

Algebraic Methods in the Congested Clique

Keren Censor-Hillel* Petteri Kaski† Janne H. Korhonen‡
 Christoph Lenzen§ Ami Paz* Jukka Suomela†

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 $O(n^{1/3})$ 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.

*Department of Computer Science, Technion, {ckeren, amipaz}@cs.technion.ac.il.

†Helsinki Institute for Information Technology HIIT & Department of Information and Computer Science, Aalto University, {petteri.kaski, jukka.suomela}@aalto.fi.

‡Helsinki Institute for Information Technology HIIT & Department of Computer Science, University of Helsinki, janne.h.korhonen@helsinki.fi.

§Department for Algorithms and Complexity, MPI Saarbrücken, clenzen@mpi-inf.mpg.de.

1 Introduction

Algebraic methods have become a recurrent tool in centralised algorithmics, employing a wide range of techniques (e.g., [10–16, 20–22, 26, 28, 29, 43, 44, 50, 58, 71, 72]). 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 a *fully-connected* synchronous network; initially, each node is aware of its neighbours in G . In comparison with the traditional CONGEST model [60], 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 [24, 25, 36, 37, 46, 49, 51, 57, 59, 63], 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 [33]. 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.

Problem	Running time	
	This work	Prior work
matrix multiplication (semiring)	$O(n^{1/3})$	—
matrix multiplication (ring)	$O(n^{0.158})$	$O(n^{0.373})$ [25]
triangle counting	$O(n^{0.158})$	$O(n^{1/3} / \log n)$ [24]
4-cycle detection	$O(1)$	$O(n^{1/2} / \log n)$ [24]
4-cycle counting	$O(n^{0.158})$	$O(n^{1/2} / \log n)$ [24]
k -cycle detection	$2^{O(k)} n^{0.158}$	$O(n^{1-2/k} / \log n)$ [24]
girth	$O(n^{0.158})$	—
weighted, directed APSP	$O(n^{1/3} \log n)$	—
· weighted diameter U	$O(Un^{0.158})$	—
· $(1 + o(1))$ -approximation	$O(n^{0.158})$	—
· $(2 + o(1))$ -approximation		$\tilde{O}(n^{1/2})$ [57]
unweighted, undirected APSP	$O(n^{0.158})$	—
· $(2 + o(1))$ -approximation		$\tilde{O}(n^{1/2})$ [57]

Table 1: Our results versus prior work, for the currently best known bound $\omega < 2.3729$ [33]; \tilde{O} notation hides polylogarithmic factors.

1.1 Matrix Multiplication on a Congested Clique

As a basic primitive, we consider the computation of the product $P = ST$ of two $n \times n$ matrices S and T 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 each node 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 product of two $n \times n$ matrices can be computed with $O(n^\sigma)$ arithmetic operations; it is known that $2 \leq \omega < 2.3728639$ [33], and it is conjectured, though not unanimously, that $\omega = 2$.

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

Theorem 1 follows by adapting known parallel matrix multiplication algorithms for semirings [1, 54] and rings [7, 52, 55, 70] to the clique model, via the routing technique of Lenzen [46]. 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. [24] suffice for implementing these matrix multiplication 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 with 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 $\rho \leq \omega - 2$ [25].

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

Lower bounds for matrix multiplication. The matrix multiplication results are optimal in the sense that for any sequential matrix multiplication implementation, any scheme for simulating that implementation in the congested clique cannot give a faster algorithm than the construction underlying Theorem 1; this follows from known results for parallel matrix multiplication [2, 8, 41, 69]. 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 [38] imply that matrix multiplication cannot be done faster than $\tilde{\Omega}(n)$ rounds.

1.2 Applications in Subgraph Detection

Cycle detection and counting. Our first application of fast matrix multiplication is to the problems of triangle counting [42] and 4-cycle counting.

Corollary 2. *For directed and undirected graphs, the number of triangles and 4-cycles can be computed in $O(n^\rho)$ rounds.*

For $\rho \leq 1 - 2/\omega$, this is an improvement upon the previously best known $O(n^{1/3})$ -round triangle detection algorithm of Dolev et al. [24] and an $O(n^{\omega-2+\varepsilon})$ -round algorithm of Drucker et al. [25]. Indeed, we disprove the conjecture of Dolev et al. [24] that any deterministic oblivious algorithm for detecting triangles requires $\tilde{\Omega}(n^{1/3})$ rounds.

When only detection of cycles is required, we observe that combining the fast distributed matrix multiplication with the well-known technique of *colour-coding* [5] allows us to detect k -cycles in $\tilde{O}(n^\rho)$ rounds for any constant k . This improves upon the subgraph detection algorithm of Dolev et al. [24], which requires $\tilde{O}(n^{1-2/k})$ rounds for detecting subgraphs of k nodes. However, we do not improve upon the algorithm of Dolev et al. for general subgraph detection.

Theorem 3. *For directed and undirected graphs, the existence of k -cycles can be detected in $2^{O(k)}n^\rho \log n$ rounds.*

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

Theorem 4. *The existence of 4-cycles can be detected in $O(1)$ rounds.*

Girth. We compute the girth of a graph by leveraging a known trade-off between the girth and the number of edges of the graph [53]. 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, unweighted graphs, the girth can be computed in $\tilde{O}(n^\rho)$ rounds.*

1.3 Applications in Distance Computation

Shortest paths. The *all-pairs shortest paths* problem (APSP) likewise admits algorithms based on matrix multiplication. The basic idea is to compute the n^{th} power of the input graph's weight matrix over the min-plus semiring, by iteratively computing squares of the matrix [27, 32, 56].

Corollary 6. *For weighted, directed graphs with integer weights in $\{0, \pm 1, \dots, \pm M\}$, all-pairs shortest paths can be computed in $O(n^{1/3} \log n \lceil \log M / \log n \rceil)$ communication rounds.*

We can leverage fast ring matrix multiplication to improve upon the above result; however, the use of ring matrix multiplication necessitates some trade-offs or extra assumptions. For example, for unweighted and undirected graphs, it is possible to recover the exact shortest paths from powers of the adjacency matrix over the Boolean semiring [65].

