Optimal Distributed All Pairs Shortest Paths and Applications

Preliminary full version - please check for updates.

Stephan Holzer Roger Wattenhofer
ETH Zurich, Switzerland
{stholzer, wattenhofer}@tik.ee.ethz.ch

22.05.2012

Abstract

We present an algorithm to compute All Pairs Shortest Paths (APSP) of a network in a distributed way. The model of distributed computation we consider is the message passing model: in each synchronous round, every node can transmit a different (but short) message to each of its neighbors. We provide an algorithm that computes APSP in $\mathcal{O}(n)$ communication rounds, where n denotes the number of nodes in the network. This implies a linear time algorithm for computing the diameter of a network. Due to a lower bound these two algorithms are optimal up to a logarithmic factor. Furthermore, we present a new lower bound for approximating the diameter D of a graph: Being allowed to answer $D+1$ or D can speed up the computation by at most a factor D. On the positive side, we provide an algorithm that achieves such a speedup of D and computes an $1+\varepsilon$ multiplicative approximation of the diameter. We extend these algorithms to compute or approximate other problems, such as girth, radius, center and peripheral vertices. At the heart of these approximation algorithms is the S-Shortest Paths problem which we solve in $\mathcal{O}(|S|+D)$ time.

1 Introduction

In networks, basically two types of routing algorithms are known: distance-vector and link-state. Link-state algorithms embody the school of centralized algorithms. First all the information about the network graph is collected, and then optimal routes between all nodes are computed, using an efficient centralized algorithm. Distance-vector routing protocols on the other hand represent the school of distributed algorithms. Nodes update their routing tables by constantly exchanging messages with their neighbors. Both approaches are used in the Internet, link-state for instance in OSPF or IS-IS, distance-vector in RIP or BGP[1]. Among network researchers there is a vivid debate on which approach is better. After all, both approaches essentially do the same thing – compute shortest paths between all nodes – a problem known as all pairs shortest paths (APSP). Despite its practical relevance, the distributed time-complexity of APSP was so far not known.

In this paper we present a new distributed algorithm that computes APSP in $\mathcal{O}(n)$ time. Because of a recent lower bound for computing the diameter [22], this APSP-algorithm is essentially optimal (up to a logarithmic factor). In addition this demonstrates that computing the diameter has about the same complexity as computing APSP in a distributed setting. These statements contrast the sequential setting: It is open to provide matching upper/lower bounds for APSP or to show whether determining the diameter of a graph can be done faster than computing APSP (or performing matrix multiplication.)

In addition, we present a new lower bound for approximating the diameter D of a graph: Being allowed to answer $D + 1$ (in addition to the correct answer D) can speed up the computation by at most a factor D. On the bright side, we provide an algorithm that achieves a speedup of D and computes an $1 + \varepsilon$ multiplicative approximation of the diameter. We extend these algorithms to compute/approximate other problems, such as girth, radius, center and peripheral vertices. At the heart of these approximation algorithms is the S-Shortest Paths problem. We essentially show that s breadth-first searches can be computed in time $\mathcal{O}(s + D)$ which is of interest on its own.

2 Model and Basic Definitions

Model: We study the message passing model with limited bandwidth (also known as CONGEST model, [32]): Our network is represented by an undirected unweighted graph $G = (V, E)$. Nodes V correspond to processors (computers or routers), two nodes are connected by an edge from set E if they can communicate directly with each other. We denote the number of

[1]These are presented in many textbooks, e.g. [6].

nodes of a graph by n, and the number of its edges by m. Furthermore we assume that each node has an unique identifier (ID) in the range of $\{1, \ldots, 2^{\mathcal{O}(\log n)}\}$, i.e. each node can be represented by $\mathcal{O}(\log n)$ bits. We assume that n is known to each node and there is a node with ID 1. These are valid assumptions since the time to compute n or to find the node with smallest ID and rename it to 1 would not affect the asymptotic runtime of the presented algorithms. For simplicity, we refer to $u \in V$ not only as a node, we use u to refer to u's ID as well when this is clear from the context. Nodes initially have no knowledge of the graph G – except that they know their immediate neighborhood. By $N_k(v)$ we denote the k-neighborhood of v, that is all nodes in G that can be reached from v using k hops/edges. We define that $v \in N_1(v)$. Given a set $S \subseteq V$, set $N_k(S)$ denotes the k-neighborhood of S, that is $\cup_{v \in S} N_k(S)$.

We consider a synchronous communication model, where every node can send B bits of information over all its edges in one synchronous round of communication. In principle it is allowed that in a round, a node can send different messages of size B to each of its neighbors (and likewise receive different messages from each of its neighbors). Typically we have $B = \mathcal{O}(\log n)$ bits, which allows us to send a constant number of node or edge IDs per message. Since communication cost usually dominates the cost of local computation, local computation is considered free. We are interested in the number of rounds that a distributed algorithm needs until a problem is solved – this is the time complexity of the algorithm. By solving a problem we refer to evaluating a function $h : \mathbb{C}_n \to SOL$ over the underlying network-structure. Here $\mathbb{C}_n$ is the set of all graphs over n vertices and SOL is e.g. $\{0, 1\}$ or $\mathbb{N}$. We define distributed round complexity as follows:

Definition 1. *(distributed round complexity). Let $\mathcal{A}_\varepsilon$ be the set of distributed algorithms that use (public) randomness (indicated by "pub") and evaluate a function h on the underlying graph G over n nodes with an error probability smaller than ε. Denote by $R_\varepsilon^{dc-pub}(A(G))$ the distributed round complexity (indicated by "dc") representing the number of rounds that an algorithm $A \in \mathcal{A}_\varepsilon$ needs in order to compute $h(G)$ on G. We define*

$$R_\varepsilon^{dc-pub}(h) = \min_{A \in \mathcal{A}_\varepsilon} \max_{G \in \mathbb{C}_n} R_\varepsilon^{dc-pub}(A(G))$$

to be the smallest amount of rounds any algorithm needs in order to compute h.

Throughout the paper we often state results in a less formal way. Example: when h is *diam* (the function that maps a graph to its diameter) and $R_0^{dc-pub}(diam) = \mathcal{O}(n)$, we often just write "the deterministic round complexity of computing the diameter is $\mathcal{O}(n)$".

Let us denote by $d(u, v)$ the distance of nodes u and v in G, which is the length of a shortest $u - v$-path in G. The problems we consider are:

Definition 2. *(APSP, S-SP) Let $G = (V, E)$ be a graph. The all pairs shortest paths (APSP) problem is to compute the shortest paths between any pair of vertices in $V \times V$. In the S-Shortest Paths (S-SP) problem, we are given a set $S \subseteq V$ and need to compute the shortest paths between any pair of vertices in $S \times V$.*

Like in [18], at the end of an S-SP computation, each node in V knows its distances to every node in S. Accordingly we assume that the result of APSP/S-SP is stored in a distributed way as well. Note that there exist graphs where storing all distance information of all pairs at all nodes takes $\Omega(n^2)$ time, such that a distributed approach is crucial.

Definition 3. *(eccentricity,diameter, radius, girth) The eccentricity $ecc(u)$ of a node $u \in V$ is defined to be $ecc(u) := \max_{v \in V} d(u, v)$ and is the maximum distance to any other node in the graph. The diameter $D := \max_{u \in V} ecc(u) = \max_{u,v \in V} d(u, v)$ of a graph G is the maximum distance between any two nodes of the graph. The radius of G denoted by $rad := \min_{u \in V} ecc(u)$ is the minimum eccentricity of any vertex. The girth g of a graph G is the length of the shortest cycle in G. If G is a forest its girth is infinity.*

Definition 4. *(center vertices, peripheral vertices). The center of a graph G is the set of nodes whose eccentricity equals the radius of the graph. A node u is a peripheral vertex of a graph if its eccentricity equals the diameter of the graph.*

Definition 5. *(approximation). Given an optimization problem P, denote by OPT the cost of the optimal solution for P and by sol_A the cost of the solution of an algorithm A for P. Let $\rho \geq 1$. We say A is a $(\times, \rho)$-approximation (multiplicative approximation) for P if $OPT \leq sol_A \leq \rho \cdot OPT$ for any input. Let $\gamma \geq 0$. We say A is a $(+, \gamma)$-approximation (additive approximation) for P if $OPT \leq sol_A \leq OPT + \gamma$ for any input.*

We extend the above definition to sets: assume we are given the problem of computing a set $S_c := \{v \mid cost(v) \leq c\}$ of nodes, where each node has a certain cost. A $(+, k)$-approximation to S_c is any subset of $\{v \mid cost(v) \leq c + k\}$ that includes S_c. As an example, consider the eccentricity of a node as its cost which allows us to think of the center of G as the set of nodes with cost $rad(G)$. A k-approximation to the center in unweighted graphs would be any subsets of the center's k-neighborhood such that $center \subseteq S \subseteq N_k(center)$.

Definition 6. *In the problem of computing/approxima-ting eccentricities, we require that each node in the graph knows (an approximation to) its own eccentricity in the end. In the problem of computing/approximating the diameter/radius /girth , we require that each node in the graph knows (the same estimate of) the networks diameter/radius /girth in the end. In* *the problem of computing (approximations to) the center/peripheral vertices, we require that each node in the graph knows whether it belongs to (the approximation of) the center/peripheral vertices.*

Sometimes we use the following facts and notion of (partial) BFS trees.

Fact 1. *It is well known that the eccentricity of any node is a good multiplicative approximation of the diameter: For any node $u \in V$ we know that $ecc\,(u) \leq D \leq 2 \cdot ecc\,(u)$.*

Fact 2. *If G is not a forest, then the girth can be bounded using the diameter by $g \leq 2 \cdot D + 1$.*

Definition 7. *(k-BFS tree). A (partial) k-BFS tree rooted in v is the subtree of a BFS tree rooted in v that contains only the nodes at distance at most k to v.*

During the paper we denote a set $\{0, \ldots, k\}$ by $[k]$ and the degree of a node by $d(v)$.

3 Related Work and Our Contributions

3.1 All Pairs Shortest Paths

In the synchronous model, a link-state APSP algorithm will finish in D time, since nodes have to learn about all the edges. Likewise a distance-vector APSP algorithm will finish in D time: Nodes with distance d will learn about each other in round d. So both algorithms have the same time complexity. However, both algorithms severely violate our restriction for message size! A link-state algorithm must exchange information about all edges, hence potentially messages are of a size which is quadratic in the number of nodes. Likewise, nodes in a distance-vector algorithm may have to send routing table updates about almost all the nodes in each round. In real networks, one can often not exchange information about all nodes in one single message. If we restrict link-state and distance-vector algorithms to messages of size $\mathcal{O}(\log n)$ (by serializing the long messages), they will need strictly superlinear (and sometimes quadratic) time.

Due to its importance in network design, shortest path-problems in general and the APSP problem in particular were among the earliest studied problems in distributed computing. Developed algorithms were immediately used e.g. as early as in 1969 in the ARPANET (see [28], p.506). Routing messages via shortest paths were extensively discussed to be beneficial in [11, 29, 30, 38, 41] and in many other papers. It is not surprising that there is plenty of literature dealing with algorithms for distributed APSP, but most of them focused on secondary targets such as trading time for message complexity. E.g. papers [1, 12, 45] obtain a communication complexity of roughly $\mathcal{O}(n \cdot m)$ bits/messages and still require superlinear runtime. Also a lot of effort was spent to obtain fast sequential algorithms for various versions of computing APSP or related problems such as the diameter problem, e.g. [3, 4, 7, 13, 39, 40]. These algorithms are based on fast matrix multiplication such that currently the best runtime is $\mathcal{O}(n^{2.3727})$ due to [46]. Despite these advances the nature of distributed computing makes it unlikely to design fast algorithms based on matrix multiplication. It seems that combinatorial algorithms for APSP (not using fast matrix multiplication) are much better suited to be implemented in a distributed way. Combinatorial APSP algorithms were studied first in [21] and then [9, 10, 15, 19, 23, 42, 43, 50] but only yield polylogarithmic improvements over $\mathcal{O}(n^3)$.

In this paper we do not follow these approaches but present a simpler algorithm with simple analysis that computes APSP by extending a classical approach to compute APSP that is taught in many lectures: Perform a breadth-first search (BFS) from each node in the graph. The depth of a node in a BFS tree is its distance to the tree's root. Since one computes all BFS trees, all distances are known in the end. This takes time $\mathcal{O}(n^2 + n \cdot m)$ in most sequential models of computing. In the distributed model considered in this paper, this approach (if not modified) takes time $\mathcal{O}(n \cdot D)$ as mentioned e.g. in [18] since each BFS requires $\mathcal{O}(D)$ time. In Section 4.1 we modify this approach by starting the breadth-first searches in a special order at special times. We prove that the chosen start times and the order yield no congestion and thus a linear runtime. This is optimal up to a logarithmic factor due to a lower bound presented in [22] which extended the techniques used in [14].

