Algebraic Methods in the Congested Clique*

Keren Censor-Hillel†
Technion
ckeren@cs.technion.ac.il

Petteri Kaski‡
Helsinki Institute for Information Technology & Aalto University
petteri.kaski@aalto.fi

Janne H. Korhonen
Helsinki Institute for Information Technology & University of Helsinki
janne.h.korhonen@helsinki.fi

Christoph Lenzen
MPI for Informatics
clenzen@mpi-inf.mpg.de

Ami Paz
Technion
amipaz@cs.technion.ac.il

Jukka Suomela
Helsinki Institute for Information Technology & Aalto University
jukka.suomela@aalto.fi

ABSTRACT

In this work, we use algebraic methods for studying distance computation and subgraph detection tasks in the *congested clique* model. Specifically, we adapt parallel matrix multiplication implementations to the congested clique, obtaining an $O(n^{1-2/\omega})$ round matrix multiplication algorithm, where $\omega < 2.3728639$ is the exponent of matrix multiplication. In conjunction with known techniques from centralised algorithmics, this gives significant improvements over previous best upper bounds in the congested clique model. The highlight results include:

- triangle and 4-cycle counting in $O(n^{0.158})$ rounds, improving upon the $O(n^{1/3})$ triangle counting algorithm of Dolev et al. [DISC 2012],
- a $(1+o(1))$-approximation of all-pairs shortest paths in $O(n^{0.158})$ rounds, improving upon the $\tilde{O}(n^{1/2})$-round $(2+o(1))$-approximation algorithm of Nanongkai [STOC 2014], and
- computing the girth in $O(n^{0.158})$ rounds, which is the first non-trivial solution in this model.

In addition, we present a novel constant-round combinatorial algorithm for detecting 4-cycles.

Categories and Subject Descriptors

F.1.1 [**Computation by Abstract Devices**]: Models of Computation; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; G.2.2 [**Mathematics of Computing**]: Discrete Mathematics—*Graph Theory*

Keywords

Distributed computing; congested clique model; lower bounds; matrix multiplication; subgraph detection; distance computation

1. INTRODUCTION

Algebraic methods have become a recurrent tool in centralised algorithmics, employing a wide range of techniques (e.g., [10–16, 21–23, 27, 29, 30, 44, 45, 51, 59, 72, 73]). In this paper, we bring techniques from the algebraic toolbox to the aid of distributed computing, by leveraging fast matrix multiplication in the *congested clique* model.

In the congested clique model, the n nodes of a graph G communicate by exchanging messages of $O(\log n)$ size in

	Running time		
Problem	This work	Prior work	
mat. mult. (semiring)	$O(n^{1/3})$	—	
mat. mult. (ring)	$O(n^{0.158})$	$O(n^{0.373})$	[26]
triangle counting	$O(n^{0.158})$	$O(n^{1/3}/\log n)$	[25]
4-cycle detection	$O(1)$	$O(n^{1/2}/\log n)$	[25]
4-cycle counting	$O(n^{0.158})$	$O(n^{1/2}/\log n)$	[25]
k-cycle detection	$2^{O(k)}n^{0.158}$	$O(n^{1-2/k}/\log n)$	[25]
girth	$O(n^{0.158})$	—	
APSP:			
weighted, directed	$O(n^{1/3}\log n)$	—	
· weighted diameter U	$O(Un^{0.158})$	—	
· $(1+o(1))$-approx.	$O(n^{0.158})$	—	
· $(2+o(1))$-approx.		$\tilde{O}(n^{1/2})$	[58]
APSP:			
unweighted, undirected	$O(n^{0.158})$	—	
· $(2+o(1))$-approx.		$\tilde{O}(n^{1/2})$	[58]

Table 1: Our results versus prior work, using $\omega < 2.3729$ [34]; $\tilde{O}$ hides polylogarithmic factors.

*A full version of this paper is available at [18].

†Supported by ISF Individual Research Grant 1696/14

‡Supported by the European Research Council under the European Union's Seventh Framework programme (FP/2007–2013) / ERC Grant Agreement n. 338077.

a *fully-connected* synchronous network; initially, each node is aware of its neighbours in G. In comparison with the traditional CONGEST model [61], the key difference is that a pair of nodes can communicate directly even if they are not adjacent in graph G. The congested clique model masks away the effect of *distances* on the computation and focuses on the limited *bandwidth*. As such, it has been recently gaining increasing attention [25, 26, 37, 38, 47, 50, 52, 58, 60, 64], in an attempt to understand the relative computational power of distributed computing models.

The key insight of this paper is that matrix multiplication algorithms from parallel computing can be adapted to obtain an $O(n^{1-2/\omega})$ round matrix multiplication algorithm in the congested clique, where $\omega < 2.3728639$ is the matrix multiplication exponent [34]. Combining this with well-known centralised techniques allows us to use fast matrix multiplication to solve various combinatorial problems, immediately giving $O(n^{0.158})$-time algorithms in the congested clique for many classical graph problems. Indeed, while most of the techniques we use in this work are known beforehand, their combination gives significant improvements over the best previously known upper bounds. Table 1 contains a summary of our results, which we overview in more details in what follows.

1.1 Matrix Multiplication in the Congested Clique

As a basic primitive, we consider the computation of the product $P = ST$ of two $n \times n$ matrices S and T on a congested clique of n nodes. We will tacitly assume that the matrices are initially distributed so that node v has row v of both S and T, and it will receive row v of P in the end. Recall that the matrix multiplication exponent ω is defined as the infimum over σ such that the product of two $n \times n$ matrices can be computed with $O(n^\sigma)$ arithmetic operations; it is known that $2 \leq \omega < 2.3728639$ [34], and it is conjectured, though not unanimously, that $\omega = 2$.