Corollary 7. *For undirected, unweighted graphs, all-pairs shortest paths can be computed in $\tilde{O}(n^\rho)$ rounds.*

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

Corollary 8. *For directed graphs with positive integer weights and weighted diameter U , all-pairs shortest paths can be computed in $\tilde{O}(Un^\rho)$ rounds.*

While this corollary is only relevant for graphs of small weighted diameter, the same idea can be combined with weight rounding [57, 64, 76] to obtain a fast approximate APSP algorithm without such limitations.

Theorem 9. *For directed graphs with integer weights in $\{0, 1, \dots, 2^{n^{o(1)}}\}$, we can compute $(1 + o(1))$ -approximate all-pairs shortest paths in $O(n^{\rho+o(1)})$ rounds.*

For comparison, the previously best known combinatorial algorithm for APSP on the congested clique achieves a $(2 + o(1))$ -approximation in $\tilde{O}(n^{1/2})$ rounds [57].

1.4 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 [18, 19, 30, 34, 35, 67, 68, 73, 75–77].

In particular, computing graph distances is vital for many distributed applications and, as such, has been widely studied in the CONGEST model of computation [60], 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 [23, 31, 39, 40, 45, 47, 48, 57, 61, 62]. Some lower bounds apply even for graphs of small diameter; however, these lower bound constructions boil down to graphs that contain *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. [51] shows that a minimum spanning tree (MST) can be computed in $O(\log \log n)$ rounds. Hegeman et al. [37] 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 [46, 49, 59]. A connection between the congested clique model and the MapReduce model is discussed by Hegeman and Pemmaraju [36], where algorithms are given for colouring problems. On top of these positive results, Drucker et al. [25] 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. [24], and if $\omega > 2$, the solution presented in this paper is faster.

2 Matrix Multiplication Algorithms

In this section, we consider computing the product $P = ST$ of two $n \times n$ matrices $S = (S_{ij})$ and $T = (T_{ij})$ on the congested clique with n nodes. For convenience, we tacitly assume that nodes $v \in V$ are identified with $\{1, 2, \dots, n\}$, and use nodes $v \in V$ to directly index the matrices. The local input in the matrix multiplication task for each node $v \in V$ is the row v of both S and T , and at the end of the computation each node $v \in V$ will output the row v of P . However, we note that the exact distribution of the input and output is not important, as we can re-arrange the entries in constant rounds as long as each node has $O(n)$ entries [46].

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

Theorem 1 follows directly by simulating known parallel matrix multiplication algorithms in the congested clique model using a result of [46]. This work discusses simulation of the *bulk-synchronous parallel* (BSP) model, which we can use to obtain Theorem 1 as a corollary from known BSP matrix multiplication results [54, 55, 70]. However, essentially the same matrix multiplication algorithms have been widely studied in various parallel computation models, and the routing scheme underlying the simulation result of [46] allows also simulation of these other models on the congested clique:

- The first part of Theorem 1 is based on the so-called parallel 3D matrix multiplication algorithm [1, 54], essentially a parallel implementation of the school-book matrix multiplication; alternatively, the same algorithm can be obtained by slightly modifying the triangle counting algorithm of Dolev et al. [24].
- The second part uses a scheme that allows one to adapt any bilinear matrix multiplication algorithm into a fast parallel matrix multiplication algorithm [7, 52, 55, 70].

A more detailed examination in fact shows that the matrix multiplication algorithms are *oblivious*, that is, the communication pattern is pre-defined and only the content of the messages depends on the input. This further allows us to use the static routing scheme of Dolev et al. [24], resulting in simpler algorithms with smaller constant factors in the running time.

To account for all the details, and to provide an easy access for readers not familiar with the parallel computing literature, we present the congested clique versions of these algorithms in full detail in Sections 2.1 and 2.2.

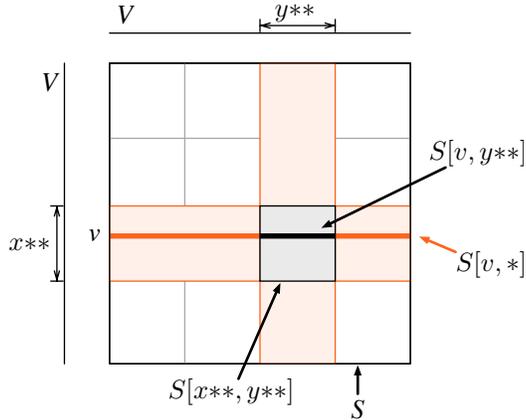


Figure 1: Semiring matrix multiplication: partitioning scheme for matrix entries.

2.1 Semiring matrix multiplication

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

For a matrix S and index sets $U, W \subseteq V$, use the notation $S[U, W]$ to refer to the submatrix obtained by taking all rows u with $u \in U$ and columns w with $w \in W$. To easily refer to specific subsets of indices, we use $*$ as a wild-card in this notation; specifically, we use notation $x** = \{v: v_1 = x\}$, $*x* = \{v: v_2 = x\}$ and $**x = \{v: v_3 = x\}$. Finally, in conjunction with this notation, we use the shorthand $*$ to denote the whole index set V and v to refer to a singleton set $\{v\}$. See Figure 1.

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

$$P_{uw} = S_{uv}T_{vw}, \quad u, v, w \in V$$

can be viewed as points in the cube $V \times V \times V$. To split the element-wise multiplications equally among the nodes, we partition this cube into n subcubes of size $n^{2/3} \times n^{2/3} \times n^{2/3}$. Specifically, each node v is assigned the subcube $v_1** \times v_2** \times v_3**$, corresponding to the multiplication task

$$S[v_1**, v_2**]T[v_2**, v_3**].$$

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

$$P^{(v_2)}[v_1**, v_3**] = S[v_1**, v_2**]T[v_2**, v_3**].$$

Specifically, this is done as follows.