3.2 S-Shortest Paths

Sometimes one might be satisfied by obtaining approximate distances in the sense that they differ by at most a small additive term. In the literature this problem is known as APASP$_k$ (All Pairs Almost Shortest Paths), where all computed estimates of the distances are at most an additive term k longer than the actual distances. In [2] a sequential algorithm for APASP$_2$ was presented that runs in time $\tilde{O}(\min\{n^{3/2} \cdot m^{1/2}, n^{7/3}\})$. Dor, Halperin and Zwick extended this to APASP$_k$ with a runtime of $\tilde{O}(\min\{n^{2-\frac{2}{k+2}} \cdot m^{\frac{2}{k+2}}, n^{2+\frac{2}{3 \cdot k-2}}\})$ in [17]. This line of research led to *approximate distance oracles* [44], where one is not interested in an additive but a multiplicative error (called stretch). These were recently extended to the distributed setting in [37]. Previously Elkin [18] suggested an approached to obtain distributed algorithms for APASP$_k$ (and APASP with small stretch) by considering almost shortest paths for the S-SP problem (denoted by S-ASP) and mainly focused on the number of bits exchanged. When comparing Elkin's results with our results we need to keep in mind that the aim of the two papers is different, which makes comparison difficult. For our model, the runtime provided in [18] is $\mathcal{O}(|S| \cdot D + n^{1+\xi/2})$ for computing almost shortest paths in the sense that the estimated distance is at most $(1+\varepsilon) \cdot d(u,v) + \beta(\xi,\rho,\varepsilon)$, where $\beta(\xi,\rho,\varepsilon)$ is constant when ξ, ρ and ε are. The runtime of our algorithm for computing exact shortest paths runs in $\mathcal{O}(|S| + D)$ time, which is faster for all parameters. However, our approach requires the exchange of $\mathcal{O}((|S| + D) \cdot m \cdot \log n)$ bits while [18] needs only $\mathcal{O}(m \cdot n^{\rho} + |S| \cdot n^{1+\xi})$ bits. In the case $\max\{S, D\} \leq n^{\rho}/\log n$ and $m \leq n^{1+\xi}/\log n$ and $D \leq \frac{|S| \cdot n^{1+\xi}}{m \cdot \log n}$, our algorithm sends fewer bits. Besides the synchronous model that we study, [18] also investigated an asynchronous setting. While Elkin focused on very precise approximations, the authors of [25] were interested in rather loose approximation factors when considering the S-ASP problem and thus obtained better time complexities. It is stated in Theorem 4.11. of [25] that an (expected) $\mathcal{O}(\log n)$-multiplicative approximation to S-SP (called k-source shortest paths in [25]) can be computed in $\mathcal{O}(|S| \cdot D \cdot \log n)$ time using $\mathcal{O}(|E| \cdot (\min(D, \log n) + |S|) + |S| \cdot n \log n)$ messages in an unweighted graph. In contrast to this we can compute exact S-SP faster by using less messages, that is in time $\mathcal{O}(|S| + D)$ with $\mathcal{O}((|S| + D) \cdot |E|)$ messages.

Based on these results, lower and upper bounds for computing exact and approximate solutions of a variety of other problems can be derived, as listed in Table 1 . Note that our algorithm demonstrates how to compute s BFS trees from s nodes in just $\mathcal{O}(s + D)$ time, which is of independent interest.

3.3 Diameter

Based on the APSP algorithm we derive an algorithm that can compute the diameter in linear time. As for APSP, this is optimal due to a lower bound stated in Theorem 5.1. of [22]. Since the authors of [22] also showed an $\Omega(\sqrt{n}/B + D)$-lower bound for any $(\times, 3/2 - \varepsilon)$-approximation it would be nice to obtain a matching upper bound. One approach towards this end is to consider a combinatorial $(\times, 3/2)$-approximation in a sequential setting by Aingworth, Chekuri, Indyk and Motwani [2]. According to the runtime of $\mathcal{O}(m \cdot \sqrt{n \cdot \log n} + n^2 \cdot \log n)$ it seems possible to implement it in our distributed model in time $\mathcal{O}(\sqrt{n} + D)$. As a first crucial step towards this approximation, [2] shows how to distinguish graphs of diameter 2 from graphs of diameter 4 in $\mathcal{O}(m \cdot \sqrt{n \cdot \log n} + n^2 \cdot \log n)$. This is a key insight that leads to their fast approximation-algorithm. They specifically mention that a step towards fast exact algorithms is being able to distinguish graphs of diameter 2 from graphs of diameter 3 in $o(n \cdot m)$ (instead of distinguishing 2 from 4 as needed for their approximation). Following this approach we show that a distributed algorithm can distinguish graphs of diameter 2 from graphs of diameter 4 in time $\mathcal{O}(\sqrt{n \cdot \log n})$, which is optimal. By extending the argument used in the proof of Theorem 5.1. in [22] we show that in contrast to this, distinguishing diameter 2 from 3 takes $\Omega(n/\log n)$ time by refining the construction of [22].

Although we were not able to transfer this $(\times, 3/2)$-approximation in such a way that it would result in distributed time $\mathcal{O}(\sqrt{n} + D)$, we are able to obtain an algorithm that yields a much better approximation factor. However, as expected, this is done by trading runtime for accuracy such that we use more than $\Omega(\sqrt{n}/\log n + D)$ time in most cases. Based on the S-SP-algorithm, we obtain a $(\times, 1 + \varepsilon)$-approximation to the diameter in time $\mathcal{O}(n/D + D)$. This result is complemented by an $\Omega(n/(D \cdot \log n) + D)$ lower bound for computing a $(+, 1)$-approximation in Theorem 4. Observe that if $D \geq \sqrt{n}$, we are unable to improve the runtime of the $(\times, 1 + \varepsilon)$-approximation, even when considering a worse approximation factor. A reason for this is that the upper bound of the algorithm matches the lower bound for $(\times, 3/2 - \varepsilon)$-approximations for these parameters. For more recent results, see Section 3.6.

3.4 Girth

In the sequential setting various results to approximate the girth are known, e.g. [5, 24, 34, 35, 36, 49]. In a sequentially setting they run faster than e.g. a $(+, 1)$-approximation that takes $\mathcal{O}(n^2)$ time [24]. When trying to transfer these algorithms into a distributed setting one has to compute a partial BFS tree of a certain depth (i.e., depth $k - 1$ if the girth is $2 \cdot k - 1$) for each of the n nodes. We show that in general computing all partial BFS trees of a certain depth might be hard by constructing a family of graphs of girth 3 where computing all 2-BFS trees takes $\Omega(n/\log n)$ time.

In a model where all n processes are connected to all other processes which want to verifying whether a subgraph contains a cycle of length d, a deterministic distributed algorithm running in time $\mathcal{O}(n^{1-2/d}/\log n)$ is stated in [16]. In contrast to this model, our processes can only communicate by using edges in the graph on which we want to compute the girth. We show how to compute the girth in this model in time $O(n)$ and extend this to a $(\times, 1 + \varepsilon)$- and a $(\times, 2 - 1/g)$-approximation with better runtime.

For more recent results, see Section 3.6.

3.5 Further Problems

We extend the results from above to the problems of computing the eccentricity, radius, center and peripheral vertices of a graph. Computing the center of a graph turned out to be important in applications such as PageRank [20, 31] and the analysis of social networks (centers will be e.g. celebrities [8]) while in spam-detectors it is proven to be useful to investigate peripheral vertices [48]. These settings are predestined to be solved by large distributed systems: The data processed is huge and exact sequential algorithms (or good approximations) for these problems usually have superquadratic runtime. Computing the eccentricity and radius are strongly

Problem	exact	$(+,1)$	$(\times,1+\varepsilon)$	$(\times,3/2-\varepsilon)$	$(\times,3/2)$	$(\times,2)$
APSP	$\tilde{\Theta}(n)^{16)}$	$\tilde{\Theta}(n)^{1,13)}$	$\tilde{\Theta}(n)^{1,13)}$	$\tilde{\Theta}(n)^{1,13)}$	–	–
eccentricity	$\tilde{\Theta}(n)^{5,11)}$	$\Omega(\frac{n}{D\cdot\log n}+D)^{11)}$	$\mathcal{O}(\frac{n}{D}+D)^{3)}$	$\Omega\left(\frac{\sqrt{n}}{\log n}+D\right)^{11)}$	–	$\Theta(D)^{18)}$
diameter	$\tilde{\Theta}(n)^{6,20)}$	$\Omega(\frac{n}{D\cdot\log n}+D)^{2)}$	$\mathcal{O}(\frac{n}{D}+D)^{17)}$	$\Omega\left(\frac{\sqrt{n}}{\log n}+D\right)^{21)*}$	$\mathcal{O}(n^{3/4}+D)^{14)}$	$\Theta(D)^{18)}$
radius	$\mathcal{O}(n)^{8)}$	–	$\mathcal{O}(\frac{n}{D}+D)^{17)}$	–	–	$\Theta(D)^{18)}$
center	$\tilde{\Theta}(n)^{9,12)}$	$\Omega(\frac{n}{D\cdot\log n}+D)^{12)}$	$\mathcal{O}(\frac{n}{D}+D)^{17)}$	$\Omega\left(\frac{\sqrt{n}}{\log n}+D\right)^{12)}$	–	$0^{19)}$
p. vertices	$\tilde{\Theta}(n)^{10,11)}$	$\Omega(\frac{n}{D\cdot\log n}+D)^{11)}$	$\mathcal{O}(\frac{n}{D}+D)^{17)}$	$\Omega\left(\frac{\sqrt{n}}{\log n}+D\right)^{11)}$	–	$0^{19)}$
girth	$\mathcal{O}(n)^{7)}$	–	$\mathcal{O}\left(\min\left\{n/g+D\cdot\log\frac{D}{g}\,,\,n\right\}\right)^{4)}$		–	–

10) Lem. 6
11) Lem. 10
12) Lem. 11
13) Lem.16
14) Cor. 1
15) Cor. 2
16) Cor. 3
17) Cor. 4
18) Rem. 1
19) Rem. 2

For the girth, two additional ratios are of interest:

Problem	$(\times,2-\varepsilon)$	$(\times,2-1/g)$
girth	$\Omega\left(\frac{\sqrt{n}}{\log n}+D\right)^{22)*}$	$\mathcal{O}\left(n^{2/3}+D\cdot\log\frac{D}{g}\right)^{15)}$

1) Thm. 1
2) Thm. 4
3) Thm. 6
4) Thm. 7
5) Lem. 2
6) Lem. 3
7) Lem. 7
8) Lem. 4
9) Lem. 5
20) [22]–Thm. 5.1
21) [22]–Thm. 6.1
22) [22]–Thm. 7.1

Table 1: The two tables above summarize the results of this paper and show which parts remain open. All entries are annotated with a number that is associated to the according Theorem/Lemma/Corollary in the list next to the tables. Entries marked with an asterisk (*) were previously known. Some fields are marked by "–" or do not appear. This indicates open problems: where 1) no almost tight bounds are known 2) only trivial bounds such as $\Omega(D)$ are known, or 3) no better upper/lower bounds than those stated for stronger/weaker approximation ratios are known. All entries in the tables reflect a choice for bandwidth B of $B=\log n$. We denote by $\tilde{\Theta}$ that according upper and lower bounds differ by at most a factor of polylog n.

related to these two problems. Table 1 summarizes the results obtained in this paper.

Remark 1. *As in the case of approximating the diameter, a* $(\times, 2)$*-approximation to the radius/eccentricity of all nodes can be computed by taking the eccentricity of any node. This can be done in* $\mathcal{O}(D)$ *by performing a breadth-first search rooted in this node.*

Remark 2. *Due to Fact 1, a* $(\times, 2)$*-approximation to the center/peripheral vertices is just the set of all nodes. Each node can decide to join the set internally thus the runtime would be* 0.

3.6 Combination with Independent Results by Peleg, Roditty and Tal

Independently, a similar algorithm to compute APSP and Diameter in time $\mathcal{O}(n)$ appears at ICALP 2012 [33]. In addition [33] demonstrates how to implement the sequential $(\times, 3/2)$-approximation algorithm mentioned in Section 3.3 in a distributed way in time $\mathcal{O}(D \cdot \sqrt{n})$. By combining this with our Corollary 4 (choosing $\varepsilon \leq 1/2$), we obtain:

Corollary 1. *Combining both algorithms yields a* $(\times, 3/2)$*-approximation to the diameter with runtime* $\mathcal{O}(\min\{D \cdot \sqrt{n}, n/D + D\})$*, which is* $\mathcal{O}(n^{3/4} + D)$.