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

Theorem 1 follows by adapting known parallel matrix multiplication algorithms for semirings [1, 55] and rings [7, 53, 56, 71] to the clique model, via the routing technique of Lenzen [47]. In fact, with little extra work one can show that the resulting algorithm is also *oblivious*, that is, the communication pattern is predefined and does not depend on the input matrices. Hence, the oblivious routing technique of Dolev et al. [25] suffices for implementing these algorithms.

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

Distributed matrix multiplication exponent. Analogously to the matrix multiplication exponent, we denote by ρ the exponent of matrix multiplication in the congested clique model, that is, the infimum over all values σ such that there exists a matrix multiplication algorithm in the congested clique running in $O(n^\sigma)$ rounds. In this notation, Theorem 1 gives us

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

prior to this work, it was known that $\rho \leq \omega - 2$ [26].

For the rest of this paper, we will – as is convention for centralised algorithmics – slightly abuse notation by writing n^ρ for the complexity of matrix multiplication in the congested clique. This hides factors of $O(n^\varepsilon)$ resulting from the fact that ρ is defined as infimum of an infinite set. We also use $\tilde{O}$ and $\tilde{\Omega}$ notation to hide polylogarithmic factors.

Lower bounds for matrix multiplication. Our results are optimal in the sense that for any sequential matrix multiplication implementation, no scheme simulating it on the congested clique can give a faster algorithm than the construction underlying Theorem 1; this follows from known results for parallel matrix multiplication [2, 8, 42, 70]. Moreover, we note that for the *broadcast congested clique model*, where each node is required to send the same message to all nodes in any given round, recent lower bounds [39] imply that matrix multiplication requires $\tilde{\Omega}(n)$ rounds.

1.2 Applications in Cycle Detection and Counting

Our first application of fast matrix multiplication is to the problems of triangle counting [43] and 4-cycle counting.

COROLLARY 2. *On directed and undirected graphs, counting triangles and 4-cycles takes $O(n^\rho)$ rounds.*

For $\rho \leq 1-2/\omega$, this is an improvement upon the previously best known $O(n^{1/3})$-round triangle detection algorithm of Dolev et al. [25]. In particular, we disprove the conjecture of Dolev et al. [25] that any deterministic oblivious algorithm for detecting triangles requires $\tilde{\Omega}(n^{1/3})$ rounds.

When only detection of cycles is required, we observe that combining fast distributed matrix multiplication with the well-known technique of *colour-coding* [5] allows to detect k-cycles in $\tilde{O}(n^\rho)$ rounds for any constant k. This improves upon the subgraph detection algorithm of Dolev et al. [25], which requires $\tilde{O}(n^{1-2/k})$ rounds for detecting (arbitrary) subgraphs of k nodes.

COROLLARY 3. *For any graph, the existence of k-cycles can be detected in $2^{O(k)} n^\rho \log n$ rounds.*

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

THEOREM 4. *On undirected graphs, the existence of 4-cycles can be detected in $O(1)$ rounds. If such a cycle exists, one such cycle can also be reported in $O(1)$ rounds.*

1.3 Applications in Girth Computation

We compute the girth of a graph by leveraging a known trade-off between the girth and the number of edges of the graph [54]. Roughly, we detect short cycles fast, and if they do not exist then the graph must have sufficiently few edges to be learned by all nodes. As far as we are aware, this is the first algorithm to compute the girth in this setting.

THEOREM 5. *For undirected graphs, the girth can be computed in* $\tilde{O}(n^\rho + n^{o(1)})$ *rounds.*

A similar result on directed girth follows by adapting an algorithm by Itai and Rodeh [43].

COROLLARY 6. *For directed graphs, the girth can be computed in* $\tilde{O}(n^\rho)$ *rounds.*

1.4 Applications in Distance Computation

The *all-pairs shortest paths* problem (APSP) likewise admits algorithms based on matrix multiplication. The basic idea is to compute the n^{th} power of the input graph's weight matrix over the min-plus semiring, by iteratively computing squares of the matrix [28, 33, 57].

COROLLARY 7. *For directed graphs with integer weights in* $\{-M, -M+1, \ldots, M\}$*, APSP can be computed in* $O(n^{1/3}(\log n + \log M))$ *rounds.*

We can leverage fast ring matrix multiplication to improve upon the above result; however, the use of ring matrix multiplication necessitates some trade-offs or extra assumptions. For example, for unweighted and undirected graphs, it is possible to recover the exact shortest paths from powers of the adjacency matrix over the Boolean semiring [66].

COROLLARY 8. *For undirected, unweighted graphs, APSP can be computed in* $\tilde{O}(n^\rho)$ *rounds.*

For small integer weights, we use the well-known idea of embedding a min-plus semiring matrix product into a matrix product over a ring; this gives a multiplicative factor to the running time proportional to the length of the longest path.

COROLLARY 9. *For directed graphs with positive integer weights and weighted diameter* U*, APSP can be computed in* $\tilde{O}(Un^\rho)$ *rounds.*

While this is relevant only for graphs of a small weighted diameter, the idea can be combined with weight rounding [58, 65, 76] to obtain a fast approximate APSP algorithm without such limitations.

COROLLARY 10. *For directed graphs with integer weights in* $\{0, 1, \ldots, 2^{n^{o(1)}}\}$*, APSP can be approximated within a factor of* $1 + o(1)$ *in* $O(n^{\rho+o(1)})$ *rounds.*

This improves on a $(2 + o(1))$-approximation algorithm in $\tilde{O}(n^{1/2})$ rounds by Nanongkai [58].