Step 1: Distributing the entries. Each node $v \in V$ sends, for each node $u \in v_1^{**}$, the submatrix $S[v, u_2^{**}]$ to node u , and for each node $w \in *v_2^{**}$, the submatrix $T[v, w_3^{**}]$ to w . Each such submatrix has size $n^{2/3}$ and there are $2n^{2/3}$ recipients, for a total of $2n^{4/3}$ messages per node.

Dually, each node $v \in V$ receives the submatrix $S[v_1^{**}, v_2^{**}]$ and the submatrix $T[v_2^{**}, v_3^{**}]$. In particular, the submatrix $S[u, v_2^{**}]$ is received from the node u for $u \in v_1^{**}$, and the submatrix $T[w, v_3^{**}]$ is received from the node $w \in *v_2^{**}$. In total, each node receives $2n^{4/3}$ messages.

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

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

Dually, each node $v \in V$ receives the submatrices $P^{(w)}[v, *]$ for each $w \in [n^{1/3}]$. In particular, the submatrix $P^{(u_2)}[v, u_3^{**}]$ is received from the node $u \in v_1^{**}$. The total number of received messages is $n^{4/3}$ per node.

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

Analysis. The maximal number of messages sent or received in one of the above steps is $O(n^{4/3})$. Moreover, the communication pattern clearly does not depend on the input matrices, so the algorithm can be implemented in oblivious way on the congested clique using the routing scheme of Dolev et al. [24, Lemma 1]; the running time is $O(n^{1/3})$ rounds.

2.2 Fast Matrix Multiplication

Bilinear matrix multiplication. Consider a *bilinear algorithm* multiplying two $d \times d$ matrices using $m < d^3$ scalar multiplications, such as the Strassen algorithm [66]. Such an algorithm computes the matrix product $P = ST$ by first computing m linear combinations of entries of both matrices,

$$\hat{S}^{(w)} = \sum_{(i,j) \in [d]^2} \alpha_{ijw} S_{ij} \quad \text{and} \quad \hat{T}^{(w)} = \sum_{(i,j) \in [d]^2} \beta_{ijw} T_{ij} \quad (1)$$

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

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

where α_{ijw} , β_{ijw} and λ_{ijw} are scalar constants that define the algorithm. In this section we show that any bilinear matrix multiplication algorithm can be efficiently translated to the congested clique model.

Lemma 10. *Let R be a ring, and assume there exists a family of bilinear matrix multiplication algorithms that can compute product of $n \times n$ matrices with $O(n^\sigma)$ multiplications. Then matrix multiplication over R can be computed in the congested clique in $O(n^{1-2/\sigma}(b/\log n))$ rounds, where b is the number of bits required for encoding a single element of R .*

In particular for integers, rationals and their extensions, it is known that for any constant $\varepsilon > 0$ there is a bilinear algorithm for matrix multiplication that uses $O(n^{\omega+\varepsilon})$ multiplications [17]; thus, the second part of Theorem 1 follows from the above lemma.

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

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

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

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

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

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

Algorithm description. The algorithm computes the matrix product $P = ST$ as follows.

Step 1: Distributing the entries. Each node v sends, for $x_2 \in [n^{1/2}]$, the submatrices $S[v, *x_2*]$ and $T[v, *x_2*]$ to the node u with label $\ell(u) = v_2x_2$. Each submatrix has $n^{1/2}$ entries and there are $n^{1/2}$ recipients each receiving two submatrices, for a total of $2n$ messages per node.

Dually, each node u with label $\ell(u) = x_1x_2$ receives the submatrices $S[v, *x_2*]$ and $T[v, *x_2*]$ from the nodes $v = v_1v_2v_3$ with $v_2 = x_1$. In particular, node u now has the submatrices $S[*x_1*, *x_2*]$ and $T[*x_1*, *x_2*]$. The total number of received messages is $2n$ per node.

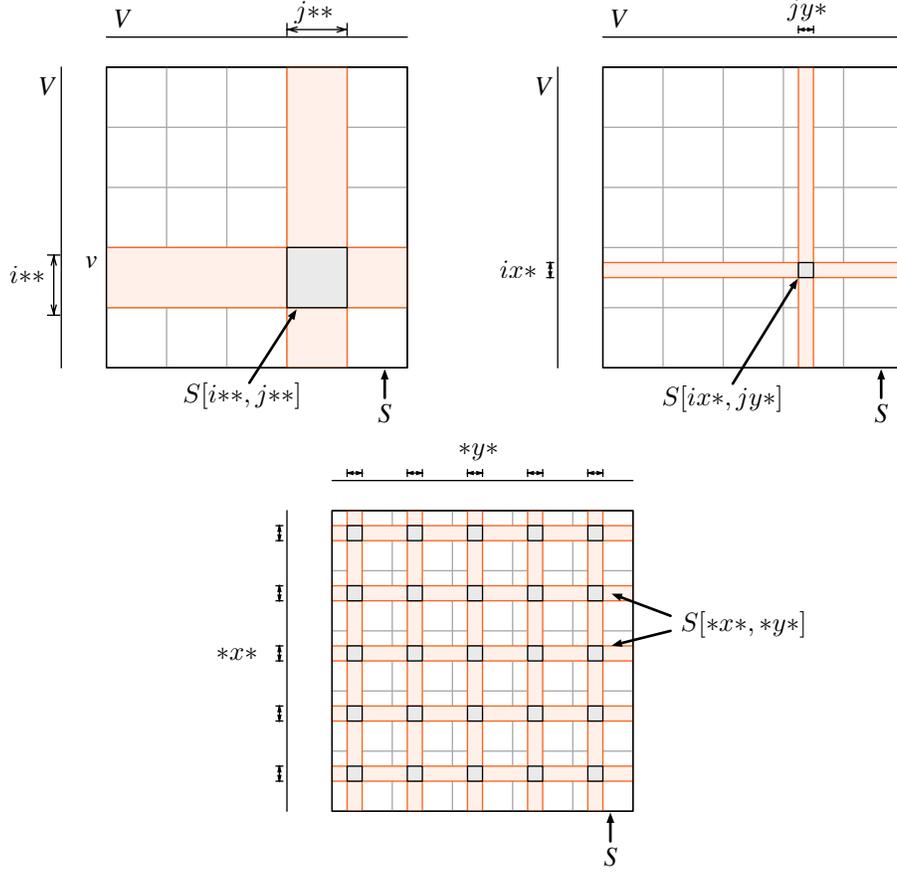


Figure 2: Fast matrix multiplication: partitioning schemes for matrix entries.

