The Distributed Computing Column

Seth Gilbert
National University of Singapore
`seth.gilbert@comp.nus.edu.sg`

This month, the Distributed Computing Column is featuring Dean Leitersdorf, winner of the 2023 Principles of Distributed Computing Doctoral Dissertation Award. His work on sparse matrix multiplication has led to several breakthroughs that improve significantly on the state of the art. These new approaches have led to faster algorithms for a variety of related problems, including constant-round algorithms for computing graph spanners, approximate all-pairs-shortest-paths, and the girth of a graph (up to an additive 1) in the congested clique model.

Recent progress in distributed matrix multiplication has been fast, and real-world applications of matrix multiplication have only been increasing in importance. In this column, Dean Leitersdorf gives an overview of the state of the art for distributed matrix multiplication, and its connection to all-pairs shortest paths in the congested clique model. He provides both a summary of his new sparity-aware approach, along with a discussion of open questions and possible future directions. Overall, then, this column provides a succinct overview of a rapidly moving area of distributed algorithms!

The Distributed Computing Column is particularly interested in contributions that propose interesting new directions and summarize important open problems in areas of interest. If you would like to write such a column, please contact me.

The Relationship between APSP and Matrix Multiplication in Congested Clique

Dean Leitersdorf

Introduction

The field of distributed computing has recently explored the fundamental bidirectional relationship between matrix multiplication and the all-pairs-shortest-path (APSP) problem in the distributed Congested Clique model. Matrix multiplication is the foundation for a wide variety of problems in the exact sciences, with significant algorithmic research effort having been invested in finding the smallest ω such that sequential matrix multiplication [12, 20, 33, 34] can be computed in $O(n^{\omega})$ time (where $\omega < 2.37286$ is the best currently-known result [4]). Similarly, graph theory has been the bedrock of computer science research for much of the past century, with the fundamental problem of distance computation having wide implications. While it is well known that the all-pairs-shortest-path (APSP) variant of distance computation can be related to matrix exponentiation through the min-plus semiring, a recent line of works in distributed computing has explored the implication of this connection on approximation algorithms that utilize only $o(\text{poly}(n))$ Congested Clique rounds.

In this text, we analyze this line of works and discuss the future research directions of the field. The current state-of-the-art results exploit exact Congested Clique matrix multiplication to perform exact APSP in $O(n^{1/3})$ rounds, and $O(1)$-approximate APSP in $O(\log\log n)$ rounds or $O(\text{poly}\log n)$-approximate APSP in $O(1)$ rounds. It is of significant interest whether it is possible to perform $O(1)$-approximate APSP in $O(1)$ rounds as this would imply a drastic reduction in time complexity between exact APSP in $O(\text{poly}(n))$ rounds and approximate APSP (with a constant approximation factor) in constant rounds. In the rest of this text we highlight the results of this research direction and speculate with regards to future results.

Preliminaries

We begin by presenting the Congested Clique model, proceed to defining matrix multiplication (MM) and the all-pairs-shortest-path (APSP) problem in this setting, and conclude the preliminaries by showing some basic relationships between MM and APSP in Congested Clique.

In CONGEST [32], computational nodes and communication links between them are represented by a graph, G. Every node can perform unlimited computation, is initially only aware of its incident edges, and can communicate in synchronous rounds with its neighbors by sending each a $O(\log n)$-bit message per round. Typically, algorithms solve problems over G (e.g. distance computations), where the main complexity measure is the number of rounds an algorithm takes. The Congested Clique model [28] is similar to CONGEST, but separates the input from the communication topology: the input is a graph G, but the communication topology allows all nodes to communicate directly with one another.

Definition 1 (MM in Congested Clique)**.** *Let S, T be two n by n matrices such that, for all i, node i holds the i^{th} rows of S and T. Computing the product $P = S \cdot T$ in* Congested Clique *is achieved when, for all i, node i knows row i of P.*

Definition 2 (APSP in Congested Clique)**.** *Let G be an input graph. Computing the APSP over G is achieved when for for every nodes u, v, node u knows the distance from u to v in G.*

Brief Overview

Many recent Congested Clique papers study distance problems [7,10,18,21,24,29,31]. Specifically, [10, 21] exploit a well-known connection between distances and matrix multiplication – the n^{th} power of the adjacency matrix A of a graph, taken over the min-plus (or tropical) semiring, corresponds to shortest-path distances. Hence, iteratively squaring a matrix $\log n$ times gives the best known algorithms for APSP in Congested Clique, including (1) an $\widetilde{O}(n^{1/3})$ round algorithm for exact APSP in weighted directed graphs [10], (2) $O(n^{0.158})$ round algorithms for exact APSP in unweighted undirected graphs and $(1 + o(1))$-approximate APSP in weighted directed graphs [10], and (3) an $O(n^{0.2096})$ round algorithm for exact APSP in directed graphs with constant weights [21].