1.5 Additional Related Work

Computing distances in graphs, such as the diameter, all-pairs shortest paths (APSP), and single-source shortest paths (SSSP) are fundamental problems in most computing settings. The reason for this lies in the abundance of applications of such computations, evident also by the huge amount of research dedicated to it [19, 20, 31, 35, 36, 68, 69, 74–77].

In particular, computing graph distances is vital for many distributed applications and, as such, has been widely studied in the CONGEST model of computation [61], where n processors located in n distinct nodes of a graph G communicate over the graph edges using $O(\log n)$-bit messages. Specifically, many algorithms and lower bounds were given for computing and approximating graph distances in this setting [24, 32, 40, 41, 46, 48, 49, 58, 62, 63]. Some lower bounds apply even for graphs of small diameter; however, these results boil down to graphs with *bottleneck* edges limiting the amount of information that can be exchanged between different parts of the graph quickly.

The intuition that the congested clique model would abstract away distances and bottlenecks and bring to light only the congestion challenge has proven inaccurate. Indeed, a number of tasks have been shown to admit sub-logarithmic or even constant-round solutions, exceeding by far what is possible in the CONGEST model with only low diameter. The pioneering work of Lotker et al. [52] shows that a minimum spanning tree (MST) can be computed in $O(\log \log n)$ rounds; this result was later improved by Pemmaraju and Sardeshmukh [64], showing that MST can be constructed in $O(\log \log \log n)$ rounds. Hegeman et al. [38] show how to construct a 3-ruling set, with applications to maximal independent set and an approximation of the MST in certain families of graphs; sorting and routing have been recently addressed by various authors [47, 50, 60]. A connection between the congested clique model and the MapReduce model is discussed by Hegeman and Pemmaraju [37], where algorithms are given for colouring problems. On top of these positive results, Drucker et al. [26] recently proved that essentially *any* non-trivial unconditional lower bound on the congested clique would imply novel circuit complexity lower bounds.

The same work also points out the connection between fast matrix multiplication algorithms and triangle detection in the congested clique. Their construction yields an $O(n^{\omega-2+\varepsilon})$-round algorithm for matrix multiplication over rings in the congested clique model, giving also the same running bound for triangle detection; if $\omega = 2$, this gives $\rho = 0$, matching our result. However, with the currently best known centralised matrix multiplication algorithm, the running time of the resulting triangle detection algorithm is $O(n^{0.3729})$ rounds, still slower than the combinatorial triangle detection of Dolev et al. [25], and if $\omega > 2$, the solution presented in this paper is faster.

2. MATRIX MULTIPLICATION ALGORITHMS

In the congested clique matrix multiplication, we want to compute the product $P = ST$ of two $n \times n$ matrices $S = (S_{ij})$ and $T = (T_{ij})$ on a congested clique with n nodes; the local input for each node $v \in V$ is the row v of both S and T, and the at the end of the computation each node $v \in V$ will output the row v of P.

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

Theorem 1 follows by simulating known parallel matrix multiplication algorithms in the congested clique model using the routing schemes of Dolev et al. [25] or Lenzen [47]. Roughly speaking, the idea in both of these algorithms is to reduce the $n \times n$ matrix product into n products of smaller matrices, where each product is handled locally by a single node.

The first part of the theorem follows from the so-called parallel 3D matrix multiplication algorithm [1, 55], essentially

a parallel implementation of the school-book solution, which splits the multiplied matrices into $n^{1/3} \cdot n^{1/3} = n^{2/3}$ blocks of size $n^{2/3} \times n^{2/3}$. Since each block needs to be multiplied by $n^{1/3}$ other blocks, we need $n^{1/3}$ nodes to hold a copy of each block. This allows us to distribute the multiplications such that each node locally multiplies a single pair of blocks. Communicating the input blocks and collecting the entries of the output matrix takes $O(n^{1/3})$ rounds, as each node needs to collect and send information about $n^{4/3}$ entries.

For the second part of the theorem, we use a known result [17] that says that any $O(n^\sigma)$ matrix multiplication algorithm can be converted into a *bilinear* matrix multiplication algorithm with running time of $O(n^{\sigma+\varepsilon})$, for any $\varepsilon > 0$. In a bilinear matrix multiplication algorithm, such as the algorithm of Strassen [67], the matrix product is computed by first computing $m < n^3$ linear combinations of entries of both matrices, then performing m multiplications of pairs of linear combinations, and finally computing the output matrix entries as linear combinations of the multiplication outputs. This can be computed over a ring, and can be obtained by recursively performing the above on a partition of the matrix into blocks, giving the speed-up in running time.

We then use a scheme that allows to adapt any bilinear matrix multiplication algorithm into a fast parallel matrix multiplication algorithm [7, 53, 56, 71]. Hence, with some additional details, the result is obtained by using an $O(n^\sigma)$ centralised bilinear matrix multiplication algorithm to reduce the $n \times n$ product to n products of $n^{1-1/\sigma} \times n^{1-1/\sigma}$ matrices. Communicating these smaller matrices of $n^{2-2/\sigma}$ entries between the nodes takes only $O(n^{1-2/\sigma})$ rounds using Dolev et al. [25] or Lenzen [47].

We refer the reader to the full version of the paper [18] for additional details.

3. UPPER BOUNDS

3.1 Subgraph Detection and Counting

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

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

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

Counting triangles and 4-cycles. For counting triangles, that is, 3-cycles, we use a technique first observed by Itai and Rodeh [43]. That is, in an undirected graph with adjacency matrix A, the number of triangles is known to be $\frac{1}{6}\operatorname{tr}(A^3)$, where the *trace* $\operatorname{tr}(S)$ of a matrix S is the sum of its diagonal entries S_{uu}. Similarly, for directed graphs, the number of triangles is $\frac{1}{3}\operatorname{tr}(A^3)$.