Step 2: Linear combination of entries. Each node v with label $\ell(v) = x_1x_2$ computes for $w \in V$ the linear combinations

$$\hat{S}^{(w)}[x_1^*, x_2^*] = \sum_{(i,j) \in [d]^2} \alpha_{ijw} S[ix_1^*, jy_2^*], \quad \text{and}$$

$$\hat{T}^{(w)}[x_1^*, x_2^*] = \sum_{(i,j) \in [d]^2} \beta_{ijw} T[ix_1^*, jy_2^*].$$

The computation is performed entirely locally.

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

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

Step 4: Multiplication. Node $w \in V$ computes the product $\hat{P}^{(w)} = \hat{S}^{(w)}\hat{T}^{(w)}$. The computation is performed entirely locally.

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

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

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

$$P[ix_1*, jx_2*] = \sum_{w \in V} \lambda_{ijw} \hat{P}^{(w)}[x_1*, x_2*].$$

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

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

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

Analysis. The maximal number of messages sent or received by a node in the above steps is $O(n^{2-2/\sigma})$. Moreover, the communication pattern clearly does not depend on the input matrices, so the algorithm can be implemented in an oblivious way on the congested clique using the routing scheme of Dolev et al. [24, Lemma 1]; the running time is $O(n^{1-2/\sigma})$ rounds.

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 [42]. That is, in an undirected graph with adjacency matrix A , the number of triangles is known to be $\frac{1}{6} \text{tr}(A^3)$, where the *trace* $\text{tr}(S)$ of a matrix S is the sum of its diagonal entries S_{uu} . Similarly, for directed graphs, the number of triangles is $\frac{1}{3} \text{tr}(A^3)$.

Alon et al. [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

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

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

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

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

Corollary 2. *For directed and undirected graphs, the number of triangles and 4-cycles can be computed in $O(n^\rho)$ rounds.*

We note that similar trace formulas exist for counting k -cycles for $k \in \{5, 6, 7\}$, requiring only computation of small powers of A and local information. We omit the detailed discussion of these in the context of the congested clique; see Alon et al. [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 $c: V \rightarrow [k]$ 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 $c: V \rightarrow [k]$, a colourful k -cycle can be detected in $O(3^k n^\rho)$ rounds.*

Proof. For each subset of colours $X \subseteq [k]$, let $C^{(X)}$ be a Boolean matrix such that $C_{uv}^{(X)} = 1$ if there is a path of length $|X| - 1$ from u to v containing exactly one node of each colour from X , and $C_{uv}^{(X)} = 0$ 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

$$C^{(X)} = \bigvee_{\substack{Y \subseteq X \\ |Y| = \lceil |X|/2 \rceil}} C^{(Y)} A C^{(X \setminus Y)}, \quad (3)$$

where the products are computed over the Boolean semiring and \vee denotes element-wise logical or. Thus, we can compute $C^{(X)}$ for all $X \subseteq [k]$ by applying (3) 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 $C_{uv}^{([k])} = 1$ and $(v, u) \in E$.

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

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.

Theorem 3. *For directed and undirected graphs, 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 $c: V \rightarrow [k]$ 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 $h: V \rightarrow [k]$ 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 $h \in \mathcal{H}$. \square

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

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

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

On a high level, the algorithm proceeds as follows.

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

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

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

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

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

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

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

In the second phase, we have

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

Using this bound, we obtain the following lemma.

Lemma 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 $i = 1, 2, \dots$, we have partitioned the square in sub-squares of dimensions $k/2^{i-1} \times k/2^{i-1}$, and each sub-square is either completely full or completely empty.
- 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.

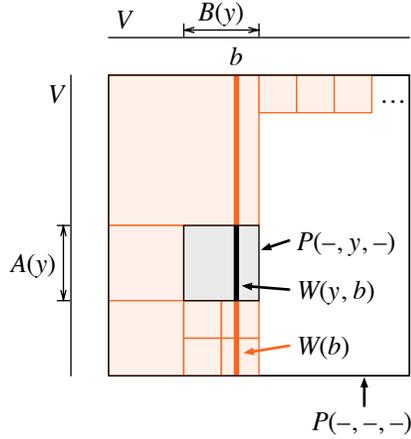


Figure 3: 4-cycle detection: how $P(*, *, *)$ is partitioned among the nodes.

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. \square

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

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

Note that we can assume that $A(y)$ and $B(y)$ are globally known by Lemma 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 3):

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 $O(1)$ words. 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

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

This is local computation; it takes 0 rounds.

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

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

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

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

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

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

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

Now we are almost done: we have distributed $P(*, *, *)$ evenly among V so that each node only holds $O(n)$ elements. Finally, we use the dynamic routing scheme [46] to gather $P(x, *, *)$ at each node $x \in V$; here each node needs to send $O(n)$ words and receive $O(n)$ words, and the running time is therefore $O(1)$ rounds. In conclusion, we can implement both phases of 4-cycle detection in $O(1)$ rounds.

Theorem 4. *The existence of 4-cycles can be detected in $O(1)$ rounds.*

3.2 Girth

Undirected girth. Recall that the *girth* g of an undirected unweighted graph $G = (V, E)$ is the length of the shortest cycle in G . To compute the girth in the congested clique model, we leverage the fast cycle detection algorithm and the following lemma giving a trade-off between the girth and the number of edges. A similar approach of bounding from above the number of edges of a graph that contains no copies of some given subgraph was taken by Drucker et al. [25].

Lemma 14 ([53, 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 graph structure.

Theorem 15. *For undirected graphs, the girth can be computed in $\tilde{O}(n^\rho)$ rounds (or in $n^{o(1)}$ rounds, if $\rho = 0$).*

Proof. Assume for now that $\rho > 0$, and fix $\ell = \lceil 2 + 2/\rho \rceil$. Each node collects 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. [24], and each node can then compute the girth locally.