Furthermore, [33] provides a $(\times, 2 - 1/g)$-approximation for the girth running in time $\tilde{\mathcal{O}}(D + \sqrt{gn})$. By combining this with Theorem 7 of our paper, (choosing $\varepsilon \leq 1/2$), we obtain:

Corollary 2. *When combining both algorithms, one can compute a* $(\times, 2 - 1/g)$*-approximation to the in time*

$$\mathcal{O}\left(\min\left\{D + \sqrt{gn}\,,\, \min\left\{n/g + D \cdot \log\frac{D}{g}\,,\, n\right\}\right\}\right),$$

which is $\mathcal{O}\left(n^{2/3} + D \cdot \log\frac{D}{g}\right)$.

4 All Pairs Shortest Paths

4.1 An Almost Optimal Algorithm

In this section we present a simple algorithm with a simple analysis that allows us to compute APSP of the underlying network in the message passing model with limited bandwidth $B = O(\log n)$ in time $\mathcal{O}(n)$. We argue that this algorithm can be used to compute solutions to several other properties of the graph in linear time as well. Combined with the $\Omega(n/\log n)$ lower bound [22] for computing the diameter, the presented algorithm is asymptotically nearly optimal (see Corollary 3.) We start with some notation.

Definition 8. *(Tree T_v) Given a node v, we denote the spanning tree of G that results from performing a breadth-first search BFS_v starting at v by T_v.*

Remark 3. *A spanning tree of G can be traversed in time $\mathcal{O}(n)$ by sending a pebble over an edge in each time slot. This can be done using e.g. a depth-first search.*

Algorithm 1 below computes shortest paths between all pairs of nodes in a graph. Given a graph G, it computes BFS tree T_1 (Line 1). Then it sends a pebble P to traverse tree T_1 (Lines 2–8). Each time pebble P enters a node v for the first time, P waits one time slot (Line 5), and then starts a breadth-first search (BFS) – using edges in G – from v with the aim of computing the distances from v to all other nodes (Line 6). Since we start a BFS from every node, each node learns its distance to all other nodes (that is APSP).

Algorithm 1 as executed by each node $v \in G$ simultaneously. Computes: APSP on G

```
1: compute $T_1$
2: send a pebble $P$ to traverse $T_1$
3: while $P$ traverses $T_1$ do
4:     if $P$ visits a node $v$ for the first time then
5:         wait one time slot //** avoid congestion
6:         start a BFS$_v$ from node $v$
                 //** compute all distances to $v$
7:     end if
8: end while
```

Remark 4. *For simplicity of the write up of the algorithm and proofs we do not state actual computations of distances. Algorithm 1 could be easily modified to compute these: During each computation of a BFS_v, tell each node u its depth in T_v. The depth is equivalent to the distance $d(u, v)$. In the end all distances are known. Shortest paths are implicitly stored via BFS trees.*

Lemma 1. *In Algorithm 1, at no time a node w is simultaneously active for both BFS_u and BFS_v.*

Proof. Assume a BFS$_u$ is started at time t_u at node u. Then node w will be involved in BFS$_u$ at time $t_u + d(u, w)$. Now, consider a node v whose BFS$_v$ is started at time $t_v > t_u$. According to Algorithm 1 this implies that the pebble visits v after u and took some time to travel from u to v. In particular, the time to get from u to v is at least $d(u, v)$, in addition at least node v is visited for the first time (which involves waiting at least one time slot), and we have $t_v \geq t_u + d(u, v) + 1$. Using this and the triangle inequality, we get that node w is involved in BFS_v strictly after being involved in BFS_u since $t_v + d(v,w) \geq (t_u + d(u,v) + 1) + d(v,w) \geq t_u + d(u,w) + 1 > t_u + d(u,w)$. $\square$

Theorem 1. *Algorithm 1 computes APSP in time $O(n)$.*

Proof. Since the previous lemma holds for any pair of vertices, no two BFS "interfere" with each other, i.e. all messages can be sent on time without congestion. Hence, all BFS stop at most D time slots after they were started. We conclude that the runtime of the algorithm is determined by the time $O(D)$ we need to build tree T_1, plus the time $O(n)$ that P needs to traverse tree T_1, plus the time $O(D)$ needed by the last BFS that P initiated. Since $D \leq n$, this is all in $O(n)$. $\square$

4.2 Applications

Given a solution for APSP, many other graph properties can be computed efficiently. The following lemmas and corollaries demonstrate several of these extensions.

Lemma 2. *The eccentricity of all nodes can be computed in $\mathcal{O}(n)$.*

Proof. Compute APSP in $\mathcal{O}(n)$. Based on this, each node v of the network computes its eccentricity internally by taking the maximum of all distances to v. The total complexity remains $\mathcal{O}(n)$. $\square$

Lemma 3. *The complexity of computing the diameter is $\mathcal{O}(n)$.*

Proof. Compute APSP in $\mathcal{O}(n)$ and aggregate the maximum of all distances using T_1 in additional time $\mathcal{O}(D)$. The result is the diameter. $\square$

Corollary 3. *Algorithm 1 is optimal up to a logarithmic factor due to Lemma 3 and Theorem 5.1 of [22].*

Lemma 4. *The complexity of comp. the radius is $\mathcal{O}(n)$.*

Proof. Compute all eccentricities in $\mathcal{O}(n)$ and aggregate the minimum of all eccentricities using T_1 in additional time $\mathcal{O}(D)$. The result is the radius. $\square$

Lemma 5. *The complexity of comp. the center is $\mathcal{O}(n)$.*

Proof. Compute all eccentricities and the diameter in $\mathcal{O}(n)$. Each node checks internally if the radius equals its eccentricity. If yes, it is a center vertex of the graph. $\square$

Lemma 6. *The complexity of computing peripheral vertices is $\mathcal{O}(n)$.*

Proof. Compute all eccentricities and the diameter in $\mathcal{O}(n)$. Each node checks internally if the diameter equals its eccentricity. If yes, it is a peripheral vertex of the graph. $\square$

Lemma 7. *The complexity of comp. the girth is $\mathcal{O}(n)$.*

Claim 1. *Executing BFS_1 can be used to check whether G is a tree or not.*

Proof. Consider the following implementation of BFS: Node 1 starts by sending an arbitrary message to all its neighbors (each neighbor receives the same message). Consider a node v that receives this message for the first time in time slot t_v. Then v forwards this message in time slot $t_v + 1$ to all its neighbors from which v did not receive a message in time slot t_v. In all other time slots, node v remains silent. Then G is a tree if and only if no node received more than one message during the execution of BFS_1 and this can be verified in time $\mathcal{O}(D)$. □

Proof. (of Lemma 7.) First, use Claim 1 to check in $\mathcal{O}(D)$ whether G is a tree or not. If yes, return ∞. If not, adopting a classical algorithm to compute the girth g: First, perform a BFS from each node (which is essentially done by Algorithm 1 in time $\mathcal{O}(n)$.) If during round t of a BFS_v, a vertex u that is already in T_v (or is included into T_v in round t) receives a second message in round t, we know that u and w belong to a cycle. If u is at depth d_u in T_v and receives a message from node w that is at depth d_w in T_v, then there is a cycle in G of length at most $d_u + d_w + 1$. In case v is the least common ancestor of nodes u and w in T_v, the cycle is exactly of size $d_u + d_w + 1$. If C is a minimal cycle in G – that is C defines the girth – the algorithm definitely detects C while performing a BFS from any node in C. At the same time the algorithm can never claim to have found a smaller cycle that does not exist. The overhead induced by this computation is only internal, min-aggregating at node 1 the size of the smallest cycle that any node is contained in takes time $\mathcal{O}(D)$. The total complexity of computing the girth is $\mathcal{O}(n)$. □

To be consistent with Definition 6, we could add in each corollary: Broadcasting the computed information to the whole network would take additional time at most $\mathcal{O}(n)$.

5 Lower Bounds

In [22] we already gave lower bounds for computing/approximating the diameter as well as lower bounds for approximating the girth. We make use of their general technique of transferring lower bounds from communication complexity into a distributed setting, but use different graph-constructions and arguments than in [22]. In the following subsection we review the notation used in the framework of [22].

5.1 Preliminaries

We need to introduce some notation: To obtain our lower bounds we need knowledge on basics of communication complexity, introduced by Yao [47]. Here, two computationally unbounded parties Alice and Bob each receive a k-bit string $a \in \{0,1\}^k$ and $b \in \{0,1\}^k$ respectively. Alice and Bob can communicate with each other one bit at a time and want to evaluate a function $h : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$ on their input. We assume that Alice and Bob have access to public randomness for their computation and we are interested in the number of bits that Alice and Bob need to exchange in order to compute h.

Definition 9. *(communication complexity). Let $\mathcal{A}_\varepsilon$ be the set of two-party algorithms that use public randomness (denoted by pub) and when used by Alice and Bob, compute h on any input a and b with an error probability smaller than ε. Denote by $R_\varepsilon^{cc-pub}(A(a,b))$ the communication complexity (denoted by cc) representing the number of 1-bit messages exchanged by Alice and Bob while executing an algorithm $A \in \mathcal{A}_\varepsilon$ to compute $h(a,b)$. We define*

$$R_\varepsilon^{cc-pub}(g) = \min_{A \in \mathcal{A}_\varepsilon} \max_{a,b \in \{0,1\}^k} R^{cc-pub}(A(a,b))$$

to be the smallest amount of bits any algorithm would need to send in order to compute h.

A well studied problem in communication complexity is that of set disjointness, where we are given two subsets of $[k-1]$ and need to decide whether they are disjoint. Here, the strings a and b indicate membership of elements to each of these sets.

Definition 10. *(disjointness problem). The set disjointness function* $\mathrm{disj_k} : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$ *is defined as follows.*

$$\mathrm{disj_k}(a,b) = \begin{cases} 0 & \text{if there is an } i \in [k-1] \text{ such that} \\ & \qquad a(i) = b(i) = 1 \\ 1 & \text{otherwise} \end{cases}$$

where $a(i)$ and $b(i)$ are the i-th bit of a and b respectively (indicating whether an element is a member of the corresponding set.)

We use the following basic theorem that was proven in Example 3.22 in [26].

Theorem 2. *For any sufficiently small $\varepsilon > 0$ we can bound $R_\varepsilon^{cc-pub}(\mathrm{disj_k})$ by $\Omega(k)$.*

Now that we have reviewed a basic result from communication complexity we use the notation of a cut:

Definition 11. *(cut). Let $G = (V, E)$ be a graph. A cut (G_a, G_b, C) is a partition of G into two disjoint subgraphs $G_a = (V_a, E_a)$ and $G_b = (V_b, E_b)$ and a cut-set $C \subseteq E$ s.t. $V = V_a \dot\cup V_b$ and $E = E_a \dot\cup E_b \dot\cup C$, where $\dot\cup$ denotes the disjoint union of sets. The cut-set C consists of $c := |C|$ edges whose endpoints are in different subsets of the partition.*

Observe that given a function f and a graph G in a distributed setting, as well as a cut with cut (G_a, G_b, C) of G, we can define a two-party communication problem f' according to the graph-problem f in a canonical way: We define

$$f'((G_a, C), (G_b, C)) := f(G)$$

Lemma 4.1. of [22] states that f' can be reduced to f. Where Alice gets input (G_a, C) and Bob gets input (G_b, C).

Definition 12. *Given a cut-set C_k (that depends on k, but not on inputs a and b), a c_k-reduction*

$$\begin{aligned}\mathcal{R} : \{Alice, Bob\} \times \{0,1\}^k \to \{(H, C_k) : G \text{ is any graph}\\ \text{and } H \text{ is any subgraph of}\\ G \text{ such that } (H, G \setminus H, C_k) \text{ is}\\ \text{a cut of } G \text{ with } |C_k| = c_K\}\end{aligned}$$

is a function that transforms any h-inputs a, b into inputs for f' such that

$$g(a, b) = f'(\mathcal{R}(Alice, a), \mathcal{R}(Bob, b)).$$

Observe that the size of C_k does not depend on a nor b.

Theorem 3. *(Version of Theorem 4.1. of [22]). Let $B \geq 1$. If a function h can be reduced to f' using a c_k-reduction. We can bound*

$$\frac{R_\varepsilon^{cc-pub}(g)}{2 \cdot c_k \cdot B} \leq R_\varepsilon^{dc}(f).$$

We finish this review by defining a map that is used during each of our lower bounds.