Alon et al. [6] generalise the above formula to counting undirected and directed k-cycles for small k. For example, the number of 4-cycles in an undirected graph is given by

$$\frac{1}{8}\Big[\operatorname{tr}(A^4) - \sum_{v \in V}\Big(2(\deg(v))^2 - \deg(v)\Big)\Big]\,.$$

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

$$\frac{1}{4}\Big[\operatorname{tr}(A^4) - \sum_{v \in V}\Big(2(\delta(v))^2 - \delta(v)\Big)\Big]\,.$$

Combining these observations with Theorem 1, we immediately obtain Corollary 2:

COROLLARY 2. *On directed and undirected graphs, counting triangles and 4-cycles takes $O(n^\rho)$ rounds.*

We note that similar trace formulas exists for counting k-cycles for $k \in \{5, 6, 7\}$, requiring only computation of small powers of A and local information. We omit the detailed discussion of these in the context of the congested clique; see Alon et al. [6] for details.

Detecting k-cycles. For detection of k-cycles we leverage the *colour-coding* techniques of Alon et al. [5] in addition to the matrix multiplication. Again, the distributed algorithm is a straightforward adaptation of a centralised one.

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

LEMMA 11. *Given a graph $G = (V, E)$ and a colouring $c\colon V \to [k]$, a colourful k-cycle can be detected in $O\big(3^k n^\rho\big)$ rounds.*

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

$$C^{(X)} = \bigvee_{\substack{Y \subseteq X \\ |Y| = \lceil |X|/2 \rceil}} C^{(Y)} A C^{(X \setminus Y)}\,, \tag{1}$$

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

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

We can now use Lemma 11 to prove Theorem 3; while we cannot directly construct a suitable colouring from scratch for an uncoloured graph, we can try an exponential in k number of colourings to find a suitable one.

COROLLARY 3. *For any graph, the existence of k-cycles can be detected in $2^{O(k)} n^\rho \log n$ rounds.*

PROOF. To apply Lemma 11, we first have to obtain a colouring $c\colon V \to [k]$ that assigns each colour once to at least one k-cycle in G, assuming that one exists. If we pick a colour $c(v) \in [k]$ for each node uniformly at random, then for any k-cycle C in G, the probability that C is colourful in the colouring c is $k!/k^k > e^{-k}$. Thus, by picking $e^k \log n$ uniformly random colourings and applying Lemma 11 to each of them, we find a k-cycle with high probability if one exists.

This algorithm can also be derandomised using standard techniques. A *k-perfect family of hash functions* $\mathcal{H}$ is a collection of functions $h\colon V \to [k]$ such that for each $U \subseteq V$ with $|U| = k$, there is at least one $h \in \mathcal{H}$ such that h assigns a distinct colour to each node in U. There are known constructions that give such families $\mathcal{H}$ with $|\mathcal{H}| = 2^{O(k)} \log n$ and these can be efficiently constructed [5]; thus, it suffices to take such an $\mathcal{H}$ and apply Lemma 11 for each colouring $h \in \mathcal{H}$. □

3.2 Detecting 4-Cycles

We have seen how to *count* 4-cycles with the help of matrix multiplication in $O(n^\rho)$ rounds. We now show how to *detect* 4-cycles in $O(1)$ rounds. The algorithm does not make direct use of matrix multiplication algorithms. However, the key part of the algorithm can be interpreted as an efficient routine for sparse matrix multiplication, under a specific definition of sparseness.

Let

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

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

On a high level, the algorithm proceeds as follows.

1. Each node x computes $|P(x,*,*)|$. If $|P(x,*,*)| \geq 2n-1$, then there exists some $z \neq x$ such that $|P(x,*,z)| \geq 2$, thus a 4-cycle exists and the algorithm stops.
2. Otherwise, each node x finds $P(x,*,*)$ and checks if there exists some $z \neq x$ satisfying $|P(x,*,z)| \geq 2$.

The first phase is easy to implement in $O(1)$ rounds. The key idea is that if the algorithm does not stop in the first phase, then the total volume of $P(*,*,*)$ is sufficiently small and we can afford to gather $P(x,*,*)$ for each node x in $O(1)$ rounds.

We now present the algorithm in more detail. We write $N(x)$ for the neighbours of node x. To implement the first phase, each node y broadcasts $\deg(y) = |N(y)|$ to all other nodes; we have

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

To find a 4-cycle, each node x broadcasts a single bit indicating whether $|P(x,*,*)| \geq 2n-1$; let x be a specific node fulfilling the inequality, e.g. the one with the smallest identifier. Each node y broadcast a bit indicating if it is a neighbor of x, and now a node z with two common neighbors with x, y and y', can find a cycle (x,y,z,y',x).

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

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

so that each node x will know $P(x,*,*)$. Due to the first phase, we have

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

Using this bound, we obtain the following lemma.

LEMMA 12. *It is possible to find sets $A(y)$ and $B(y)$ for each $y \in V$ such that the following holds:*

- *$A(y) \subseteq V$, $B(y) \subseteq V$, and $|A(y)| = |B(y)| \geq \deg(y)/8$,*
- *the tiles $A(y) \times B(y)$ are disjoint subsets of the square $V \times V$.*

Moreover, this can be done in $O(1)$ rounds in the congested clique.