Otherwise, by Lemma 14, the graph has girth at most ℓ . Thus, for $k = 3, 4, \dots, \ell$, we try to find a k -cycle using Theorem 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. \square

Directed girth. For a directed graph, the girth is defined as the length of the shortest directed cycle; the main difference is that directed girth can be 1 or 2. While the trade-off of Lemma 14 cannot be used for directed graphs, we can use a simpler technique of Itai and Rodeh [42].

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

$$B_{uv}^{(i)} = \begin{cases} 1 & \text{if there is a path of length } \ell \text{ from } u \text{ to } v \text{ for } 1 \leq \ell \leq i, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, we have that $B^{(1)} = A$. Moreover, if $i = j + k$, we have

$$B^{(i)} = (B^{(j)} B^{(k)}) \vee A, \quad (4)$$

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

Corollary 16. *For directed graphs, the girth can be computed in $\tilde{O}(n^\rho)$ rounds.*

Proof. It suffices to find smallest ℓ such that there is $v \in V$ with $B_{vv}^{(\ell)} = 1$; clearly ℓ is then the girth of graph G . We first compute $A = B^{(1)}, B^{(2)}, B^{(4)}, B^{(8)}, \dots$ using (4) with $j = k = i/2$ until we find i such that $B_{vv}^{(i)} = 1$ 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 (4) to evaluate the intermediate matrices. This requires $O(\log n)$ calls to the matrix multiplication algorithm. \square

3.3 Routing and Shortest Paths

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

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

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

Our APSP algorithms are mostly based on the manipulation of the weight matrix W and the adjacency matrix A , as defined in Section 3.1.

Distance product and iterated squaring. Matrix multiplication can be used to compute the shortest path distances via *iterated squaring* of the weight matrix over the min-plus semiring [27, 32, 56]. That is, the matrix product is the *distance product*, also known as the *min-plus product* or *tropical product*, defined as

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

Given a graph $G = (V, E)$ with weight matrix W , the n^{th} distance product power W^n gives the actual distances in G as $d(v, u) = W_{vu}^n$. Computing W^n can be done with $\lceil \log n \rceil$ distance products by iteratively squaring W , that is, we compute

$$W^2 = W \star W, \quad W^4 = W^2 \star W^2, \quad \dots, \quad W^n = W^{n/2} \star W^{n/2}.$$

Combining this observation with the semiring algorithm from Theorem 1, we immediately obtain a simple APSP algorithm for the congested clique.

Corollary 6. *For weighted, directed graphs with integer weights in $\{0, \pm 1, \dots, \pm M\}$, all-pairs shortest paths can be computed in $O(n^{1/3} \log n \lceil \log M / \log n \rceil)$ communication rounds.*

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

Constructing routing tables. The iterated squaring algorithm of Corollary 6 can be adapted to also compute a routing table R as follows. Assume that our distance product algorithm also provides for the distance product $S \star T$ a *witness matrix* Q such that if $Q_{uv} = w$, then $(S \star T)_{uv} = S_{uw} + T_{wv}$. With this information, we can compute the routing table R during the iterated squaring algorithm; when we compute the product $W^{2i} = W^i \star W^i$, we also obtain a witness matrix Q , and update the routing table by setting

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

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

The semiring matrix multiplication can be easily modified to produce witnesses, but for the subsequent distance product algorithms based on fast matrix multiplication this is not directly possible. However, we can apply known techniques from the centralised setting to obtain witnesses also in these cases [4, 65, 76]; we refer to Section 3.4 for details.

Unweighted undirected APSP. In the case of unweighted undirected graphs, we can obtain exact all-pairs shortest paths via a technique of Seidel [65]. Specifically, let $G = (V, E)$ an unweighted undirected graph with adjacency matrix A ; the k^{th} power G^k of G is a graph with node set V and edge set $\{\{u, v\} : d(u, v) \leq k\}$. In particular, the *square graph* G^2 can be constructed in $O(n^\rho)$ rounds from G , as the adjacency matrix of G^2 is

$A^2 \vee A$, where the product is over the Boolean semiring and \vee denotes element-wise logical or.

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

Lemma 17 ([65]). *Let $G = (V, E)$ be an unweighted undirected graph with adjacency matrix A , and let D be a distance matrix for G^2 , that is, a matrix with the entries $D_{uv} = d_{G^2}(u, v)$. Let $S = DA$, where the product is computed over integers. We have that*

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

We can now recover all-pairs shortest distances in an undirected unweighted graph by recursively applying Lemma 17.

Corollary 7. *For undirected, unweighted graphs, all-pairs shortest paths can be computed in $\tilde{O}(n^\rho)$ rounds.*

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

1. If $G = G^2$, then $d_G(u, v) = 1$ if u and v are adjacent in G , and $d_G(u, v) = \infty$ otherwise; thus, we are done.
2. Otherwise, we compute all-pairs shortest path distances in the graph G^2 ; since we have already constructed the adjacency matrix for G^2 , we can do the distance computation in G^2 by recursively calling this algorithm with input graph G^2 . Then, we construct the matrix D with entries $D_{uv} = d_{G^2}(u, v)$ as in Lemma 17 and compute $S = DA$. We can recover distances in G using Lemma 17, as each node can transmit their degree in G to each other node in a single round and then check the conditions of the lemma locally.

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

Weighted APSP with small weights. By embedding the distance product of two matrices into a suitable ring, we can use fast ring matrix multiplication to compute all-pairs shortest distances [74]; however, this is only practical for very small weights, as the ring embedding exponentially increases the amount of bits required to transmit the matrix entries. The following lemma encapsulates this idea.

