graphs $K_{2,k}$ for any $k \geq 1$, or two nodes connected by k paths of constant length.

The graph of minimum size that is not contained in the aforementioned class is the clique of size five, K_5. For this particular pattern we do not know if it is possible to test its absence in constant time. In this case, we can still easily find in constant time an edge that is part of K_5, but then we would need to detect a triangle composed by nodes that are all connected to the endpoints of the chosen edge. This problem finds analogies in the coordinator model, where all the nodes of a graph are connected to a central authority that helps the nodes to solve a task. In our setting the coordinator would be one of the two endpoints. Since it is not known how to detect triangles in the coordinator model (we do not even know how to solve it in constant time in a very strong model such as the Congested Clique), this does not seem to be the right way to address the problem of detecting K_5. Instead, it would be a really interesting research question to understand something more general related to distributed property

testing, that is, if the ability of efficiently choose a "good" edge is all what this model can offer (and thus be equivalent to the coordinator model), or if we can exploit the presence of many copies of H in a more clever way.

In Chapter 4 we examined the role of the bandwidth in distributed computing, by investigating how much the speed of a distributed algorithm can scale while changing the amount of allowed bandwidth. We modified algorithms designed to use messages of size $O(\log n)$, to be time efficient when more bandwidth is allowed. We showed that different problems exhibit different behaviors. For example, while the time complexity of the APSP problem fully scales with the available bandwidth, the time complexity of the MST problem does not. Since in real world networks the latency could dominate the computational time, one could be interested in reducing the number of rounds at the cost of using more bandwidth. This work suggests that when we provide algorithms for the CONGEST model, we should analyze it on a wider spectrum of bandwidths, since limiting the analysis only to the case where the available bandwidth is logarithmic could hide some important details about the nature of the problem. It would be really interesting to further investigate this phenomenon, that is, it may be the case that there is a direct connection between time complexity and ability to scale. For example, it would be nice to understand if all problems that require linear time, like APSP, can fully scale with the available bandwidth, and, on the other hand, if all problems that can be solved in sublinear time, like MST, can not.

In Chapters 5 and 6 we studied problems related to the *Core-Periphery* model. First, we showed graphs that are able to emulate the clique communication efficiently, by first providing graphs that achieve an optimal tradeoff between the number of edges and the rounds required to perform the all-to-all communication, and then by showing that Erdős-Rényi random graphs $\mathcal{G}_{n,p}$ can emulate the clique communication efficiently. Then, we showed an efficient way to solve the Minimum Spanning Tree construction problem in this model. Core-Periphery networks are a nice alternative to the Congested Clique model, since they can solve many tasks efficiently, while still using a number of edges that is linear in the number of nodes. For this model no lower bounds are known, and an intriguing question is to understand the real power of Core-Periphery networks, for example by trying to see if ultra fast MST algorithms designed

for the Congested Clique can be used in this model to solve the MST problem in $o(\log n)$ rounds.

Bibliography

[1] Amir Abboud, Keren Censor-Hillel, and Seri Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In *30th International Symposium in Distributed Computing (DISC)*, pages 29–42, 2016.

[2] Noga Alon, Sonny Ben-Shimon, and Michael Krivelevich. A note on regular ramsey graphs. *Journal of Graph Theory*, 64(3):244–249, 2010.

[3] Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000.

[4] Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM J. Discrete Math.*, 22(2):786–819, 2008.

[5] Noga Alon and Asaf Shapira. A characterization of easily testable induced subgraphs. *Combinatorics, Probability & Computing*, 15(6):791–805, 2006.

[6] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[7] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[8] Chen Avin, Michael Borokhovich, Zvi Lotker, and David Peleg. Distributed computing on core-periphery networks: Axiom-based design. In *41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 399–410, 2014.

[9] Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti. What can be verified locally? In *34th Symposium on Theoretical Aspects of Computer Science (STACS)*, 2017.

[10] Alkida Balliu, Michele Flammini, Giovanna Melideo, and Dennis Olivetti. Nash stability in social distance games. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 342–348, 2017.