PROOF. Let $f(y)$ be $\deg(y)/4$ rounded down to the nearest power of 2, and let k be n rounded down to the nearest power of 2. We have $\sum_y f(y)^2 \leq \sum \deg(y)^2/16 < n^2/8 < k^2$. Now it is easy to place the tiles of dimensions $f(y) \times f(y)$ inside a square of dimensions $k \times k$ without any overlap with the following iterative procedure:

- Before step $i = 1, 2, \ldots$, we have partitioned the square into sub-squares of dimensions $k/2^{i-1} \times k/2^{i-1}$, and each sub-square is either completely full or completely empty.
- During step i, we divide each sub-square in 4 parts, and fill empty squares with tiles of dimensions $f(y) = k/2^i$.
- After step i, we have partitioned the square in sub-squares of dimensions $k/2^i \times k/2^i$, and each sub-square is either completely full or completely empty.

This way we have allocated disjoint tiles $A(y) \times B(y) \subseteq [k] \times [k] \subseteq V \times V$ for each y, with $|A(y)| = |B(y)| = f(y) \geq \deg(y)/8$.

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

Now, we use the tiles $A(y) \times B(y)$ to implement the second phase of 4-cycle detection. For convenience, we use the following notation for each $y \in Y$:

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

Note that we can assume that $A(y)$ and $B(y)$ are globally known by Lemma 12. Hence a node can compute $N_A(y,a)$ and $N_B(y,b)$ if it knows $N(y)$.

With this notation, the algorithm proceeds as follows (see Figure 1):

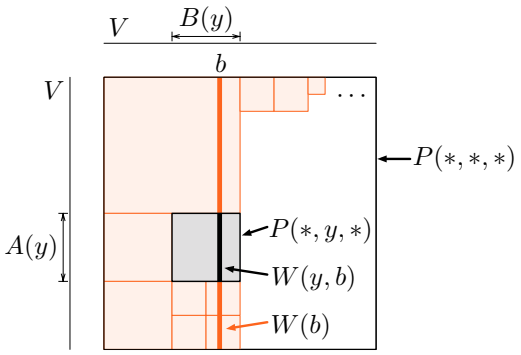

Figure 1: 4-cycle detection: how $P(*,*,*)$ is partitioned among the nodes.

1. For all $y \in V$ and $a \in A(y)$, node y sends $N_A(y,a)$ to a. This step can be implemented in $O(1)$ rounds.
2. For each y and each pair $(a,b) \in A(y) \times B(y)$, node a sends $N_A(y,a)$ to b. Note that for each (a,b) there is at most one y such that $(a,b) \in A(y) \times B(y)$; hence over each edge we send only $O(1)$ messages. Therefore this step can be implemented in $O(1)$ rounds.
3. At this point, each $b \in V$ has received a copy of $N(y)$ for all y with $b \in B(y)$. Node b computes
$$W(y,b) = N(y) \times \{y\} \times N_B(y,b)\,;$$
$$W(b) = \bigcup_{y\colon b \in B(y)} W(y,b)\,.$$
This is local computation; it takes no communication rounds.

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

Lemma 13. *The sets $W(b)$ form a partition of $P(*,*,*)$. For each b we have $|W(b)| = O(n)$.*

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

For the second claim, let Y consist of all $y \in V$ with $b \in B(y)$. As the tiles $A(y) \times B(y)$ are disjoint for all $y \in Y$, and all $y \in Y$ have the common value $b \in B(y)$, it has to hold that the sets $A(y)$ are disjoint subsets of V for all $y \in Y$. Therefore
$$\sum_{y \in Y} |N(y)| = \sum_{y \in Y} \deg(y) \le \sum_{y \in Y} 8|A(y)| \le 8|V| = 8n\,.$$
With $|N_B(y)| \le 8$ we get
$$|W(b)| = \sum_{y \in Y} |W(y,b)| \le 8 \sum_{y \in Y} |N(y)| \le 64n\,,$$
which completes the proof. □

Now we are almost done: we have distributed $P(*,*,*)$ evenly among V so that each node only holds $O(n)$ elements. Finally, we use the dynamic routing scheme [47] to gather $P(x,*,*)$ at each node $x \in V$; each node needs to send and receive $O(n)$ words, and therefore $O(1)$ rounds suffice. If a 4-cycle exists, any node x in it can now detect it from $P(x,*,*)$.

Theorem 4. *On undirected graphs, the existence of 4-cycles can be detected in $O(1)$ rounds. If such a cycle exists, one such cycle can also be reported in $O(1)$ rounds.*

3.3 Girth

Recall that the *girth* g of a graph $G = (V,E)$ is the length of the shortest cycle in G. For *undirected* girth, we leverage the fast cycle detection algorithm and the following lemma giving a trade-off between the girth and the number of edges. A similar approach of bounding from above the number of edges of a graph that contains no copies of some given subgraph was taken by Drucker et al. [26].

Lemma 14 ([54, pp. 362–363]). *A graph with girth g has at most $n^{1+1/\lfloor (g-1)/2 \rfloor} + n$ edges.*

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

Theorem 5. *For undirected graphs, the girth can be computed in $\tilde{O}(n^\rho + n^{o(1)})$ rounds.*

Proof. Assume for now that $\rho > 0$, and fix $\ell = \lceil 2 + 2/\rho \rceil$. Each node collects all graph degrees and computes the total number of edges. If there are at most $n^{1+1/\lfloor \ell/2 \rfloor} + n = O(n^{1+\rho})$ edges, we can collect full information about the graph structure to all nodes in $O(n^\rho)$ rounds using an algorithm of Dolev et al. [25], and each node can then compute the girth locally. Otherwise, by Lemma 14, the graph has girth at most ℓ. Thus, for $k = 3, 4, \ldots, \ell$, we try to find a k-cycle using Corollary 3, in $\ell \cdot 2^{O(\ell)} n^\rho \log n = \tilde{O}(n^\rho)$ rounds. When such a cycle is found for some k, we stop and return k as the girth. Finally, if $\rho = 0$, we pick $\ell = \log\log n$, and both cases take $n^{o(1)}$ rounds. □

For a directed graph, the girth is defined as the length of the shortest directed cycle; the main difference comparing to the undirected case is that directed girth can be 1 or 2. While the trade-off of Lemma 14 cannot be used for directed graphs, we can use a simpler technique of Itai and Rodeh [43].