Faster approximations for larger constants are obtained by computing a k-spanner (sparse subgraph approximating distances by a factor of k), and having all nodes learn the spanner. Using the results of [31], this gives a $(2k - 1)$-approximation for APSP in $\widetilde{O}(n^{1/k})$ rounds, which is polynomial for any constant k. This raises the following fundamental question.

Question 1. *Are there constant-factor APSP approximations in sub-polynomial time?*

For SSSP, this is indeed possible [7, 23], with a gradient-descent-based algorithm obtaining a $(1 + \epsilon)$-approximation in $O(\epsilon^{-3}\text{polylog } n)$ rounds (even in BCC) [7].

In [27], we develop a line of sparsity aware matrix multiplication algorithms. The complexities of our algorithms do not depend on the sparsity structures of the matrices, rather, only on the total number of non-zero elements. We then apply our sparse matrix multiplication algorithms to efficiently implement basic primitives for distance computations. For instance, we compute for every node the distances to the $O(n^{2/3})$ nodes closest to it in $O(\text{poly} \log n)$ rounds. Together with other tools we develop, we show a $(2 + \epsilon)$ APSP approximation in $O(\log^2 n/\epsilon)$ rounds. This is the first sub-polynomial constant-factor APSP approximation, and is an *exponential* speedup over previous results. Following our work and using our distance tools, [16] show further improvement, bringing the round complexity down to $O(\text{poly} \log \log n)$, leading to the following question.

Question 2. *Is it possible to obtain an $o(\log \log n)$-round good APSP approximation?*

In [27], we enhance our tools using a technique we call *partition trees*, to develop a sparsity-aware sparsification tool to answer this in the affirmative. We construct $O(\log n)$-spanners and approximate APSP (with polylogarithmic approximation factors) in $O(1)$ rounds.

The Matrix Multiplication Cube

We describe the following visualization accompanying our techniques. How does one multiply matrices? By definition, given two n by n matrices, S and T, matrix P is their product if

$$\forall i, j \in [n] : P[i, j] = \sum_{k \in [n]} S[i, k] T[k, j] \, .$$

Typically, in introductory linear algebra courses, this is shown via a 2-dimensional depiction: “multiply (element-wise) a row of S by a column of T, and sum the values”. Instead, we use the *3D Cube of Matrix Multiplication* (see e.g. [2, 3]). Formally, it is an $n \times n \times n$ cube, where entry (i, j, k) corresponds to $S[i][k]T[k][j]$. Two dimensions correspond to S and T, and each index in the third dimension represents a page, where P is the sum of all n pages.

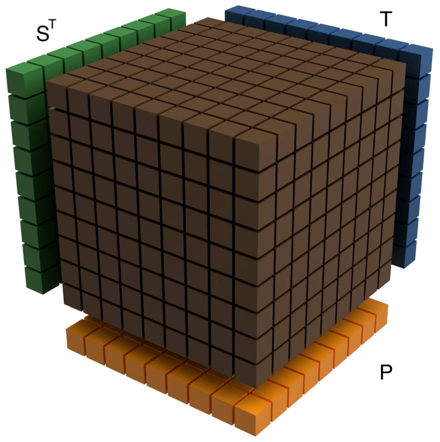

Figure 1: An illustration of the multiplication cube. The green (left) and blue (right) shapes correspond to S (transposed) and T, respectively, the brown (center) are point-wise multiplications, and the orange (bottom) are P. The value of every brown shape is the product of its green and blue projections. The value of every orange shape is the sum of the brown values above it.