[11] Alkida Balliu, Michele Flammini, and Dennis Olivetti. On pareto optimality in social distance games. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 349–355, 2017.

[12] Alkida Balliu, Pierre Fraigniaud, Zvi Lotker, and Dennis Olivetti. Sparsifying congested cliques and core-periphery networks. In *Structural Information and Communication Complexity - 23rd International Colloquium, SIROCCO 2016, Helsinki, Finland, July 19-21, 2016, Revised Selected Papers*, pages 307–322, 2016.

[13] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Approximate undirected transshipment and shortest paths via gradient descent. *CoRR*, abs/1607.05127, 2016.

[14] Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Computing*, 24(2):79–89, 2011.

[15] Andrei Z. Broder, Alan M. Frieze, and Eli Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM J. Comput.*, 23(5):976–989, 1994.

[16] Luciana Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *25th ACM Symposium on Principles of Database Systems (PODS)*, pages 253–262, 2006.

[17] Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. In *30th Int. Symposium on Distributed Computing (DISC)*, volume 9888 of *LNCS*, pages 43–56. Springer, 2016.

[18] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 143–152, 2015.

[19] Keren Censor-Hillel and Tariq Toukan. On fast and robust information spreading in the vertex-congest model. In *22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 270–284, 2015.

[20] David Conlon and Jacob Fox. Graph removal lemmas. *CoRR*, abs/1211.3487, 2012.

[21] Artur Czumaj, Oded Goldreich, Dana Ron, C. Seshadhri, Asaf Shapira, and Christian Sohler. Finding cycles and trees in sublinear time. *Random Struct. Algorithms*, 45(2):139–184, 2014.

[22] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *43rd ACM Symposium on Theory of Computing (STOC)*, pages 363–372, 2011.

[23] Danny Dolev, Christoph Lenzen, and Shir Peled. Tri, tri again: Finding triangles and small subgraphs in a distributed setting. In *26th International Symposium on Distributed Computing*, pages 195–209, 2012.

[24] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 367–376, 2014.

[25] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 359–368, 2004.

[26] Michael Elkin. Distributed exact shortest paths in sublinear time. *CoRR*, abs/1703.01939, 2017.

[27] Michael Elkin. A simple deterministic distributed MST algorithm, with near-optimal time and message complexities. *CoRR*, abs/1703.02411, 2017.

[28] Yuval Emek, Christoph Pfister, Jochen Seidel, and Roger Wattenhofer. Anonymous networks: randomization = 2-hop coloring. In *33rd ACM Symposium on Principles of Distributed Computing*, pages 96–105, 2014.

[29] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3), 1999.

[30] Paul Erdös, Peter Frankl, and Vojtech Rödl. The asymptotic number of graphs not containing a fixed subgraph and a problem for hypergraphs having no exponent. *Graphs and Combinatorics*, 2(1):113–121, 1986.

[31] Paul Erdős, András Hajnal, and J. W. Moon. A problem in graph theory. *The American Mathematical Monthly*, 71(10):1107–1110, 1964.

[32] Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. Three notes on distributed property testing. In *31th International Symposium in Distributed Computing (DISC)*, 2017.

[33] Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Struct. Algorithms*, 1(4):447–460, 1990.

[34] Laurent Feuilloley and Pierre Fraigniaud. Randomized local network computing. In *27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 340–349, 2015.

[35] Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bulletin of the EATCS*, 119:41–65, 2016.

[36] Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. A hierarchy of local decision. In *43rd Int. Colloquium on Automata, Languages, and Programming (ICALP)*, pages 118:1–118:15, 2016.

[37] Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, and David Peleg. Randomized distributed decision. *Distributed Computing*, 27(6):419–434, 2014.

[38] Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *J. ACM*, 60(5):35:1–35:26, 2013.

[39] Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. In *29th ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2017.

[40] Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *30th Int. Symposium on Distributed Computing (DISC)*, volume 9888 of *LNCS*, pages 342–356. Springer, 2016.