Lemma 18. *Given $n \times n$ matrices S and T with entries in $\{0, 1, \dots, M\} \cup \{\infty\}$, we can compute the distance product $S \star T$ in $O(Mn^\rho)$ rounds.*

Proof. We construct matrices S^* and T^* by replacing each matrix entry w with X^w , where X is a formal variable; values ∞ are replaced by 0. We then compute the product $S^* \cdot T^*$ over the polynomial ring $\mathbb{Z}[X]$; all polynomials involved in the computation have degree at most $2M$ and their coefficients are integers of absolute value at most $n^{O(1)}$, so

this computation can be done in $O(Mn^\rho)$ rounds. Finally, we can recover each matrix entry $(S \star T)_{uv}$ in the original distance product by taking the degree of the lowest-degree monomial in $(S^* \cdot T^*)_{uv}$. \square

Using iterated squaring in combination with Lemma 18, we can compute all-pairs shortest paths up to a small distance M quickly; that is, we want to compute a matrix B such that

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

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

Lemma 19. *Given a directed, weighted graph with non-negative integer weights, we can compute all-pairs shortest paths up to distance M in $O(Mn^\rho)$ rounds.*

The above lemma can be used to compute all-pairs shortest paths quickly assuming that the *weighted diameter* of the graph is small; recall that the weighted diameter of a weighted graph is the maximum distance between any pair of nodes.

Corollary 8. *For directed graphs with positive integer weights and weighted diameter U , all-pairs shortest paths can be computed in $\tilde{O}(Un^\rho)$ rounds.*

Proof. If we know that the weighted diameter is U , we can simply apply Lemma 19 with $M = U$. However, if we do not know U beforehand, we can (1) first compute the reachability matrix of the graph from the unweighted adjacency matrix, (2) guess $U = 1$ and compute all-pairs shortest paths up to distance U , and (3) check if we obtained distances for all pairs that are reachable according to the reachability matrix; if not, then we double our guess for U and repeat steps (2) and (3). \square

Approximate weighted APSP. We can leverage the above result and a rounding technique to obtain a fast $(1 + o(1))$ -approximation algorithm for the weighted directed APSP problem. Similar rounding-based approaches were previously used by Zwick [76] in a centralised setting and by Nanongkai [57] in the distributed setting; however, the idea can be traced back much further [64].

We first consider the computation of a $(1 + \delta)$ -approximate distance product over integers for a given $\delta > 0$; the following lemma is an analogue of one given by Zwick [76] in a centralised setting.

Lemma 20. *Given $n \times n$ matrices S and T with entries in $\{0, 1, \dots, M\} \cup \{\infty\}$, we can compute a matrix \tilde{P} satisfying*

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

where $P = S \star T$ is the distance product of S and T , in $O(n^\rho(\log_{1+\delta} M)/\delta)$ rounds.

Proof. For $i \in \{0, \dots, \lceil \log_{1+\delta} M \rceil\}$, let $S^{(i)}$ be the matrix defined as

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

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

$$\tilde{P}_{uv} = \min_i \{ \lfloor (1 + \delta)^i P_{uv}^{(i)} \rfloor \}$$

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

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

$$P_{uv} = S_{uw} + T_{wv}.$$

Finally, let $j = \lfloor \log_{1+\delta}(\delta P_{uv}/2) \rfloor$. The choice of j means that $P_{uv} \leq 2(1 + \delta)^{j+1}/\delta$; since S_{uw} and T_{wv} are bounded from above by P_{uv} , the entries $S_{uw}^{(j)}$ and $T_{wv}^{(j)}$ are finite. Furthermore, we have

$$(1 + \delta)^j S_{uw}^{(j)} \leq S_{uw} + (1 + \delta)^j, \quad (1 + \delta)^j T_{wv}^{(j)} \leq T_{wv} + (1 + \delta)^j,$$

and therefore

$$\begin{aligned} (1 + \delta)^j P_{uv}^{(j)} &\leq (1 + \delta)^j (S_{uw}^{(j)} + T_{wv}^{(j)}) \\ &\leq S_{uw} + T_{wv} + 2(1 + \delta)^j \\ &\leq P_{uv} + \delta P_{uv} = (1 + \delta)P_{uv}. \end{aligned}$$

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

To see that we can compute the matrix \tilde{P} in the claimed time, we first note that each of the matrices $S^{(i)}$ and $T^{(i)}$ can be constructed locally by the nodes. The product $P^{(i)} = S^{(i)} \star T^{(i)}$ can be computed in $O(n^\rho/\delta)$ rounds for a single index i by Lemma 18, as the entries of $S^{(i)}$ and $T^{(i)}$ are integers bounded from above by $O(1/\delta)$; this is repeated for each index i , and the number of iterations is thus $O(\log_{1+\delta} M)$. Finally, the matrix \tilde{P} can be constructed from matrices $P^{(i)}$ locally. \square

Using Lemma 20, we obtain a $(1 + o(1))$ -approximate APSP algorithm.

Theorem 9. *For directed graphs with integer weights in $\{0, 1, \dots, 2^{n^{o(1)}}\}$, we can compute $(1 + o(1))$ -approximate all-pairs shortest paths in $O(n^{\rho+o(1)})$ rounds.*

Proof. Let $G = (V, E)$ be a directed weighted graph with edge weights in $\{0, 1, \dots, M\}$, where $M = 2^{n^{o(1)}}$. To compute the approximate shortest paths, we apply iterated squaring over the min-plus semiring to the weight matrix W of G , but use the approximate distance product algorithm of Lemma 20 to compute the products. After $\lceil \log n \rceil$ iterations, we obtain a matrix \tilde{D} ; by induction we have

$$d(u, v) \leq \tilde{D}_{uv} \leq (1 + \delta)^{\lceil \log n \rceil} d(u, v) \quad \text{for } u, v \in V.$$

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

To analyse the running time, we observe that we call the algorithm of Lemma 20 $\lceil \log n \rceil$ times; as the maximum distance between nodes in G is $nM = 2^{n^{o(1)}}$, the running time of each call is bounded by

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

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

3.4 Witness Detection for Distance Product

Witness problem for the distance product. As noted in Section 3.3, to recover the routing table in the APSP algorithms based on fast matrix multiplication in addition to computing the shortest path lengths, we need the ability to compute a *witness matrix* for the distance product $S \star T$. That is, we need to find a matrix Q such that if $Q_{uv} = w$, then $(S \star T)_{uv} = S_{uw} + T_{wv}$; in this case, the index w is called a *witness* for the pair (u, v) .