The cube is useful for visualizing parallel and distributed algorithms. Assume k computational devices. A subcube $V_1 \times V_2 \times V_3$, where $V_1, V_2, V_3 \subseteq [n]$, corresponds to the task $S[V_1, V_2] \cdot T[V_2, V_3]$, where $S[V_1, V_2]$ is the submatrix limited to rows V_1 and columns V_2 of S, and similarly for T. Thus, a partition of n^3 into k subcubes breaks the larger matrix multiplication into smaller tasks.

Efficiently partitioning the cube is heavily dependant on the distributed setting considered. To compute the value at index (i, j, k), a device needs both its projections, $S[i][k]$ and $T[k][j]$. How can we factor in the (potentially different) sparsities of S and T? Is it easier to learn the projections onto S than those onto T? Computing the values of P requires computational devices computing intermediate values “above” one another (in the figure) to communicate – is this communication more expensive than that for learning the projections onto S and T?

We split our approaches for overcoming these challenges to input-sparsity awareness (S and T) and output-sparsity awareness (P). Designing output-sparsity aware algorithms is problematic as P, by definition, is not known in advance. Thus, this necessitates adaptive behaviour throughout the communication, before the output is fully computed. Our most complex algorithm, Filtered Sparse Matrix Multiplication, deals with dense output, where only some sparse set of it (matching specific predicates) is desired. Thus, already at the stage when only some of the intermediate computation is performed, we deduce which values of P we want to fully compute.

Note that in [27], we extend this further to deal with *subgraph existence problems* and not just matrix multiplication and distance computations. What if we use higher dimensions (4, 5, 6, etc.)? What if we reverse the information flow such that data flows *into* the cube from the bottom (orange shapes) instead of *out*? What if, in some dimensions, we can perform *quantum* computation? For instance, reversing the information flow w.r.t. P turns a boolean matrix multiplication algorithm to one that finds all triangles (three fully connected nodes) in a graph. Thus, the cube amounts to *a unified perspective for seemingly unrelated distributed problems – intuition w.r.t. a property of the cube is carried over to many problems.*

Warmup: Using the MM Cube for Input Sparsity Awareness for exact APSP

In [11], we investigate how to perform matrix multiplication faster if it is given that the two input matrices are sparse. As a warmup, we leverage this to design faster exact APSP algorithms for sparse graphs. Later, we show how to extend the matrix multiplication algorithms to also take into account the sparsity of the output matrix, which allows approximating APSP exponentially faster, even on general graphs.

A central challenge in non-sequential matrix multiplication is high skew in input matrices, as Ballard et al. [5] describe in the parallel setting: "*[We] are not aware of any algorithms that dynamically determine and efficiently exploit the structure of general input matrices. In fact, a common technique of current library implementations is to randomly permute rows and columns of the input matrices in an attempt to destroy their structure and improve computational load balance.*" We show deterministic algorithms overcoming this, as well as other challenges which arise in distributed settings and not in parallel or sequential ones.

In Congested Clique, the round complexity is typically dominated by the node participating in the most communication. This leads us to defining two main goals: minimizing the total message count, and implementing load balancing mechanisms to ensure the round complexity is governed by the average number of messages each node communicates, and not the maximal.

On a high-level, our approach is threefold, with the first part minimizing the total number of messages sent, and the latter parts load balancing among the nodes.

First, split the $n \times n \times n$ matrix multiplication cube into n equally sized sub-cubes whose dimensions are determined dynamically, based on the sparsity of the input matrices. Fix some values a, b. We partition P into ab sub-matrices of size $n/a \times n/b$, denoted by $P_{i,j}$ for $i \in [a]$, $j \in [b]$, and assign n/ab nodes, denoted $N_{i,j}$, for computing $P_{i,j}$.

Second, notice that permutations of rows of S and columns of T result in a reversible permutation of P. Thus, we permute the S and T such that the number of non-zero entries required for computing each $n/a \times n/b$ sub-matrix is roughly the same for each sub-matrix. We call the two permuted matrices, S' and T', *sparsity-balanced matrices with respect to* (a, b). The rest of our algorithm deals with computing the product of such matrices.

Third, we assign the computation of pages of sub-matrices to nodes in a non-consecutive manner. Each $P_{i,j}$ is the sum of n sub-pages $P_{i,j,\ell}$. Each $v \in N_{i,j}$ computes some of the $P_{i,j,\ell}$ sub-pages and sums them locally. The local sums are then aggregated, to obtain $P_{i,j}$. We assign sub-pages to nodes in a non-consecutive manner, such that each node receives a roughly equal number of non-zero entries to compute its assigned sub-pages (See Figure 2).