Definition 13. *Denote by m a map that maps f'-inputs $((G_a, C_k), (G_b, C_k))$ to the graph $G_{a,b}$ that corresponds to the cut, that is $G_{a,b} := (V_{a,b}, E_{a,b})$, s.t. $V_{a,b} := V_a \cup V_b$ and $E_{a,b} := E_a \cup E_b \cup C_k$.*

5.2 A $(+,1)$-Approximation Lower Bound

Theorem 4. *For any $\delta > 0$, parameter $d \geq 4$, where d is even, and $n \geq d + 6$ and $B \geq 1$ and sufficiently small ε, any distributed ε-error algorithm A that computes a $(+,1)$-approximation to the diameter requires at least $\Omega\left(\frac{n}{D \cdot B} + D\right)$ time for some n-node graph of diameter $D \in \{d, d+2\}$. This can be extended to odd d.*

Proof. In order to prove our lower bounds, we introduce the constant p that later defines the length of a path and in this prove we set it to be $p := d/2 - 1$ (we assumed d to be even). During the reduction we construct graphs $G_{a,b}$ of diameter $diam(G_{a,b}) \in \{d, d+2\}$ depending on inputs a and b such that any $(+,1)$-approximation algorithm for the diameter would estimate $diam(G_{a,b})$ to be strictly less than $2p + 4 = d + 2$ if a and b are disjoint. If a and b are not disjoint, the diameter (and thus the estimate) would always be at least $d+2$. We prove both statements later in a formal way. Since the reduction $\mathcal{R}$ delivers the above promise-problem[2] we can just use the function $diam_{d+2}$ that decides whether $diam(G) < d + 2$ or not as the decision-version of $(+,1)$-approximating the diameter.

$$diam_{d+2}(G) := \begin{cases} 1 & : diam(G) < d+2 \\ 0 & : \text{else} \end{cases}$$

We use the technique described in [22] in order to prove Theorem 4: We derive a function $diam'_{d+2}$ from $diam_{d+2}$ as described in Section 5.1. To prove lower bounds depending on n, we choose the length k of the inputs a and b to the base-function h to be the function $k(n)^2$ depending on n. We set $k(n) := \left\lfloor \frac{n-4}{2\cdot(p+2)} \right\rfloor$ (here we need $n \geq d+6$) as provided in the statement of Theorem 4). Later the graph will have a subgraph consisting of $\Theta(k(n))$ nodes and $\mathcal{O}(k(n)^2)$ edges that will encode $2k(n)^2$ bits of a and b. Using the framework of [22] sketched in the previous section, we consider the $\mathrm{disj}_{\mathrm{k(n)}^2}$ problem to be the base-function h. Now we need to define a reduction $\mathcal{R}$ that given inputs a and b to h, maps $(Alice, a)$ and (Bob, b) to inputs $(G_a, C_{k(n)^2})$ and $(G_b, C_{k(n)^2})$ for $diam'_{d+2}$. During the reduction $\mathcal{R}$, Alice defines L and L', Bob defines R and R' to be the following sets of nodes (as displayed in Figure 1):

$$\begin{aligned} L &= \{l_\nu | \nu \in [2k(n)-1]\} & L' &= \{l'_\nu | \nu \in [k(n)-1]\} \\ R &= \{r_\nu | \nu \in [2k(n)-1]\} & R' &= \{r'_\nu | \nu \in \{k(n), \ldots, 2k(n)-1\}\} \end{aligned}$$

Given inputs $a \in \{0,1\}^{k(n)^2}$ and $b \in \{0,1\}^{k(n)^2}$ Alice constructs G_a and Bob G_b. For each $\nu \in [k(n)-1]$ Alice adds a path T^a_ν of length p connecting nodes l_ν to l'_ν as depicted in Figure 1. Furthermore for each $\nu \in [k(n)-1]$ and $\mu \in \{k(n), \ldots, 2k(n)-1\}$ Alice adds an edge (l_ν, l_μ) connecting l_ν to l_μ iff $a\,(k(n)\cdot(\mu - k(n) + 1) + \nu) = 0$. In addition we add nodes $\{c_0, c_1\}$ and connect them by edges $\{(l_\nu, c_0) : \nu \in [k(n)-1]\}$ and $\{(l_\nu, c_1) : \nu \in \{k(n), \ldots, 2k(n)-1\}\}$ as well as edge (c_0, c_1). The graph G_b is constructed by Bob in almost the same way using paths T^b_ν and edges that are added depending on b. Bob also adds nodes c_2 and c_3 and corresponding edges. The main difference is that paths T^b_ν are added for each $\nu \in \{k(n), \ldots, 2k(n)-1\}$ instead of $\nu \in [k(n)-1]$ and edges (r_ν, r_μ) if $b\,(k(n)\cdot(\mu - k(n) + 1) + \nu) = 0$.

[2]In a promise problem the input is promised to belong to a certain subset of all possible inputs. In this case this subset is defined by the diameter.

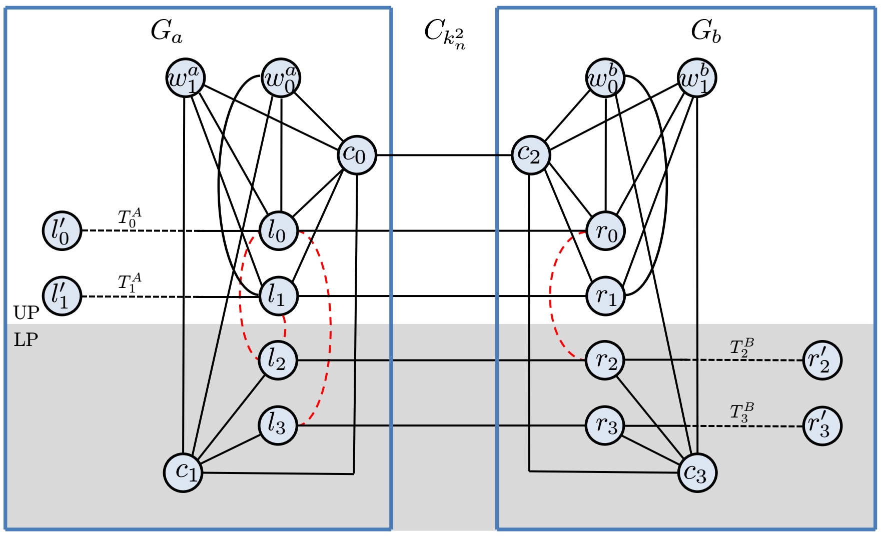

Figure 1: Graph used to calculate $\mathrm{disj}_{\mathrm{k(n)}^2}$ when given a diameter estimator algorithm. This graph has $n = 20$ and $d = 4$, thus $k(n) = 2$ and $p = 1$. Displayed is $G_{a,b}$ for strings $a = (0, 0, 0, 1)$ and $b = (0, 1, 1, 1)$. The red dashed edge (l_0, l_2) represents $a(0) = 0$, edge (l_0, l_3) represents $a(1) = 0$, edge (l_1, l_2) represents $a(2) = 0$, edge (r_0, r_2) represents $b(0) = 0$. The line connecting l_0 to l_0' as well as similar lines are partly dashed to indicate that these are paths, not edges. Since the sets are not disjoint the diameter is $d + 2$ as $d(l_1', r_3') = d + 2$.

Now we set the cut-set $C_{k(n)^2}$ that connects G_a with G_b to be $C_{k(n)^2} := \cup_{i\in[2k(n)-1]}\{(l_i, r_i)\} \cup \{(c_0, c_2)\}$. Thus $\mathcal{R}$ is a $(2k(n) + 1)$-reduction. At this stage, we cannot define $G_{a,b} := m((G_a, C_{k(n)^2}), (G_b, C_{k(n)^2}))$ (see definition 13), since the graph does not necessarily have n nodes as each of V_a and V_b might be smaller than $n/2$. E.g. for V_a we know:

$$\begin{aligned}|V_a| &= |L| + |L'| + \#\text{nodes in paths } \{T_\mu^A\}_{\mu=0}^{k(n)-1} + |\{c_0, c_1\}|\\ &= 2 \cdot k(n) + k(n) + k(n) \cdot (p - 1) + 2\\ &\le k(n) \cdot (p + 2) + 2\end{aligned}$$

Using the definition of $k(n)$ we can rewrite this to be $\left\lfloor \frac{n-4}{2\cdot(p+2)} \right\rfloor \cdot (p+2)+2 \le \frac{n-4}{2\cdot(p+2)} \cdot (p + 2) + 2 = n/2$. We can show $|V_b| \le n/2$ in a similar way. However, we want our lower bound to be valid for all graph-sizes n and thus need to fill up the graph with nodes until there are n nodes in total. Therefore we add as many fill-up nodes $\{w_\nu^a\}$ to G_a (each connected by edges to all the nodes in $\{l_0, \ldots, l_{k(n)-1}, c_0, c_1\}$) such that $|V_a| = n/2$ and as many fill-up nodes $\{w_\nu^b\}$ to G_b (each connected by edges to all the nodes in

$\{r_0, \ldots, r_{k(n)-1}, c_2, c_3\})$ such that $|V_b| = n/2$. To be more precise, we add $n/2 - \left\lfloor \frac{n-4}{2\cdot(p+2)} \right\rfloor (p+2) - 2$ fill-up nodes to each of the two graphs. Finally we set $G_{a,b} := m((G_a, C_{k(n)^2}), (G_b, C_{k(n)^2}))$ using Definition 13. Observe that $(G_a, C_{k(n)^2})$ can be computed from a without knowing b and $(G_b, C_{k(n)^2})$ can be computed from b without knowing a, thus the reduction $\mathcal{R}$ has the desired properties.

On our way towards Theorem 4 we consider two parts of the constructed graphs $G_{a,b}$:

- Upper part **UP** (with white background in Figure 1): these are the nodes in $\{l_i : i \in [k(n)-1]\} \cup \{l'_i : i \in [k(n)-1]\} \cup \{r_i : i \in [k(n)-1]\} \cup \{c_0, c_2\}$ as well as the nodes contained in the paths T^A_ν for $\nu \in [k(n)-1]$ and the fill-up nodes w^a_μ and w^b_μ.

- Lower part **LP** (with gray background in Figure 1): these are the nodes in $\{l_i : i \in \{k(n), \ldots, 2\cdot k(n)-1\} \cup \{r_i : i \in \{k(n), \ldots, 2\cdot k(n)-1\}\} \cup \{r'_i : i \in \{k(n), \ldots, 2\cdot k(n)-1\}\} \cup \{c_1, c_3\}$ as well as the nodes contained in the paths T^B_ν for $\nu \in \{k(n), \ldots, 2\cdot k(n)-1\}$.

Lemma 8. *If a and b are not disjoint, the diameter of the graph $G_{a,b}$ is $d+2$.*

Proof. If a and b are not disjoint there exists at least one i such that $a(i) = b(i) = 1$. Lets fix such an i. We show that the two nodes $l'_{i \bmod k(n)}$ and $r'_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}$ have distance strictly greater than $d+1$. First observe that any path that connects them includes paths $\Gamma^A_{i \bmod k(n)}, \Gamma^B_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}$ and nodes $l_{i \bmod k(n)}$ and $r_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}$. Thus $d(l'_{i \bmod k(n)}, r'_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}) = d(l_{i \bmod k(n)}, r_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}) + 2p$. Now we argue that nodes $l_{i \bmod k(n)}$ and $r_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}$ must have distance of at least 4. The reason for this is that $a(i) = b(i) = 1$ implies that there is neither a direct edge between $l_{i \bmod k(n)}$ and $l_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}$ nor between $r_{i \bmod k(n)}$ and $r_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}$. In each of the two cases one needs to make a detour visiting two additional nodes in $L \cup \{c_0, c_1\}$ (or $R \cup \{c_2, c_3\}$ respectively). Due to the construction of $G_{a,b}$ this takes 3 edges instead of one. To finally extend this to a path from $l_{i \bmod k(n)}$ to $r_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}$ one needs to cross the cut using either edge $(l_{i \bmod k(n)}, r_{i \bmod k(n)})$ or $(r_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}, l_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor})$ and thus obtains a path of length four. Thus $d(l'_{i \bmod k(n)}, r'_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}) = 2\cdot p + 4 = d+2$. This implies that the diameter is at least $d+2$. Now we prove, that the diameter is at most $d+2$. For this we observe (case 1) that any two nodes $u, v \in G_a$ have distance of at most $p+3 \le d+2$ via c_0 (since $p \ge 1$). (Case 2) any two nodes $u, v \in G_b$ have distance of at most $p+3 \leq d+2$ via c_2. Furthermore we notice that

Claim 2. *If $d \geq 4$, each of the nodes in $\{c_0, c_1, c_2, c_3, w_0^a, w_1^a, \dots, w_0^b, w_1^b, \dots\}$ has distance of at most d to any other node in $G_{a,b}$.*

Proof. Notice that this is certainly true for c_0, c_1, c_2, c_3 since they have distance of at most $p+3 = d/2+2$ to any other node in $G_{a,b}$ which is less than d since we assume $d \geq 4$ in the statement of this Claim (and Theorem 4).