While one can easily modify the semiring matrix multiplication algorithm to provide witnesses, this is not directly possible with the fast matrix multiplication algorithms. However, known techniques from centralised algorithms [4, 65, 76] can be adapted to the congested clique to bridge this gap.

Lemma 21. *If we can compute the distance product for two $n \times n$ matrices S and T in M rounds, we can also find a witness matrix for $S \star T$ in $M \text{ polylog}(n)$ rounds.*

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

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

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

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

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

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

Finding witnesses in the general case. To find witnesses for all indices (u, v) , we reduce the general case to the case of unique witnesses. For simplicity, we only present a randomised version of this algorithm; for derandomisation see Zwick [76] and Alon and Naor [4].

Let $i \in \{0, 1, \dots, \lceil \log n \rceil - 1\}$. We use the following procedure to attempt to find witnesses for all (u, v) that have exactly r witnesses for $n/2^{i+1} \leq r < n/2^i$:

1. Let $m = \lceil c \log n \rceil$ for a sufficiently large constant c . For $j = 1, 2, \dots, m$, construct a subset $V_j \subseteq V$ by picking 2^i values v_1, v_2, \dots, v_{2^i} from V with replacement, and let $V_j = \{v_1, v_2, \dots, v_{2^i}\}$.
2. For each V_j , use the unique witness detection for the product $S(*, V_j) \star T(V_j, *)$ to find candidate witnesses Q_{uv} for all pairs (u, v) , and keep those Q_{uv} that are witnesses for $S \star T$.

Let (u, v) be a pair with r witnesses for $n/2^{i+1} \leq r < n/2^i$. For each $j = 1, 2, \dots, m$, the probability that V_j contains exactly one witness for (u, v) is at least $(2e)^{-1}$ (see Seidel [65]). Thus, the probability that we do not find a witness for (u, v) is bounded by $(1 - (2e)^{-1})^{\lceil c \log n \rceil} = n^{-\Omega(c)}$.

Repeating the above procedure for $i = 0, 1, \dots, \lceil \log n \rceil - 1$ ensures that the probability of not finding a witness for any fixed (u, v) is at most $n^{-\Omega(c)}$. By the union bound, the probability that there is any pair of indices (u, v) for which no witness is found is $n^{-\Omega(c)}$, i.e., with high probability the algorithm succeeds. Moreover, the total number of calls to the distance product is $O((\log n)^3)$, giving Lemma 21.

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 1.4, 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 the wires in C between gates assigned to different nodes defining the communication cost of the algorithm.

Various authors, considering different parallel models, have shown that in any implementation of the trivial $\Theta(n^3)$ matrix multiplication on a parallel machine with P processors there is at least one processor that has to send or receive $\Omega(n^2/P^{2/3})$ matrix entries [2, 41, 69]. As these models can simulate the congested clique, a similar lower bound holds for congested clique implementations of the trivial $O(n^3)$ matrix multiplication. In the congested clique, each processor sends and receives n messages per round (up to logarithmic factors) and $P = n$, yielding a lower bound of $\tilde{\Omega}(n^{1/3})$ rounds.

The trivial $\Theta(n^3)$ matrix multiplication is optimal for circuits using only semiring addition and multiplication. 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. This means that current techniques for similar problems, like the one used in the fast MST algorithm of Lotker et al. [51] cannot be extended to solve APSP.

Corollary 22. *Any implementation of the trivial $\Theta(n^3)$ matrix multiplication, and any APSP algorithm which only sums weights and takes the minimum of such sums, require $\tilde{\Omega}(n^{1/3})$ communication rounds in the congested clique model.*

However, known results on 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 [73] 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 (See [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 23. *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.

Frischknecht et al. [31] have shown that approximating the diameter of an unweighted graph any better than factor $3/2$ requires $\tilde{\Omega}(n)$ rounds in the CONGEST model; the same can be applied to the broadcast congested clique. A variation of the approach was recently used by Holzer and Pinsker [38] to show that computing any approximation better than factor 2 to all-pairs shortest paths in weighted graphs takes $\tilde{\Omega}(n)$ rounds as well. As discussed in Section 3.3, $\tilde{o}(n)$ -round matrix multiplication algorithms imply $\tilde{o}(n)$ -round algorithms for exact unweighted and $(1 + o(1))$ -approximate weighted APSP. Together, this immediately implies that matrix multiplication on the broadcast congested clique is hard.

Corollary 24. *In the broadcast congested clique model, matrix multiplication algorithms that are applicable to matrices over the Boolean semiring and APSP algorithms require $\tilde{\Omega}(n)$ communication rounds.*

We remark that the phrase “that is applicable to matrices over the Boolean semiring” refers to the issue that, in principle, it is possible that matrix multiplication exponents may be different for different underlying semirings. However, at the very least the lower bound applies matrix multiplication over Booleans, integers, and rationals, as well as the min-plus semiring. We stress that, unlike the lower bounds presented beforehand, this bound holds without any assumptions on the algorithm itself.

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 algorithms that outperform the 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 also remains open whether corresponding lower bounds exist; however, it increasingly looks like lower bounds for the congested clique would imply lower bounds for centralised algorithms, and are thus significantly more difficult to prove than for the CONGEST model.

While the present work focuses on a fully connected communication topology (clique), we expect that the same techniques can be applied more generally in the usual CONGEST model. For example, fast triangle detection in the CONGEST model is trivial in those areas of the network that are sparse. Only dense areas of the network are non-trivial, and in those areas we may have enough overall bandwidth for fast matrix multiplication algorithms. On the other hand, there are non-trivial lower bounds for distance computation problems in the CONGEST model [23], though significant gaps still remain [57].