Let $G = (V,E)$ be a directed graph; we can assume that there are no self-loops in G, as otherwise girth is 1 and we can detect this with local computation. Let $B^{(i)}$ be a Boolean matrix defined as
$$B^{(i)}_{uv} = \begin{cases} 1 & \text{if there is a path from } u \text{ to } v \text{ of length} \le i, \\ 0 & \text{otherwise.} \end{cases}$$
Clearly, we have that $B^{(1)} = A$. Moreover, if $i = j + k$, we have
$$B^{(i)} = \left(B^{(j)} B^{(k)}\right) \vee A\,, \tag{2}$$
where the matrix product is over the Boolean semiring and $\vee$ denotes element-wise logical or.

Corollary 15. *For directed graphs, the girth can be computed in $\tilde{O}(n^\rho)$ rounds.*

Proof. It suffices to find the smallest ℓ such that there is $v \in V$ with $B^{(\ell)}_{vv} = 1$; clearly ℓ is then the girth of graph G. We first compute $B^{(1)} = A, B^{(2)}, B^{(4)}, B^{(8)}, \ldots$ using (2) with $j = k = i/2$ until we find i such that $B^{(i)}_{vv} = 1$ for some

$v \in V$. We then know that the girth is between i and $i/2$; we can perform binary search on this interval to find the girth, using (2) to evaluate the intermediate matrices. This requires $O(\log n)$ calls to the matrix multiplication algorithm. □

3.4 All-pairs Shortest Paths from Matrix Multiplication

Matrix multiplication can be used to compute the shortest path distances via *iterated squaring* of the weight matrix W – entry W_{uv} is the weight of the edge (u, v) or ∞ if the edge does not exist – over the min-plus semiring [28, 33, 57]. That is, the matrix product is the *distance product*, also known as the *min-plus product* or *tropical product*, defined as

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

Given a graph $G = (V, E)$ with weight matrix W, the n^{th} distance product power W^n gives the actual distances in G as $d(v, u) = W^n_{vu}$. Computing W^n can be done with $\lceil \log n \rceil$ distance products by iteratively squaring W, that is, we compute $W^{2i} = W^i \star W^i$, for values of i that are powers of 2. Using this basic idea, we can compute all-pairs shortest path distances in various settings:

- For general weights, we compute the distance product using semiring matrix multiplication (Corollary 7).
- For unweighted and undirected graphs, we use a slightly different recursive technique of Seidel [66], which allows deducing distances in G based on those of the *square graph* G^2 (Corollary 8).
- If the weights are bounded by a small integer M, we can compute the distance product in $O(Mn^\rho)$ rounds by embedding the min-plus semiring into a suitable ring (Corollary 9).
- For approximate APSP, we adapt a lemma of Zwick [76] to show that we can compute a $(1 + \delta)$-approximate distance product in $O(n^\rho/\delta)$ rounds for any $\delta > 0$ (Corollary 10).

These algorithms can also be extended to compute full shortest path information by modifying the distance product algorithms so that they provide *witnesses*; an index w is a witness for uv if $(S \star T)_{uv} = S_{uw} + T_{wv}$. The semiring matrix multiplication can be directly modified to provide witnesses, and for distance product algorithms based on fast matrix multiplication known techniques from the centralised setting can be applied [4, 66, 76]

For more details on the APSP algorithms, see the full version of this paper [18].

4. LOWER BOUNDS

Lower bounds for matrix multiplication implementations. While proving unconditional lower bounds for matrix multiplication in the congested clique model seems to be beyond the reach of current techniques, as discussed in Section 1.5, it can be shown that the results given in Theorem 1 are essentially optimal distributed implementations of the corresponding centralised algorithms. To be more formal, let C be an arithmetic circuit for matrix multiplication; we say that an *implementation* of C in the congested clique model is a mapping of the gates of C to the nodes of the congested clique. This naturally defines a congested clique algorithm for matrix multiplication, with communication cost given by the wires in C between gates assigned to different nodes.

Various authors, considering different parallel models, have shown that in any implementation of the trivial $\Theta(n^3)$ matrix multiplication on a parallel machine with P processors there is at least one processor that has to send or receive $\Omega(n^2/P^{2/3})$ matrix entries [2, 42, 70]. As these models can simulate the congested clique, a similar lower bound holds for congested clique implementations of the trivial $O(n^3)$ matrix multiplication. In the congested clique, each processor sends and receives $\tilde{\Theta}(n)$ messages per round and $P = n$, yielding a lower bound of $\tilde{\Omega}(n^{1/3})$ rounds.

The trivial $\Theta(n^3)$ algorithm is optimal for circuits using only semiring addition and multiplication. The task of $n \times n$ matrix multiplication over the min-plus semiring can be reduced to APSP with a constant blowup [3, pp. 202–205], hence the above bound applies also to any APSP algorithm that only uses minimum and addition operations. Thus, current techniques for similar problems, like the one used in the fast MST algorithm of Lotker et al. [52], cannot be extended to solve APSP.

COROLLARY 16. *Any implementation of the trivial $\Theta(n^3)$ matrix multiplication and any APSP algorithm which only sums weights and takes the minimum of such sums require $\tilde{\Omega}(n^{1/3})$ communication rounds in the congested clique model.*

However, known results on centralised APSP and distance product computation give reasons to suspect that this bound can be broken if we allow subtraction; in particular, translating the recent result of Williams [74] might allow for running time of order $n^{1/3}/2^{\Omega(\sqrt{\log n})}$ for APSP in the congested clique.