Even more precisely: Notice that nodes c_0 and c_2 have distance of at most $\max\{2, p+2\} = p+2$ to any other node in **UP**. While nodes c_1 and c_3 have distance of at most $\max\{2, p+1\} = p+1$ to any other node in **LP**. Nodes $w_0^a, w_1^a, \dots$ have distance 1 to c_0 as well as distance 1 to c_1 and thus are at most $\max\{1 + d(c_0, v), 1 + d(c_1, v)\} \leq p+3$ hops away from any node v in $G_{a,b}$. Similarly, via c_2 and c_3 the nodes $w_0^a, w_1^a, \dots$ can reach any node in $G_{a,b}$ using $p+3$ hops. Since we assume $d \geq 4$ in the statement of this Claim we conclude that $p+3 = d/2+2 \leq d$. □

Using this claim we only need to treat the remaining case in the version of $u \in L \cup L' \cup \{\text{nodes on paths } T_\nu^A\}$ and $v \in R \cup R' \cup \{\text{nodes on paths } T_\nu^B\}$. Now we observe that $d(u, c_0) \leq \max\{2, p+1\} \leq p+1$ (since $p \geq 1$) and $d(v, c_2) \leq \max\{2, p+2\} \leq p+2$. Thus in total $d(u, v) \leq d(u, c_0) + d(c_0, c_2) + d(c_2, v) \leq 2p+4 \leq d+2$. □

Lemma 9. *The sets a and b are disjoint, if and only if the diameter of $G_{a,b}$ is d.*

Proof. Consider nodes u and v that define the diameter, that is u and v are chosen such that $d(u, v)$ is maximal with respect to $G_{a,b}$. Now we distinguish three cases:

1. Assume $u, v \in$ **UP**: Due to Claim 2 we already know in case that u or v belong to the set $\{c_0, c_1, c_2, c_3, w_0^a, w_1^a, \dots, w_0^b, w_1^b, \dots\}$ that the distance $d(u, v)$ is d. In the other case there is a node $l_\nu \in L$ such that $d(u, l_\nu) \leq \max\{1, p\}$. Similarly there is a node $l_\mu \in L$ such that $d(v, l_\mu) \leq \max\{1, p\} = p$. Since both l_ν and l_μ are connected via c_0, we conclude that $d(u, v) \leq d(u, l_\nu) + d(l_\nu, l_\mu) + d(l_\mu, v) \leq 2p+2 \leq d$. Observe that this is independent of the inputs a and b.

2. Assume $u, v \in$ **LP**: The proof is similar to the one for "Assume $u, v \in$ **UP**" and results in $d(u, v) \leq d$ as well. This is independent of inputs a and b as well.

3. Assume $u \in$ **UP** and $v \in$ **LP**: Due to Claim 2 we only need to care about the case in which u or v to not belong to the set $\{c_0, c_1, c_2, c_3, w_0^a, w_1^a, \dots, w_0^b, w_1^b, \dots\}$. Using a similar argument as in

both previous cases we know that there are nodes $u' \in L \cap \mathbf{UP}$ and $v' \in R \cap \mathbf{LP}$ such that $d(u, u') \leq \max\{p, 1\} = p$ and $d(v, v') \leq p$. Like in the first case we again assume that $u' = l_\nu$ for some $i \in [k(n) - 1]$ and $v' = r_\mu$ for some $j \in \{k(n), \ldots, 2 \cdot k(n) - 1\}$. Due to a and b being disjoint, there is either the edge (l_ν, l_μ) or (r_ν, r_j) in graph $G_{a,b}$ for each ν and μ. Therefore $d(l_\nu, r_\mu) = 2$ via l_μ or r_ν. From this we conclude that $d(u, v) = d(u, l_\nu) + d(l_\nu, r_\mu) + d(r_\mu, v) \leq 2 \cdot p + 2 \leq d$

□

Now we relate the problem of deciding whether a and b are disjoint to the problem of $(+, 1)$-approximating the diameter of a graph.

Proof. (of Theorem 4). By Lemma 8 and 9 the estimate to the diameter produced by any $(+, 1)$-approximation algorithm is at least $d+2$ if and only if a and b are not disjoint else it is d or $d + 1$. These two lemmas also guarantee that the exact diameter D is either d or $d + 2$ as stated in the Theorem.

To solve the $\mathrm{disj}_{\mathrm{k(n)}^2}$ problem using any $(+, 1)$-approximation-algorithm for *diam* we use the reduction $\mathcal{R}$ from $\mathrm{disj}_{\mathrm{k(n)}^2}$ to $diam'_{d+2}$ and observed that $\mathcal{R}$ delivered a promise-problem such that there is a reduction from $\mathrm{disj}_{\mathrm{k(n)}^2}$ to $diam_{d+2}$ via $diam'_{d+2}$. According to Theorem 3, we know that

$$\frac{R_\varepsilon^{cc-pub}(\mathrm{disj}_{\mathrm{k(n)}^2})}{2 \cdot |C_{k(n)^2}| \cdot B} \leq R_\varepsilon^{dc}(diam_{d+2})$$

Due to Theorem 2 we know that $R_\varepsilon^{cc-pub}(\mathrm{disj}_{\mathrm{k(n)}^2})$ is at least $\Omega(k(n)^2)$. Together with the fact that $|C_{k(n)^2}| = 2 \cdot k(n) + 1$ we conclude that for all inputs to $\mathrm{disj}_{\mathrm{k(n)}^2}$ of size $k(n)^2$ we obtain $R_\varepsilon^{dc}(diam_{d+2}) \in \Omega(k(n)/B)$. We derive the stated result since we chose $k(n) := \left\lfloor \frac{n-4}{2(p+2)} \right\rfloor = \left\lfloor \frac{n-4}{2(d/2-1+2)} \right\rfloor \in \Theta(n/D)$. □

Remark 5. *The graphs $G_{a,b}$ in the above construction always have even diameter D as we assume d to be even. To obtain odd diameter, just replace the cut-edges by paths of length two.*

□

In the remainder of this section we extend this lower bound to several other problems. In the according statements, we implicitly assume similar conditions as stated in those Theorems/Lemmas used in the reductions.

Lemma 10. *The following problems: 1) computing APSP 2) computing the eccentricity of each node of a graph 3) finding a peripheral vertex, take $\Omega(n/B + D)$ time. Any $(\times, 3/2 - \varepsilon)$-approximation to the above problems takes $\Omega(\sqrt{n}/B + D)$ time. Any $(+, 1)$-approximation takes time $\Omega(n/(B \cdot D) + D)$.*

Proof. Solutions for APSP, eccentricity and peripheral-vertex can directly be used to obtain (an estimate of) the diameter in additional time $\mathcal{O}(D)$: In case of APSP and eccentricity we can do so by computing the maximum of the known distances or eccentricities by max-aggregation. In case we are given the (approximate) peripheral vertices, we just compute the eccentricity of any of them. These reductions yield that, if any of these tasks could be done faster than $o\left(\frac{n}{B}\right)$, $o\left(\frac{\sqrt{n}}{B}\right)$ or $o\left(\frac{n}{D \cdot B}\right)$ respectively, this is in contradiction to the already established lower bounds of Theorem 4 of this paper as well as Theorems 5.1. and 6.1. of [22]. □

Note that in Lemma 16 we provide a better lower bound for $(\times, 3/2 - \varepsilon)$-approximating APSP.

Lemma 11. *Computing the center of a graph takes $\Omega(n/B + D)$. Computing a $(\times, 3/2 - \varepsilon)$-approximation to the center takes $\Omega(\sqrt{n}/B + D)$ time. Computing a $(+, 1)$-approximation to the center takes $\Omega(\frac{n}{D \cdot B} + D)$ time.*

Proof. We need to have a look into the reductions in the proofs of the diameter lower bounds. In the proof of Theorem 8TODO: das kennt man hier noch nicht , the radius of the considered graphs is always 2 (as witnessed by node c_L), while the diameter can be 2 or 3. We notice that the diameter of a graph is 2 if and only if each vertex is in the center of the graph, otherwise the diameter is 3. Given all center vertices, we can decide by sum-aggregation in $\mathcal{O}(D)$ whether these are all nodes and based on the result decide whether the diameter is 2 or 3. Thus if we could solve these problems faster than $O(n/B)$, this part of the lower bound would be violated (observe that the $\Omega(D)$-part is the trivial lower bound).

For a similar reason we could distinguish between diameter d and $d + 2$ in the worst case graph provided in proof of Theorem 4 when given a $(+, 1)$-approximation for the center. According lower bounds follow as described above.

Similarly we can use Theorem 6.1 of [22], by using a center-approximation to distinguish between diameter $6 \cdot p_s$ and $4 \cdot p_s$, where p_s is defined in the proof of Theorem 6.1 in [22]. According lower bounds follow as described above. □

6 Approximation Algorithms

In the previous section we have seen lower bounds that demonstrated that obtaining $(+, 1)$-approximations takes $\Omega(n/D + D)$ time for the diameter. In order to approach this by an upper bound, we present an algorithm running in time $\mathcal{O}(n/D + D)$ that computes a $(\times, 1 + \varepsilon)$-approximation of the diameter. Furthermore we present a $(\times, 1 + \varepsilon)$-approximation to the girth g running in time $\mathcal{O}\left(n/g + D \cdot \log \frac{D}{g}\right)$. At the heart of these two approximations is the S-shortest paths problem (S-SP). In this problem we are given a graph and a subset S of its vertices. We are interested in computing the distances between all pairs of nodes in $S \times V$.

6.1 S - Shortest Paths

The idea of the algorithm is that we compute BFS trees T_v from each node $v \in S$. Differently from Algorithm 1, the trees start growing at the same time from each node $v \in S$. This causes that while growing T_v, the development of T_v might be delayed once reaching a node that is already part of a BFS tree T_u started in u if ID u is strictly smaller then ID v. We will prove that the total delay of any BFS is $\mathcal{O}(|S|)$ and that the resulting trees are indeed BFS trees. Clearly this is directly an alternative (more complicated, less elegant) algorithm/proof for APSP running in time $\mathcal{O}(n)$.

Algorithm 2 is executed by each node $v \in V$, the pseudocode demonstrates what a node v does. Each node v locally stores $d(v)$ sets L_i, one for each of the $d(v)$ neighbors $v_1, \ldots, v_{d(v)}$, and a set L. These locally stored sets depend on v and therefore the content of these sets might be different in different nodes during the execution of Algorithm 2.

At the beginning all these lists of a node v contain ID v if and only if $v \in S$, else they are empty (lines 1–6). Furthermore v maintains an array δ that will eventually store at position u (indicated by $\delta[u]$ the distance of v to node u. Initially $\delta[u]$ is set to infinity for all u and will only get updated at the time the distance is known (Line 21).

At time t, set L contains all node-IDs corresponding to the BFS tree computations that reached v until time t. That is at the end of the algorithm L contains all nodes of S.

At any time L_i contains all IDs that are currently in L except those that were forwarded successfully to neighbor v_i in the past. We say an ID l_i is forwarded successfully to neighbor u_i, if u_i is not sending a smaller ID r_i to v at the same time.

To compute the trees in Algorithm 2, the unique node with ID 1 computes $D' := ecc(1)$ and thus a $(\times, 2)$-approximation to the distance-diameter D'. This value is subsequently broadcasted to the network (lines 7–9). Then the computation of the $|S|$ trees starts and runs for $|S| + D'$ time steps.