Acknowledgements

Many thanks to 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. doi:10.1147/rd.395.0575.
- [2] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71(1):3–28, 1990. doi:10.1016/0304-3975(90)90188-N.
- [3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. ISBN 0-201-00029-6.
- [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. doi:10.1007/BF01940874.
- [5] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
- [6] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
- [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. doi:10.1145/2312005.2312044.
- [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. doi:10.1145/2395116.2395121.
- [9] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Communication costs of strassen’s matrix multiplication. *Commun. ACM*, 57(2):107–114, 2014. doi:10.1145/2556647.2556660. URL <http://doi.acm.org/10.1145/2556647.2556660>.
- [10] Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM Journal on Computing*, 43(1):280–299, 2014. doi:10.1137/110839229.
- [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)*, volume 8572 of *LNCS*, pages 211–222, 2014. doi:10.1007/978-3-662-43948-7_18.
- [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. doi:10.1145/1250790.1250801.
- [13] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009. doi:10.1137/070683933.
- [14] Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Probably optimal graph motifs. In *Proc. 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *LIPICs*, pages 20–31, 2013. doi:10.4230/LIPICs.STACS.2013.20.
- [15] Andreas Björklund, Petteri Kaski, and Lukasz 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. doi:10.1137/1.9781611973402.45.
- [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. doi:10.1007/978-3-642-39206-1_17.
- [17] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.
- [18] Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008. doi:10.1007/s00453-007-9062-1.
- [19] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010. doi:10.1137/08071990X.
- [20] 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. doi:10.1109/FOCS.2011.23.
- [21] 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. doi:10.1145/2488608.2488646.
- [22] 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.
- [23] 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. doi:10.1145/1993636.1993686.
- [24] 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. doi:10.1007/978-3-642-33651-5_14.
- [25] 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. doi:10.1145/2611462.2611493.
- [26] Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1–3):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.
- [27] 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. doi:10.1109/SWAT.1971.4.
- [28] Fedor V. Fomin, Daniel Lokshantov, 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. doi:10.1016/j.jcss.2011.10.001.
- [29] Fedor V. Fomin, Daniel Lokshantov, 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. doi:10.1137/1.9781611973402.10.
- [30] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5(1):83–89, 1976. doi:10.1137/0205006.
- [31] 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.
- [32] 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.
- [33] 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. doi:10.1145/2608628.2608664.
- [34] 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. doi:10.1007/s00453-007-9063-0.
- [35] 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. doi:10.1007/978-3-642-31155-0_12.
- [36] 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. doi:10.1007/978-3-319-09620-9_13.
- [37] 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. doi:10.1007/978-3-662-45174-8_35.
- [38] Stephan Holzer and N. Pinsker. Approximation of distances and shortest paths in the broadcast congest clique, 2014. arXiv:1412.3445 [cs.DC].
- [39] 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. doi:10.1145/2332432.2332504.
- [40] 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.

- [41] 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. doi:10.1016/j.jpdc.2004.03.021.
- [42] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978. doi:10.1137/0207033.
- [43] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, volume 5125 of *LNCS*, pages 575–586, 2008. doi:10.1007/978-3-540-70575-8_47.
- [44] Mirosław 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. doi:10.1137/1.9781611973082.114.
- [45] Shay Kutten and David Peleg. Fast distributed construction of small k -dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998. doi:10.1006/jagm.1998.0929.
- [46] 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. doi:10.1145/2484239.2501983.
- [47] 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. doi:10.1145/2488608.2488656.
- [48] 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. doi:10.1145/2484239.2484262.
- [49] 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. doi:10.1145/1993636.1993639.
- [50] Daniel Lokshantov and Jesper Nederlof. Saving space by algebraization. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC 2010)*, 2010. doi:10.1145/1806689.1806735.
- [51] 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. doi:10.1137/S0097539704441848.
- [52] 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. doi:10.1145/315891.315965.
- [53] Jirí Matoušek. *Lectures on Discrete Geometry*. Graduate Texts in Mathematics. Springer, 2002. ISBN 9780387953731.
- [54] William F. McColl. Scalable computing. In *Computer Science Today*, volume 1000 of *LNCS*, pages 46–61. Springer, 1995. doi:10.1007/BFb0015236.
- [55] 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.
- [56] Ian Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1(2):56–58, 1971. doi:10.1016/0020-0190(71)90006-8.
- [57] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. 46th ACM Symposium on Theory of Computing (STOC 2014)*, pages 565–573, 2014.

- [58] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- [59] 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. doi:10.1145/1993806.1993851.
- [60] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.
- [61] 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. doi:10.1137/S0097539700369740.
- [62] 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. doi:10.1007/978-3-642-31585-5_58.
- [63] Sriram V. Pemmaraju and Vivek B. Sardeshmukh. Minimum-weight spanning tree construction in $\Theta(\log \log \log n)$ rounds on the congested clique. *CoRR*, abs/1412.2333, 2014. URL <http://arxiv.org/abs/1412.2333>.
- [64] 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.
- [65] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995. doi:10.1006/jcss.1995.1078.
- [66] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. doi:10.1007/BF02165411.
- [67] 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. doi:10.1007/978-3-540-27798-9_31.
- [68] 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. doi:10.1016/j.ipl.2005.08.008.
- [69] 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. doi:10.1007/BFb0055078.
- [70] Alexandre Tiskin. *The Design and Analysis of Bulk-Synchronous Parallel Algorithms*. PhD thesis, University of Oxford, 1999.
- [71] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal of Computing*, 42(3):831–854, 2013. doi:10.1137/09076619X.
- [72] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009. doi:10.1016/j.ipl.2008.11.004.
- [73] 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. doi:10.1145/2591796.2591811.
- [74] Gideon Yuval. An algorithm for finding all shortest paths using $n^{2.81}$ infinite-precision multiplications. *Information Processing Letters*, 4(6):155–156, 1976. doi:10.1016/0020-0190(76)90085-5.
- [75] Uri Zwick. Exact and approximate distances in graphs – A survey. In *Proc. 9th European Symposium on Algorithms (ESA 2001)*, pages 33–48, 2001. doi:10.1007/3-540-44676-1_3.
- [76] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

- [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. doi:[10.1007/s00453-005-1199-1](https://doi.org/10.1007/s00453-005-1199-1).