[41] Alan M. Frieze. Disjoint paths in expander graphs via random walks: A short survey. In *Second International Conference on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 1–14, 1998.

[42] Alan M. Frieze. Edge-disjoint paths in expander graphs. In *11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 717–725, 2000.

[43] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1162, 2012.

[44] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[45] Juan A. Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302–316, 1998.

[46] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and routing in almost mixing time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 131–140, 2017.

[47] Mohsen Ghaffari and Merav Parter. Mst in log-star rounds of congested clique. In *35th ACM Symposium on Principles of Distributed Computing (PODC)*, 2016.

[48] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[49] Oded Goldreich and Luca Trevisan. Three theorems regarding testing graph properties. *Random Struct. Algorithms*, 23(1):23–57, 2003.

[50] Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(1):1–33, 2016.

[51] James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 91–100, 2015.

[52] James W. Hegeman and Sriram V. Pemmaraju. Lessons from the congested clique applied to MapReduce. *Theor. Comput. Sci.*, 608:268–281, 2015.

[53] James W. Hegeman, Sriram V. Pemmaraju, and Vivek Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. In *28th Int. Symposium on Distributed Computing (DISC)*, pages 514–530, 2014.

[54] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 489–498, 2016.

[55] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 355–364, 2012.

[56] Qiang-Sheng Hua, Haoqiang Fan, Lixiang Qian, Ming Ai, Yangyang Li, Xuanhua Shi, and Hai Jin. Brief announcement: A tight distributed algorithm for all pairs shortest paths and applications. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 439–441, 2016.

[57] Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *ACM Symposium on Principles of Distributed Computing (PODC)*, 2017.

[58] Stasys Jukna and Georg Schnitger. Triangle-freeness is hard to detect. *Combinatorics, Probability, & Computing*, 11(6):549–569, 2002.

[59] Tomasz Jurdzinski and Krzysztof Nowicki. MST in O(1) rounds of the congested clique. *CoRR*, abs/1707.08484, 2017.

[60] Valerie King, Shay Kutten, and Mikkel Thorup. Construction and impromptu repair of an MST in a distributed network with o(m) communication. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 71–80, 2015.

[61] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010.

[62] Shay Kutten and David Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

[63] Tom Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann, 1992.

[64] Tom Leighton, Satish Rao, and Aravind Srinivasan. Multicommodity flow and circuit switching. In *31st Hawaii International Conference on System Sciences*, pages 459–465, 1998.

[65] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 42–50, 2013.

[66] Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages: extended abstract. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 381–390, 2013.

[67] Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 153–162, 2015.

[68] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 375–382, 2013.

[69] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005.

[70] Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for constant diameter graphs. In *20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 63–71, 2001.

[71] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[72] Burkhard Monien. How to find long paths efficiently. In *Analysis and design of algorithms for combinatorial problems*, volume 109 of *North-Holland Math. Stud.*, pages 239–254. North-Holland, Amsterdam, 1985.

[73] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *ACM Symposium on Theory of Computing (STOC)*, pages 565–573, 2014.

[74] Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.

[75] Noam Nisan and Avi Wigderson. Rounds in communication complexity revisited. *SIAM J. Comput.*, 22(1):211–219, 1993.

[76] Dennis Olivetti. How bandwidth affects the CONGEST model. *CoRR*, abs/1704.06092, 2017.

[77] Hiroaki Ookawa and Taisuke Izumi. Filling logarithmic gaps in distributed complexity for global problems. In *41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 8939 of *LNCS*, pages 377–388. Springer, 2015.

[78] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. A time- and message-optimal distributed algorithm for minimum spanning trees. *CoRR*, abs/1607.06883, 2016.

[79] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artif. Intell.*, 29(3):241–288, 1986.

[80] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[81] David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 660–672, 2012.

[82] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.

[83] Ron Shamir and Dekel Tsur. Faster subtree isomorphism. *J. Algorithms*, 33(2):267–280, 1999.

[84] Julian R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.