Lines 13–17 make sure that at any time the smallest ID, that was not already successfully forwarded to neighbor u_i is sent. If a node ID r_i was received successful for the first time (verified in lines 19 and 20), we update $\delta[r_i]$, add r_i to the according lists (Line 22) and remember in variable $parent[r_i]$ who v's parent in T_{r_i} is (Line 23). In case a node-ID u is received several times, the algorithm adds the edge to tree T_u through which ID u was received at the earliest point in time. In case ID u was received at this (first) time from several neighbors, the algorithm (as we stated it) chooses the edge with lowest index i due to iterating in this way in Line 18. On

Algorithm 2 as executed by each node $v \in G$ simultaneously. Input: $S \subseteq V$
Computes: S-SP on G

```
    //** INITIALIZATION
 1: $L := \emptyset$; $\delta := \{\infty, \infty, \ldots, \infty\}$
 2: if $v \in S$ then
 3:     $L := \{v\}$
 4:     $\delta[v] := 0$
 5: end if
 6: $L_1, \ldots, L_{d(v)} := L$
 7: if  $u = 1$ then
 8:     compute $D' := 2 \cdot ecc(u)$ //** upper bound on $D$
 9:     broadcast $D'$
10: else
11:     wait until $D'$ was received
12: end if

    //** COMPUTATION of $S$-SP
13: for $|S| + D'$ time steps do
14:     for $i = 1, \ldots, d(v)$ do
15:         $l_i := \begin{cases} \infty & : \text{if } L_i = \emptyset \\ \min(L_i) & : \text{else} \end{cases}$
16:     end for
17:     within one time slot:
    —send $(l_1, \delta[l_1] + 1)$ to neighbor $v_1$, receive $(r_1, \delta_{r_1})$ from $v_1$
    —send $(l_2, \delta[l_2] + 1)$ to neighbor $v_2$, receive $(r_2, \delta_{r_2})$ from $v_2$
    —...
    —send $(l_{d(v)}, \delta[l_{d(v)}] + 1)$ to neighbor $v_{d(v)}$, receive
    —                         $(r_{d(v)}, \delta_{r_{d(v)}})$ from $v_{d(v)}$
18:     for $i = 1, \ldots, d(v)$ do
19:         if $r_i < l_i$ then
                //** $T_{l_i}$'s message is delayed due to $T_{r_i}$
20:             if $r_i \notin L$ then
                    //** first time received "$r_i$" successfully
21:                 $\delta[r_i] = \delta_{r_i}$   //** updates distances
22:                 $L := L \cup \{r_i\}$   //** updates lists
                    $L_1 := L_1 \cup \{r_i\}$
                    ...
                    $L_{i-1} := L_{i-1} \cup \{r_i\}$
                    $L_{i+1} := L_{i+1} \cup \{r_i\}$
                    ...
                    $L_{d(v)} := L_{d(v)} \cup \{r_i\}$
23:                 $parent[r_i] := v_i$
24:             end if
25:         else
26:             $L_i := L_i \setminus \{l_i\}$ //** "$l_i$" was successfully
27:         end if              //** sent to neighbor $i$.
28:     end for
29: end for
```

the other hand if we did not successfully receive a message from neighbor v_i but sent successfully a message to v_i, the transmitted ID is removed from L_i (Line 26).

Theorem 5. *Algorithm 2 computes S-SP, in time $\mathcal{O}(|S| + D)$.*

Proof. First we prove the correctness of Algorithm 2: Let us choose a node $u \in S$ and consider the computation T_u (for now ignoring that we actually want to compute S-SP.) In such a computation, at time t, nodes at distance t from u receive a message ID u from all neighbors that are at distance $t-1$ to u. An edge incident to the neighbor with lowest index that sent such a message is added to tree T_u.

Now consider Algorithm 2 and node v at distance t from u, as well as two nodes $w_1, w_2 \subseteq N_1(v)$; we can ignore the case that v has only one neighbor. A message containing ID u is sent over the edge (w_1, v) earlier than over edge (w_2, v) if and only if $d(u, w_1) < d(v, w_2)$. To see this, note that the set of lower IDs which delay the messages of T_u is the same for both paths (u, w_1, v) and (u, w_2, v). To see this assume that T_i is delaying the message ID u sent over (u, w_1, v) at some point. Then ID i will reach (or will have reached in case ID i is coming from v's direction) v earlier then ID u. Thus it will also block the message ID u running through path (u, w_2, v), if it did not already block it earlier.

Now we prove that Algorithm 2 runs in time $\mathcal{O}(|S| + D)$: The BFS executed by node 1 to compute D' takes $\mathcal{O}(D)$. The for-loop in Lines 13–31 is executed for $|S| + D'$ times, each time taking 1 round of communication, which is $\mathcal{O}(|S| + D)$.

Note that during traveling on any $u-v$-path of T_u, the message ID u gets delayed at most once by the computation BFS$_i$ if ID i is strictly smaller then ID u. This happens either by waiting in the set L_i of some node or by trying to cross an edge of the path at the same time as ID i (which will not be successful during the according time slot). Thus the total delay of computing T_u is $|S|$ and the total runtime of Algorithm 2 is $\mathcal{O}(|S| + D)$.

Finally observe, that in Line 21, after $\delta[r_i]$ is changed from ∞ to the value received, $\delta[r_i]$ stores the correct distance between v and node r_i. This can be shown by induction over the levels in the computed BFS tree rooted in the node with id r_i. $\square$

6.2 A $(\times, 1+\varepsilon)$-Approximation to Diameter and Girth

The presented algorithms are based on computing a k-dominating set $\mathcal{DOM} \subseteq V$ and solving $\mathcal{DOM}$-SP. There is plenty of literature on k-dominating sets. We use the results provided in [27].

Definition 14. *(composed from [27]) A k-dominating set for a graph G is a subset $\mathcal{DOM}$ of vertices with the property that for every $v \in V$ there is some*

$u \in \mathcal{DOM}$ at distance of at most k from v. For every such k-dominating set we define a partition $\mathcal{P} = \{P_1, \ldots, P_{|\mathcal{DOM}|}\}$ such that each node of V is exactly in one P_i and of distance less than or equal k to the dominator in $\mathcal{DOM}$ of P_i.

Lemma 12. *(Version of Lemma 2.3 in [27]). Algorithm `Diam_DOM` of [27] computes a k-dominating set $\mathcal{DOM}$ of size $|\mathcal{DOM}| \le \max\{1, \lfloor n/(k+1) \rfloor\}$ deterministically and its time complexity is $6 \cdot D + k$. A partition $\mathcal{P}$ can be computed in additional time $\mathcal{O}(k)$.*

Theorem 6. *We can compute a $(\times, 1+\varepsilon)$-approximation of all eccentricities in $\mathcal{O}(\frac{n}{D} + D)$ time.*

Proof. To do so, we use Fact 1 and determine a $(\times, 2)$-estimate $D' := 2 \cdot ecc(1)$ of the diameter by computing the eccentricity $ecc(1)$ of the node with ID 1. Next we set $k := \lfloor \varepsilon \cdot D'/4 \rfloor$ and use Lemma 12 to compute a k-dominating set $\mathcal{DOM}$ of size $|\mathcal{DOM}| \le \max\{1, \lfloor n/(k+1) \rfloor\}$ in time $\mathcal{O}(D+k)$. Then we solve $\mathcal{DOM}$-SP in time $\mathcal{O}(|\mathcal{DOM}| + D)$. At the end of this computation each node $v \in V$ knows its distance to all vertices in $\mathcal{DOM}$. Let $u \in \mathcal{DOM}$ be a node in $\mathcal{DOM}$ of maximal distance to v. Then $d(u, v)$ is at most k hops shorter than v's actual eccentricity due to the use of k-dominating sets. Thus the computed estimate of the eccentricity of v is less than $k + \max_{u \in \mathcal{DOM}} d(u, v) \le k + ecc(v) = \lfloor \varepsilon \cdot D'/4 \rfloor + ecc(v) = \lfloor \varepsilon \cdot ecc(1)/2 \rfloor + ecc(v) \le (1+\varepsilon) \cdot ecc(v)$ where the last bound follows due to Fact 1. The total time for this computation is $\mathcal{O}(|\mathcal{DOM}| + D + k) = \mathcal{O}(n/D + D)$. □

Corollary 4. *We can compute a $(\times, 1+\varepsilon)$-approximation of the diameter, radius, center and peripheral vertices in time $\mathcal{O}(n/D + D)$.*

Theorem 7. *We can compute a $(\times, 1+\varepsilon)$-approximation of the girth in time*

$$\mathcal{O}\left(\min\left\{\left(n/g + D \cdot \log \frac{D}{g}\right), n\right\}\right).$$

Proof. Since we cannot get a good estimate for g as easy as for the diameter, we start with a loose upper bound on g and iteratively run Subroutine *ImproveEstimate* to improve this bound. We continue calling *ImproveEstimate* until the updated estimate is a $(\times, 4)$-approximation of g. Calling *ImproveEstimate* a last time–using modified parameters–will yield a $(\times, 1+\varepsilon)$-approximation.

In more detail, in Line 2 of Algorithm 3 we execute a BFS starting at node 1 to figure out whether G is a tree. At the same time we compute $ecc(1)$. In case G is a tree (what we can check during the computation of BFS_1 using Claim 1), we set the girth-estimate to ∞. Else we define $g_1 := 4 \cdot ecc(1) + 2$. Due to Facts 2 and 1 it turns out that g_1 is the upper bound on the girth that was mentioned above (Line 2). In case G is not a tree, Algorithm 3 enters the while-loop in Line 3 and calls *ImproveEstimate* for estimate g_1 and obtains a better estimate g_2. Then we run *ImproveEstimate* again, this time on g_2 to obtain an even better estimate g_3 and so on (see details below). Algorithm 3 stops doing this when $g_j > \frac{3}{4} \cdot g_{j-1}$ and we denote this value j by $j_{\max}$

During the j^{th} execution of *ImproveEstimate*:

we start by choosing $k_j = \lfloor g_j/4 \rfloor$. Then it proceeds similar as in the idea of Lemma 7, but restricted to the nodes in a k_j-dominating set $\mathcal{DOM}_j$. That is Algorithm 3 computes $\mathcal{DOM}_j$-SP instead of APSP and executes the algorithm of Lemma 7 restricted to the breadth-first searches rooted in nodes of $\mathcal{DOM}_j$. Now we use tree T_1 to aggregate the minimum estimate in its root in time $\mathcal{O}(D)$ and denote the obtained value by g_{j+1}. The total complexity of this computation is $\mathcal{O}(|\mathcal{DOM}_j| + D)$. Using Lemma 12 and Fact 1 this turns out to be $\mathcal{O}((n/g_j) + D)$.

Lemma 13. *If $g_j \geq 4 \cdot g$, Subroutine ImproveEstimate computes a new estimate g_{j+1} such that $g_{j+1} \leq \frac{3}{4} \cdot g_j$*

Proof. Let C be a minimal cycle in G. Due to executing a BFS only from the nodes in $\mathcal{DOM}_j$ and not from all nodes in V, it is possible, that no BFS is executed from any node in C. Still we know that a BFS is executed from at least one node at distance at most k_j from some node in C. This results in an estimate g_{j+1} of the size of C that differs from its actual length by at most $2 \cdot k_j$ (this is not the best bound, but this is sufficient). That is $g_{j+1} \leq g + 2 \cdot k_j \leq g + g_j/2$. This in turn is at most $\frac{3}{4} \cdot g_j$ since $g_j \geq 4 \cdot g$. $\square$

Lemma 14. *If $g_{j+1}/g_j > 3/4$ we have found found $j_{\max} := j + 1$ such that $g_{j+1} < 4 \cdot g$. Furthermore: $j_{\max} \in \mathcal{O}(\log(g_1/g))$.*

Proof. The first statement immediately follows from Lemma 13. From Lemma 13 we also conclude that $g_j \geq 4 \cdot g$ is only possible during the first $\mathcal{O}(\log(g_1/g))$ iterations. An improvement of this $(\times, 4)$-approximation of the girth by a factor of $3/4$ is only possible during at most four further iterations. Thus $j_{\max} \in \mathcal{O}(\log(g_1/g))$. $\square$

We let *ImproveEstimate* run a last time using an "estimate" $g' := g_{j_{\max}} \cdot \varepsilon/2$ for g. By following a similar reasoning as above, the resulting output differs from g by at most $g_{j_{\max}} \cdot \varepsilon/4$, which is at most $\varepsilon \cdot g$ due to Lemma 14. This implies that the result is a $(\times, 1 + \varepsilon)$-approximation of g.

Algorithm 3 as executed by each node $v \in G$.