While the above ensure nodes receive roughly the same number of messages, it is paramount that nodes also send a roughly equal number of non-zero matrix entries. We rearrange the entries of S and T held by each node such that each holds a roughly equal amount of non-zero entries. In this step, we do not permute S or T, rather, we merely redistribute their entries.

Crucially, these assignments are not global knowledge, leading to routing challenges. That is, for every $P_{i,j}$, nodes $N_{i,j}$ decide which matrix entries are received by which node, yet, this is

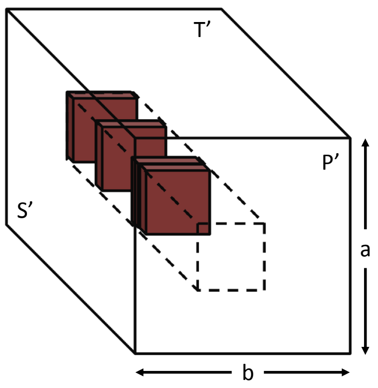

Figure 2: An illustration of the multiplication cube for $P' = S'T'$. Each sub-matrix is assigned to n/ab nodes, with a not necessarily consecutive page assignment that is computed on-the-fly to minimize communication.

unknown to other nodes who need to send the entries. Likewise, the redistribution of S and T is not known to all. However, clearly, a node must know the destination of each message it sends. We design our solution such that every node computes a small set of nodes potentially holding information it requires, and request it from them. Upon receipt of the request, the nodes can compute whether they actually have the relevant information, and, if so, send it over.

For matrix M, let $nz(M)$ denote the total number of non-zero entries in M, and let $\rho_M = \lceil nz(M)/n \rceil$ denote the density of M. Our first main contribution is the following.

Theorem 1. *Given two $n \times n$ matrices S and T, it is possible to deterministically compute $P = ST$ over a semiring, within $O((\rho_S \rho_T)^{1/3}/n^{1/3} + 1)$ rounds in* Congested Clique.

An important case of Theorem 1, especially when squaring the adjacency matrix of a graph, is when the sparsities of the input matrices are roughly the same.

Corollary 2. *Given two $n \times n$ matrices S and T, where $O(nz(S)) = O(nz(T)) = m$, it is possible to compute $P = ST$ over a semiring, in $O(m^{2/3}/n + 1)$ rounds in* Congested Clique.

For $m = O(n^2)$, Corollary 2 gives the same $O(n^{1/3})$ rounds complexity as that of [10], the state-of-the-art for non-sparse matrix multiplication. Our algorithm is fast also when only one of the matrices is sparse, as stated in the following.

Corollary 3. *Given two $n \times n$ matrices S and T, where $\min\{nz(S), nz(T)\} = m$, it is possible to compute $P = ST$ over a semiring, in $O((m/n)^{1/3} + 1)$ rounds in* Congested Clique.

This allows computing powers that are larger than 2 of a sparse input matrix. We cannot repeatedly square a matrix, as this may require multiplying dense matrices, yet, we can repeatedly increase its power by 1. This gives the following for exact APSP, whose comparison to the state-of-the-art depends on the trade-off between the number of edges and the graph diameter.

Theorem 4. *Given an unweighted graph G, there is a deterministic algorithm that computes APSP in $O(D((m/n)^{1/3} + 1))$ rounds in the* Congested Clique *model.*

To compare, the best known complexity for general graphs is $O(n^{1-2/\omega})$ [10] (currently roughly $O(n^{0.158})$). For a graph with $m = o(n^{4-6/\omega}/D^3)$ (currently $o(n^{1.474}/D^3)$), our algorithm is faster.

$O(1)$ APSP approximation in $O(\text{poly}\log\log n)$ rounds

In [9], we turn our attention to general graphs, and aim to compute a constant approximation of APSP in sub-polynomial rounds.

Distance Products. We start from the basic idea of using matrix multiplication to compute distances. Specifically, if A is the weighted adjacency matrix of a graph G, it is well known that distances in G can be computed by iterating the distance product $A \star A$, defined as

$$(A \star A)[i, j] = \min_k (A[i, k] + A[k, j]),$$

that is, the matrix multiplication over the min-plus semiring.