Concerning fast matrix multiplication algorithms, Ballard et al. [8] have proven lower bounds for parallel implementations of *Strassen-like* algorithms. Their seminal work is based on building a DAG representing the linear combinations of the inputs before the block multiplications, and the linear combinations of the results of the multiplications ("decoding") as the output matrix. The parallel computation induces an assignment of the graph vertices to the processes, and the edges crossing the partition represent the communication. Using an expansion argument, Ballard et al. show that in any partition a graph representing an $\Omega(n^\sigma)$ algorithm there is a process communicating $\Omega(n^{2-2/\sigma})$ values. See also [9] for a concise account of the technique.

The lower bound holds for Strassen's algorithm, and for a family of similar algorithms, but not for any matrix multiplication algorithm [8, §5.1.1]. A matrix multiplication algorithm is said to be *Strassen-like* if it is recursive, its decoding graph discussed above is connected, and it computes no scalar multiplication twice. As each process communicates at most $O(n)$ values in a round, the implementation of an $\Omega(n^\sigma)$ strassen-like algorithm must take $\Omega(n^{1-2/\sigma})$ rounds.

COROLLARY 17. *Any implementation of a Strassen-like matrix multiplication algorithm using $\Omega(n^\sigma)$ element multiplications requires $\tilde{\Omega}(n^{1-2/\sigma})$ communication rounds in the congested clique model.*

Lower bound for broadcast congested clique. Recall that the broadcast congested clique is a version of the congested

clique model with the additional constraint that all $n-1$ messages sent by a node in a round must be identical.

Holzer and Pinsker [39] show that computing any approximation better than factor 2 to all-pairs shortest paths in weighted graphs takes $\tilde{\Omega}(n)$ rounds in the congested clique. As discussed in Section 3.4, $\tilde{o}(n)$-round matrix multiplication algorithms imply $\tilde{o}(n)$-round algorithms for $(1+o(1))$-approximate weighted APSP.

COROLLARY 18. *In the broadcast congested clique model, matrix multiplication over the Boolean semiring and APSP algorithms require $\tilde{\Omega}(n)$ communication rounds.*

We remark that the restriction "over the Boolean semiring" refers to the issue that, in principle, it is possible that matrix multiplication exponents may be different for different underlying semirings. However, at the very least the lower bound also applies to matrix multiplication over integers and rationals, as well as the min-plus semiring, as in all cases a Boolean matrix multiplication algorithm trivially follows. We stress that this bound holds without any assumptions on the algorithm itself.

5. CONCLUSIONS

In this work, we demonstrate that algebraic methods – especially fast matrix multiplication – can be used to design efficient algorithms in the congested clique model, resulting in solutions that outperform all previous combinatorial algorithms. Moreover, we have certainly not exhausted the known centralised literature of algorithms based on matrix multiplication, so similar techniques should also give improvements for other problems. It remains open whether corresponding unconditional lower bounds exist; however, suspicion grows that these would imply strong lower bounds for centralised algorithms, and will thus be highly challenging to prove.

While the present work focuses on a fully connected communication topology (clique), we expect that the same techniques can be applied more generally in the usual CONGEST model. For example, fast triangle detection in the CONGEST model is trivial in those areas of the network that are sparse. Only dense areas of the network are non-trivial, and in those areas we may have enough overall bandwidth for fast matrix multiplication algorithms. On the other hand, there are non-trivial lower bounds for distance computation problems in the CONGEST model [24], though significant gaps still remain [58].

Acknowledgements

We thank Morteza Zadimoghaddam for pointing out that our 4-cycle detection algorithm can also be used to report a cycle, and we thank Keijo Heljanko, Juho Hirvonen, Fabian Kuhn, Tuomo Lempiäinen, and Joel Rybicki for discussions.

References

[1] Ramesh C. Agarwal, Susanne M. Balle, Fred G. Gustavson, Mahesh V. Joshi, and Prasad V. Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39(5):575–582, 1995.

[2] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71(1):3–28, 1990.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[4] Noga Alon and Moni Naor. Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4–5):434–449, 1996.

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

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

[7] Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *Proc. 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2012)*, pages 193–204, 2012.

[8] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM*, 59 (6):32, 2012.

[9] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Communication costs of strassen's matrix multiplication. *Commun. ACM*, 57(2):107–114, 2014.

[10] Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM Journal on Computing*, 43(1): 280–299, 2014.

[11] Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. In *Proc. 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, pages 211–222, 2014.

[12] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 67–74, 2007.

[13] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.

[14] Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Probably optimal graph motifs. In *Proc. 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, pages 20–31, 2013.

[15] Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Counting thin subgraphs via packings faster than meet-in-the-middle time. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 594–603, 2014.

[16] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP 2013)*, pages 196–207, 2013.

[17] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.

[18] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela.

Algebraic methods in the congested clique, 2015. arXiv:1503.04963 [cs.DC].

[19] Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2): 236–243, 2008.

[20] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

[21] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proc. 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011)*, pages 150–159, 2011.

[22] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. In *Proc. 45th ACM Symposium on Theory of Computing (STOC 2013)*, pages 301–310, 2013.

[23] Artur Czumaj and Andrzej Lingas. Finding a heaviest triangle is not harder than matrix multiplication. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, pages 986–994, 2007.

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

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

[26] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proc. 33rd ACM Symposium on Principles of Distributed Computing (PODC 2014)*, pages 367–376, 2014.

[27] Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1–3):57–67, 2004.

[28] M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *Proc. 12th Symposium on Switching and Automata Theory (FOCS 1971)*, pages 129–131, 1971.