Input: accuracy parameter ε Output: $(\times, 1+\varepsilon)$-approximation to the girth of G

```
1: $j := 1$; $g_0 = \infty$
2: perform BFS$_1$ to compute $g_1 := \begin{cases} \infty & : G \text{ is a tree} \\ ecc(1) & : \text{else} \end{cases}$
                                   //** upper bound on $g$
3: while $g_j \leq \frac{3}{4} \cdot g_{j-1}$ do
4:     $g_{j+1} := ImproveEstimate(g_j)$
           //** get better estimate
5:     $j := j + 1$
6: end while
7: $j_{\max} := j$
8: output $ImproveEstimate(g_{j_{\max}} \cdot \varepsilon/2)$ //** final estimate

   Subroutine $ImproveEstimate$ on input $g_j$ :
9: $k_j := \lfloor g_j/4 \rfloor$
10: $\mathcal{DOM}_j := k_j$-dominating set of $G$
11: execute modified $\mathcal{DOM}_j$-SP to estimate small cycles
12: $g_{j+1} :=$ min-aggregate of the estimates any node in $\mathcal{DOM}_j$ found
13: return $g_{j+1}$
```

The time used to find this estimate of g is given by

$$\begin{aligned}
\sum_{j=1}^{j_{\max}} \mathcal{O}\left(\frac{n}{g_j} + D\right) &= \sum_{\log g}^{\nu=\log g_1} \mathcal{O}\left(\frac{n}{2^\nu} + D\right) \\
&\subseteq (\log g_1 - \log g) \cdot \mathcal{O}(D) + \sum_{\log g}^{\nu=\log D} \mathcal{O}\left(\frac{n}{2^\nu}\right) \\
&= \mathcal{O}\left(n/g + D \cdot \log \frac{g_1}{g}\right).
\end{aligned}$$

The way we choose g_1 allows us to bound $g_1 = \mathcal{O}(D)$ and to conclude a total runtime of $\mathcal{O}\left(n/g + D \cdot \log \frac{D}{g}\right)$. In case $D \cdot \log \frac{D}{g} > n$, we could just use the exact algorithm for g (described in Lemma 7) to obtain a linear runtime. $\square$

7 Distinguishing Graphs of Small Diameter

7.1 Distinguishing Diameter 2 from 3 takes time $\Omega(n/B + D)$

Theorem 8. *Let $\mathcal{G}$ be the family of all graphs of diameter 2 or 3. For any $n \geq 6$ and $B \geq 1$ and sufficiently small ε any distributed randomized ε-error* *algorithm A that can decide whether a graph $G \in \mathcal{G}$ has diameter 2 or 3 needs $\Omega\left(\frac{n}{B} + D\right)$ time for some n-node graph.*

Remark 6. *This is an improvement of Theorem 5.1. of [22]: computing the diameter of a graph takes time $\Omega(n/B + D)$ even if the diameter is 3 (compared to five as in [22]).*

Proof. Deciding whether a graph G has diameter less than 3 or not is the decision-version $diam_3$ of the function $diam$, that is

$$diam_3(G) := \begin{cases} 1 & : diam(G) < 3 \\ 0 & : \text{else} \end{cases}$$

Again we use the technique described in [22]: We define a two-party communication problem $diam'_3$ according to the graph-problem $diam_3$ in a canonical way as described in Section 5.1. To prove bounds for $diam'_3$ depending on n, we choose the length k of the input to the base-function h to depend on n and set $k(n) := \left\lfloor \frac{n-2}{4} \right\rfloor$. As base-function h we consider the $disj_{k(n)^2}$ problem. Now we need to define a reduction $\mathcal{R}$ that given inputs a and b to h maps $(Alice, a)$ and (Bob, b) to $(G_a, C_{k(n)^2})$ and $(G_b, C_{k(n)^2})$ which in turn are inputs for $diam'_3$.

During the reduction $\mathcal{R}$, Alice defines L and Bob defines R in the same way as in Section 5.2.

Alice adds a node c_L to G_a that Alice connects to all nodes in L and Bob adds a node c_R to G_b that is connected to all nodes in R. Furthermore, for each $i \in [k(n)^2 - 1]$, if and only if $a(i) = 0$, Alice connects node $l_{i \bmod k(n)}$ from the upper half to node $l_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}$ in the lower half by an edge. In addition Alice adds clique-edges $\{(l_\mu, l_\nu) | \mu \neq \nu \in [k(n) - 1]\}$ in the upper part as well as $\{(l_\mu, l_\nu) | \mu \neq \nu \in \{k(n), \ldots, k(n) - 1\}$ in the lower part. An example of this can be found in Figure 2 with detailed explanations in the caption. Note that this is the only part that depends on the input a and we can represent all values of the $k(n)^2$ bits of a by the $k(n)^2$ possible edges between the $k(n)$ nodes $\{l_\nu : \nu \in [k(n) - 1]\}$ and the $k(n)$ nodes $\{l_\nu : \nu \in \{k(n), \ldots, 2 \cdot k(n) - 1\}\}$. We call the resulting graph $G_a = (V_a, E_a)$ (see formal definition below) and define G_b in a similar way depending on b. That is e.g. for each $i \in [k(n)^2 - 1]$, if and only if $b(i) = 0$, Bob connects node $r_{i \bmod k(n)}$ from the upper half to node $r_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}$ in the lower half by an edge. Observe that $|V_a| = |V_b| = 2 \cdot k(n) + 1$ which is smaller than $n/2$ due to the choice of $k(n)$ and the fact that $n \geq 6$ implies $k(n) \geq 1$. Therefore we add $n - 4 \cdot k(n) - 2$ fill up nodes w_i to G_b and connect them to each of the nodes $R \cup \{c_R\}$. This ensures that the final graph $m((G_a, C_{k(n)^2}), (G_b, C_{k(n)^2}))$ has exactly n nodes such that the lower bound holds for all n. Formally we have:

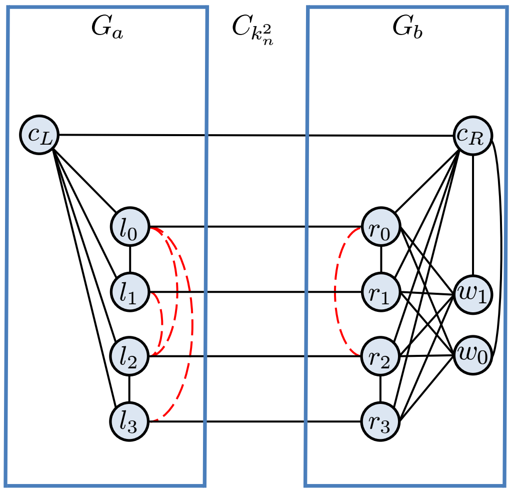

Figure 2: The above graph $G_{a,b}$ is for $n = 12$ (therefore we set $k(n) = 2$) and results from inputs $a = (0, 0, 0, 1), b = (0, 1, 1, 1)$ using the reduction $\mathcal{R}$. Accordingly the dashed red edges represent a and b. To be more specific, edge (l_0, l_2) represents $a(0) = 0$, edge (l_0, l_3) represents $a(1) = 0$, edge (l_1, l_2) represents $a(2) = 0$ and edge (r_0, r_2) represents $b(0) = 0$. This causes the diameter to be larger than 2 witnessed by $d(l_1, r_3) = 3$. Using Theorem 9 we conclude that a and b are not disjoint, which is indeed true in this example.

$$\begin{aligned}
V_a :=\ & L \cup \{c_L\} \\
E_a :=\ & \textstyle\bigcup_{\nu=0}^{2\cdot k(n)-1} \{(l_\nu, c_L)\} \\
& \cup\{(l_\nu, l_\mu) : \nu \neq \mu \wedge \nu, \mu \in [k(n) - 1]\} \\
& \cup\{(l_\nu, l_\mu) : \nu \neq \mu \wedge \nu, \mu \in \{k(n), \dots, 2 \cdot k(n) - 1\}\} \\
& \cup\{(l_{i \bmod k(n)}, l_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}) : \\
& \qquad\qquad i \in [k(n)^2 - 1], a(i) = 0\}
\end{aligned}$$

$$\begin{aligned}
V_b :=\ & R \cup \{c_R\} \cup \{w_i : i \in [n - 4 \cdot k(n) - 2]\} \\
E_b :=\ & \textstyle\bigcup_{\nu=0}^{2\cdot k(n)-1} \{(r_\nu, c_R)\} \\
& \cup\{(w_i, v) : i \in [n - 4 \cdot k(n) + 1], v \in R \cup \{c_R\}\} \\
& \cup\{(r_{i \bmod k(n)}, r_{k(n)+\left\lfloor \frac{i}{k(n)} \right\rfloor}) : \\
& \qquad\qquad i \in [k(n)^2 - 1], b(i) = 0\}
\end{aligned}$$

Finally we define the cut-set $C_{k(n)^2} := \{(l_\nu, r_\nu) : \nu \in [2\cdot k(n)-1]\} \cup \{(c_L, c_R)\}$ to consist of $2 \cdot k(n) + 1$ edges. Thus $\mathcal{R}$ is a $(2 \cdot k(n) + 1)$-reduction. Observe that $(G_a, C_{k(n)^2})$ can be computed from a without knowing b and $(G_b, C_{k(n)^2})$ can be computed from b without knowing a, thus the reduction $\mathcal{R}$ has the desired properties. Now we set $G_{a,b} := m((G_a, C_{k(n)^2}), (G_b, C_{k(n)^2}))$.

Lemma 15. *The graph $G_{a,b}$ is an n-node graph with diameter at most* 3.

Proof. The graph $G_{a,b}$ contains the $4 \cdot k(n)$ nodes in L and R. Furthermore it contains $\{c_L, c_R\}$ and $n - 4 \cdot k(n) - 2$ fill up nodes. Thus, in total there are n nodes in the graph.

We prove that the diameter is at most 3 by showing that for any nodes u and v in $G_{a,b}$ the distance $d(u, v)$ is at most 3. To do this we distinguish three cases:

1. nodes u and v are both in G_a: Observe that every node in G_a is connected to c_L. This implies that the distance between any two nodes u and v in G_a is at most 2.

2. nodes u and v are both in G_b: This case is completely analog to the previous, thus $d(u, v) \leq 2$.

3. node u is in G_a and node v is in G_b (or the other way round): There is the following u-v-path of length 3: (u, c_L, c_R, v).

□

Now we relate the problem of deciding whether a and b are disjoint to the problem of computing the diameter of a graph. To achieve this we extend the analysis of the diameter of $G_{a,b}$.

Theorem 9. *The diameter of $G_{a,b}$ is* 2 *if the sets a and b are disjoint, else it is* 3.

Proof. If a and b are not disjoint, then there exists an $i \in [n-1]$ such that $a(i) = b(i) = 1$. Let us fix such an i for now and let $\nu := i \mod k(n)$ and $\mu = k(n) + \left\lfloor \frac{i}{k(n)} \right\rfloor$. We show that the two nodes l_ν and r_μ have distance of at least 3. The path must contain an edge from the cut-set $C_{k(n)^2}$ since these are the only edges that connect G_a to G_b. To obtain a path of length 2 we were only allowed to add one more edge from either G_a or G_b. When looking at the construction, the only two paths of length 2 that we could hope for are (l_ν, l_μ, r_μ) and (l_ν, r_ν, r_μ). However, due to $a(i) = b(i) = 1$ and the implied choice of ν and μ causes that the construction of $G_{a,b}$ does neither include edge (l_ν, l_μ) nor edge (r_ν, r_μ). Thus none of these paths exists and we conclude that $d(l_\nu, r_\mu) > 2$. Combined with Lemma 15 this implies that $d(l_\nu, r_\mu) = 3$ if and only if a and b are not disjoint.

Conversely if a and b are disjoint, the diameter of $G_{a,b}$ is at most 2. We prove this by showing that for any nodes u and v in $G_{a,b}$ the distance $d(u, v)$ is at most 2. To do this we distinguishing three cases:

1. Nodes u and v are both in G_a: same as case 1 in proof of Lemma 15, thus $d(u, v) \leq 2$.

2. Nodes u and v are both in G_b: same as case 2 in proof of Lemma 15, thus $d(u,v) \leq 2$.

3. Node u is in G_a and node v is in G_b (or the other way round): When considering the nodes $\{c_L, c_R, w_0, w_1, w_2, \dots\}$, we notice that from each of these nodes each other node in the graph can be reached within 2 hops. Now we can assume without loss of generality that $u = l_\nu \in L$ and $v = r_\mu \in R$ for some $\mu, \nu \in [2 \cdot k(n) - 1]$. Since we assumed that a and b are disjoint there must be either at least one of the edges (l_ν, l_μ) or (r_ν, r_μ) in case that one of the nodes is in **UP** and the other node is in **LP**. Thus there is at least one of the paths (l_ν, l_μ, r_μ) or (l_ν, r_ν, r_μ) witnessing that $d(l_\nu, r_\mu) \leq 2$. In the remaining case u, v are both in **UP** or both in **LP**, w make use of the clique-edges and conclude that u and v are connected by path (l_ν, r_ν, r_μ) of length 2.