[29] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B. V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *Journal of Computer and System Sciences*, 78(3):698–706, 2012.

[30] Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 142–151, 2014.

[31] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5(1):83–89, 1976.

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

[33] M. E. Furman. Application of a method of fast multiplication of matrices to problem of finding graph transitive closure. *Doklady Akademii Nauk SSSR*, 194 (3):524, 1970.

[34] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th Symposium on Symbolic and Algebraic Computation (ISSAC 2014)*, pages 296–303, 2014.

[35] Yijie Han. An $O(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest path. *Algorithmica*, 51 (4):428–434, 2008.

[36] Yijie Han and Tadao Takaoka. An $O(n^3\log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. In *Proc. 13th Scandinavian Symposium on Algorithm Theory (SWAT 2012)*, pages 131–141, 2012.

[37] James W. Hegeman and Sriram V. Pemmaraju. Lessons from the congested clique applied to mapreduce. In *Proc. 21st Colloquium on Structural Information and Communication Complexity (SIROCCO 2014)*, pages 149–164, 2014.

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

[39] Stephan Holzer and N. Pinsker. Approximation of distances and shortest paths in the broadcast congest clique, 2014. arXiv:1412.3445 [cs.DC].

[40] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proc. 31st ACM Symposium on Principles of Distributed Computing (PODC 2012)*, pages 355–364, 2012.

[41] Stephan Holzer, David Peleg, Liam Roditty, and Roger Wattenhofer. Brief announcement: Distributed 3/2-approximation of the diameter. In *Proc. 28th International Symposium on Distributed Computing (DISC 2014)*, pages 562–564, 2014.

[42] Dror Irony, Sivan Toledo, and Alexandre Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

[43] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4): 413–423, 1978.

[44] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, pages 575–586, 2008.

[45] Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations and matrix multiplication. In *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 1468–1476, 2011.

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

[47] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proc. 32nd ACM Symposium on Principles of Distributed Computing (PODC 2013)*, pages 42–50, 2013.

[48] Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages. In *Proc. 45rd ACM Symposium on Theory of Computing (STOC 2013)*, pages 381–390, 2013.

[49] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *Proc. 32nd ACM Symposium on Principles of Distributed Computing (PODC 2013)*, pages 375–382, 2013.

[50] Christoph Lenzen and Roger Wattenhofer. Tight bounds for parallel randomized load balancing. In *Proc. 43rd ACM Symposium on Theory of Computing (STOC 2011)*, pages 11–20, 2011.

[51] Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC 2010)*, 2010.

[52] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-Weight Spanning Tree Construction in $O(\log \log n)$ Communication Rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.

[53] Qingshan Luo and John B. Drake. A scalable parallel Strassen's matrix multiplication algorithm for distributed-memory computers. In *Symposium on Applied Computing (SAC)*, pages 221–226, 1995.

[54] Jirí Matoušek. *Lectures on Discrete Geometry*. Graduate Texts in Mathematics. Springer, 2002.

[55] William F. McColl. Scalable computing. In *Computer Science Today*, pages 46–61. Springer, 1995.

[56] William F. McColl. A BSP realisation of Strassen's algorithm. In *Abstract Machine Models for Parallel and Distributed Computing*, volume 48 of *Concurrent Systems Engineering*, pages 43–46. IOS Press, 1996.

[57] Ian Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1(2):56–58, 1971.

[58] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. 46th ACM Symposium on Theory of Computing (STOC 2014)*, pages 565–573, 2014.

[59] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[60] Boaz Patt-Shamir and Marat Teplitsky. The round complexity of distributed sorting. In *Proc. 30th ACM Symposium on Principles of Distributed Computing (PODC 2011)*, pages 249–256, 2011.

[61] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

[62] David Peleg and Vitaly Rubinovich. Near-tight lower bound on the time complexity of distributed MST construction. *SIAM Journal on Computing*, 30(5): 1427–1442, 2000.

[63] David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In *Proc. 39th International Colloquium on Automata, Languages and Programming (ICALP 2012)*, pages 660–672, 2012.

[64] Sriram V. Pemmaraju and Vivek B. Sardeshmukh. Minimum-weight Spanning Tree Construction in $O(\log \log \log n)$ Rounds on the Congested Clique, 2014. arXiv:1412.2333 [cs.DC].

[65] P Raghavan and C D Thompson. Provably Good Routing in Graphs: Regular Arrays. In *Proc. 7th ACM Symposium on Theory of Computing (STOC 1985)*, pages 79–87, 1985.

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

[67] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[68] Tadao Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *Proc. 10th International Conference on Computing and Combinatorics (COCOON 2004)*, pages 278–289, 2004.

[69] Tadao Takaoka. An $O(n^3 \log \log n / \log n)$ time algorithm for the all-pairs shortest path problem. *Information Processing Letters*, 96(5):155–161, 2005.

[70] Alexandre Tiskin. Bulk-synchronous parallel multiplication of boolean matrices. In *Proc. 25th Colloquium on Automata, Languages and Programming (ICALP 1998)*, pages 494–506. Springer Berlin Heidelberg, 1998.

[71] Alexandre Tiskin. *The Design and Analysis of Bulk-Synchronous Parallel Algorithms*. PhD thesis, University of Oxford, 1999.

[72] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal of Computing*, 42(3):831–854, 2013.

[73] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009.

[74] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proc. 46th ACM Symposium on Theory of Computing (STOC 2014)*, pages 664–673. ACM, 2014.

[75] Uri Zwick. Exact and approximate distances in graphs – A survey. In *Proc. 9th European Symposium on Algorithms (ESA 2001)*, pages 33–48, 2001.

[76] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.

[77] Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. *Algorithmica*, 46(2):181–192, 2006.