□

Proof. (of Theorem 8) To solve $\mathrm{disj}_{\mathrm{k(n)}^2}$ using any algorithm for *diam* we use the reduction from $diam'_3$ to $diam_3$ (*diam*, respectively) and the reduction $\mathcal{R}$ from $\mathrm{disj}_{\mathrm{k(n)}^2}$ to $diam'_3$ presented above. We can apply Theorem 3 and know that

$$\frac{R_\varepsilon^{cc-pub}(\mathrm{disj}_{\mathrm{k(n)}^2})}{2 \cdot |C_{k(n)^2}| \cdot B} \leq R_\varepsilon^{dc}(diam_3)$$

Due to Theorem 2 we know that $R_\varepsilon^{cc-pub}(\mathrm{disj}_{\mathrm{k(n)}^2})$ is at least $\Omega(k(n)^2)$. Together with the fact that $|C_{k(n)^2}| = 2 \cdot k(n) + 1$ we conclude that $R_\varepsilon^{dc}(diam_3) \in \Omega(k(n))$. We obtain the stated result since we chose $k(n) := \lceil \frac{n-2}{4} \rceil$. □

□

Lemma 16. *Computing a $(\times, 3/2-\varepsilon)$-approximation for APSP takes $\Omega(\frac{n}{B} + D)$ time.*

Proof. From Theorem 8 we know that $\Omega(\frac{n}{B} + D)$ is needed to distinguish diameter 3 from 2. Any $(\times, 3/2 - \varepsilon)$-approximation algorithm for APSP can distinguishing graphs of diameter 2 from graphs of diameter 2 with only $O(D) = O(1)$ communication rounds overhead. This can be extended to the case of larger diameters: Construct a graph by adding a path of the desired length to one node in the graph. In this setting we are interested in deciding whether the diameter of a certain subgraph is 2 or 3. This subgraph is just the previously described graph to which we later added the path. □

7.2 Distinguishing Diameter 2 from 4 in time $\mathcal{O}(\sqrt{n \cdot \log n})$

We now demonstrate how to distinguish graphs of diameter 2 from graphs of diameter 4 in time $\mathcal{O}(\sqrt{n})$. This algorithm is inspired by an algorithm called 2-vs-4 presented in [2]. The authors of [2] considered the idea leading to this algorithm to be an important step towards obtaining their $(\times, 3/2)$-approximation algorithm. In the light of Theorem 8 (and Theorem 4), where we showed that distinguishing diameter 2 graphs from diameter 3 graphs (and diameter k graphs from diameter $k+2$ graphs for $k \geq 4$, respectively) takes long time, it is intriguing that distinguishing diameter 2 graphs from 4 graphs can be done rather fast. Before we state Algorithm 4 (a.k.a. Algorithm 2-vs-4), we introduce some notation and review some results of [2] depending on a parameter s. Later in the paper they choose the parameter s to be $s := \sqrt{n \cdot \log n}$ and we do the same in our distributed setting.

Definition 15. *We define* $L(V) := \{u \in V : |N_1(u)| < s\}$ *and* $H(V) := V \setminus L(V) = \{u \in V : |N_1(u)| \geq s\}$.

Remark 7. *(Version of Remark 2.1. in [2]). Choosing a set of* $\Theta(s^{-1} \cdot n \cdot \log n)$ *vertices uniformly at random results in an* 1*-dominating set for* $H(V)$ *with high probability.*

Theorem 10. *Algorithm 4 distinguishes diameter* 2 *from* 4 *and can be implemented in a distributed way (using randomness) terminating whp within* $\mathcal{O}(\sqrt{n \cdot \log n})$ *rounds of communication.*

Algorithm 4 – same as **Algorithm** 2-vs-4 from [2].
Input: G with diameter 2 or 4   Output: diameter of G

```
1: if $L(V) \neq \emptyset$ then
2:     choose $v \in L(V)$
3:     compute a BFS tree from each vertex in $N_1(v)$
4: else
5:     compute a dominating set $\mathcal{DOM}$ for $H(V)$
6:     compute a BFS tree from each vertex in $\mathcal{DOM}$
7: end if
8: if all BFS trees have depth 2 then
9:     return 2
10: else
11:     return 4
12: end if
```

Proof. Correctness is shown in Theorem 3.1. in [2] and we only need to take care of analyzing the distributed runtime. Each node can decide internally without communication whether it belongs to set $L(V)$ or $H(V)$. Choosing the node v in Line 2 takes $\mathcal{O}(D)$. Computing the BFS trees from each vertex in $N_1(v)$ can be done in time $\mathcal{O}(|N_1(v)| \cdot D) = \mathcal{O}(s \cdot D) = \mathcal{O}(\sqrt{n \cdot \log n})$, due to the choice of v and s as well as the fact that $D \leq 4 = \mathcal{O}(1)$. (Note: This is already fast enough and we do not need to use $N_1(v)$-SP here.) Computing a dominating set $\mathcal{DOM}$ for $H(V)$ can be done locally without communication: each node in $H(V)$ independently joins $\mathcal{DOM}$ with probability $\sqrt{\frac{\log n}{n}}$. With high probability this results in a set $\mathcal{DOM}$ of size $\Theta(\sqrt{n \cdot \log n})$ which in turn is a dominating set with high probability according to Remark 7. Computing BFS trees from each of the vertices in $\mathcal{DOM}$ takes $\mathcal{O}(|\mathcal{DOM}| \cdot D) = \mathcal{O}(\sqrt{n \cdot \log n})$. Deciding whether all computed BFS trees have depth at most 2 can be done by max-aggregation in an arbitrary node in time $\mathcal{O}(D) = \mathcal{O}(1)$. Thus the total time complexity is $\mathcal{O}(\sqrt{n \cdot \log n})$. □

8 Counting the Number of Nodes in the Greater Neighborhood

In this section we argue that computing all depth k-BFS trees can be a hard task by giving a worst case example for $k = 2$. Towards this end we construct a family of graphs where computing all depth 2-BFS trees takes $\Omega(n/B + D)$ time. At the same time these graphs have girth 3.

Theorem 11. *Let $\mathcal{G}$ be the family of all graphs of diameter 2 or 3. For any $n \geq 6$ and $B \geq 1$ and sufficiently small ε any distributed randomized ε-error algorithm A that can compute a 2-BFS trees for each nodes needs $\Omega(n/B + D)$ time for some n-node graph.*

Proof. Consider the following problem: "Is there a node v, such that the number of nodes in the 2-neighborhood $N_2(v)$ (including v) is strictly less than n?" The problem of computing all 2-BFS trees can be reduced to this problem in time $\mathcal{O}(D) = \mathcal{O}(1)$: Simply check whether there is a node that is not included in some 2-BFS tree. This problem in turn can be reduced to distinguishing whether the graph used in the proof of Theorem 8 has diameter 2 or 3. If for all nodes the number of nodes in the k-neighborhood is $n-1$, this means that the diameter is 2. Else the diameter is 3. Applying Theorem 8 immediately yields the lower bound. □

Acknowledgments: We thank an anonymous reviewer for helpful comments on the presentation.

References

[1] J. Abram and I. Rhodes. A decentralized shortest path algorithm. In *Proceedings of the 16th Allerton Conference on Communication, Control and Computing (Allerton)*, pages 271–277, 1978.

[2] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing (SICOMP)*, 28(4):1167–1181, 1999.

[3] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 569–575, 1991.

[4] N. Alon, O. Margalit, Z. Galilt, and M. Naor. Witnesses for boolean matrix multiplication and for shortest paths. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 417–426, 1992.

[5] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[6] U. Black. *IP routing protocols: RIP, OSPF, BGP, PNNI and Cisco routing protocols.* Prentice Hall PTR, 2000.

[7] G. Blelloch, V. Vassilevska, and R. Williams. A new combinatorial approach for sparse graph problems. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I (ICALP)*, pages 108–120, 2008.

[8] P. Carrington, J. Scott, and S. Wasserman. *Models and methods in social network analysis.* Cambridge University Press, 2005.

[9] T. Chan. All-pairs shortest paths for unweighted undirected graphs in o (mn) time. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm (SODA)*, pages 514–523. ACM, 2006.

[10] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *Proceedings of the 39th annual ACM symposium on Theory of computing*, (STOC), pages 590–598, New York, NY, USA, 2007. ACM.

[11] K. Chandy and J. Misra. Distributed computation on graphs: Shortest path algorithms. *Communications of the ACM (CACM)*, 25(11):833–837, 1982.

[12] C. Chen. A distributed algorithm for shortest paths. *IEEE Transactions on Computers (TC)*, 100(9):898–899, 1982.

[13] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation (JSC)*, 9(3):251–280, 1990.

[14] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. *Proceedings of the 43rd annual ACM Symposium on Theory of Computing (STOC)*, 2011.

[15] W. Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *International journal of computer mathematics*, 32(1-2):49–60, 1990.

[16] D. Dolev, C. Lenzen, and S. Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting. *CoRR*, http://arxiv.org/abs/1201.6652, 2012.

[17] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing (SICOMP)*, 29:1740, 2000.

[18] M. Elkin. Computing almost shortest paths. In *Proceedings of the 20th annual ACM symposium on Principles of distributed computing (PODC)*, pages 53–62, 2001.

[19] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. In *Proceedings of the 23rd annual ACM symposium on Theory of computing (STOC)*, pages 123–133, 1991.

[20] G. Flake, S. Lawrence, and C. Giles. Efficient identification of web communities. In *Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 150–160. ACM, 2000.

[21] M. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing (SICOMP)*, 5:83, 1976.

[22] S. Frischknecht, S. Holzer, and R. Wattenhofer. Networks Cannot Compute Their Diameter in Sublinear Time. In *Proceedings of the 23rd annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1162.

[23] Y. Han. Improved algorithm for all pairs shortest paths. *Information Processing Letters (IPL)*, 91(5):245–250, 2004.

[24] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing (SICOMP)*, 7:413, 1978.

[25] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. In *Proceedings of the 27th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 263–272, 2008.

[26] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997.

[27] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998.

[28] N. Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

[29] J. McQuillan, I. Richer, and E. Rosen. The new routing algorithm for the arpanet. *IEEE Transactions on Communications (TC)*, 28(5):711–719, 1980.

[30] P. Merlin and A. Segall. A failsafe distributed routing protocol. *IEEE Transactions on Communications (TC)*, 27(9):1280–1287, 1979.

[31] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical Report 1999-66, Stanford InfoLab*, 1999.

[32] D. Peleg. *Distributed computing: a locality-sensitive approach*. 2000.

[33] D. Peleg, L. Roditty, and E. Tal. Distributed algorithms for network diameter and girth. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP), to appear*, 2012.

[34] L. Roditty and R. Tov. Approximating the girth. In *Proceedings of the 22nd annual ACM-SIAM symposium on Discrete algorithm (SODA)*, pages 1446–1454, 2011.

[35] L. Roditty and V. Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 180–189, 2011.

[36] L. Roditty and V. Williams. Subquadratic time approximation algorithms for the girth. In *Proceedings of the 23rd annual ACM-SIAM symposium on Discrete algorithm (SODA)*, pages 833–845, 2012.

[37] A. Sarma, M. Dinitz, and G. Pandurangan. Efficient computation of distance sketches in distributed networks. *24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), to appear*, 2012.

[38] M. Schwartz and T. Stern. Routing techniques used in computer communication networks. *IEEE Transactions on Communications (TC)*, 28(4):539–552, 1980.

[39] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences (JCSS)*, 51(3):400–403, 1995.

[40] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 605–614, 1999.

[41] W. Tajibnapis. A correctness proof of a topology information maintenance protocol for a distributed computer network. *Communications of the ACM (CACM)*, 20(7):477–485, 1977.

[42] T. Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters (IPL)*, 43(4):195–199, 1992.

[43] T. Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. *Proceedings of the 10th Annual International Computing and Combinatorics Conference (COCOON)*, pages 278–289, 2004.

[44] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.

[45] S. Toueg. An all-pairs shortest-paths distributed algorithm. *Tech. Rep. RC 8327, IBM TJ Watson Research Center, Yorktown Heights, NY 10598, USA*, 1980.

[46] V. Williams. Multiplying matrices faster than coppersmith-winograd. *Proceedings of the 44th annual ACM Symposium on Theory of Computing (STOC)*, 2012.

[47] A. Yao. Some complexity questions related to distributive computing. In *Proceedings of the 11th annual ACM symposium on Theory of computing (STOC)*, pages 209–213, 1979.

[48] S. Yardi, D. Romero, G. Schoenebeck, and D. Boyd. Detecting spam in a twitter network. *First Monday*, 15(1), 2009.

[49] R. Yuster and U. Zwick. Finding even cycles even faster. *SIAM Journal on Discrete Mathematics (SIDMA)*, 10:209, 1997.

[50] U. Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. *Algorithms and Computation*, pages 841–843, 2005.