However, as $A \star A$ can be dense even if A is sparse (e.g. a star graph), iterative squaring is not guaranteed to be efficient. Moreover, our goal is to compute distances in general graphs and so we do not even assume A itself is sparse. We thus take a step back, first showing several distance computation building blocks, before directly tackling end-problems such as APSP.

Our Distance Tools. The key observation is that building blocks for distance computation are actually based on computations in sparse graphs or subgraphs. Concrete examples include:

- k-NEAREST: Compute distances for each node to the k nodes closest to it.
- (S, d, k)-SOURCE DETECTION: Given a set of sources S, compute the distances for each node to the k sources closest to it, using paths of at most d hops.
- DISTANCE THROUGH SETS: Given a set of nodes S and distances to all nodes in S, compute the distances between all nodes using paths through nodes in S.

For all of these problems, there is a degree of sparsity we can hope to exploit if k or $|S|$ are small. For example, the (S, d, k)-SOURCE DETECTION problem, requires the multiplication of a dense adjacency matrix with a possibly sparse matrix, depending on the size of S. However, for any S of polynomial size, the round complexity of the input sparsity aware algorithm is polynomial. An interesting property in this problem is that the output matrix is also sparse, giving room for improvement. As another example, in k-NEAREST both input matrices are sparse, which makes it fast using the previous sparse matrix multiplication algorithm. However, this does not exploit the sparsity of this problem to the end: we are interested only in computing the k nearest nodes to each node, hence there is no need to compute the full output matrix. The challenge in this case is that we do not know the identity of the k closest nodes before the computation.

To exploit this sparsity we design new matrix multiplication algorithms that, in particular, can sparsify the output matrix throughout its computation, and get a complexity that depends only on the size of the output we are interested in. The core reason that significant challenges arise in output sparsity awareness w.r.t. input sparsity awareness is that the input matrices are known beforehand, while clearly this does not hold in the output case. Thus, we are required to recognize patterns in the output during the computation and perform actions affected by these patterns. In particular, our approaches uses both *binary search and sorting*, in a novel way, to efficiently perform the summation step of the matrix multiplication, while abstracting away effects of the output patterns. We obtain the following matrix multiplication variants.

- One variant assumes that the sparsity of the output matrix is known.

- The other sparsifies the output on the fly, keeping the ρ_P smallest entries in each row.

For these two scenarios, we obtain round complexities

$$O\Big(\frac{(\rho_S\rho_T\rho_P)^{1/3}}{n^{2/3}}+1\Big), \qquad \text{and} \qquad O\Big(\frac{(\rho_S\rho_T\rho_P)^{1/3}}{n^{2/3}}+\log n\Big),$$

respectively, improving over our input sparsity aware matrix multiplication for $\rho_P = o(n)$.

Applications of Sparse Matrix Multiplication. Using output sensitive sparse matrix multiplication, we obtain faster distance tools:

- We solve k-NEAREST in $O\left(\left(\frac{k}{n^{2/3}}+\log n\right)\log k\right)$ rounds.
- We solve (S, d, k)-SOURCE DETECTION in $O\left(\left(\frac{m^{1/3}|S|^{2/3}}{n}+1\right)d\right)$ rounds, where m is the number of edges in the output graph; note that dependence on d becomes linear in order to exploit the sparsity.

In concrete terms, with these output sensitive distance tools we still get subpolynomial running times even when the parameters are polynomial. For example, we get the distances to the $\widetilde{O}(n^{2/3})$ closest nodes in $\widetilde{O}(1)$ rounds. Note that though our final results are only for undirected graphs, these distance tools work for directed, weighted graphs.

Hopsets. An issue with our (S, d, k)-SOURCE DETECTION algorithm is that in order to exploit the sparsity of the matrices, we must perform d multiplications to learn the distances of nodes at hop-distance at most d from S. Hence, to learn the distances of all nodes from S, we need to do n multiplications, which is no longer efficient. To overcome this challenge, we use hopsets. Given a (β, ϵ)-hopset H, it is enough to look only at β-hop distances in $G \cup H$ to approximate distances by a factor of $(1+\epsilon)$. Using our source detection algorithm together with a hopset allows getting an efficient algorithm for approximating distances, as long as β is small enough.

However, the round complexities of all current hopset constructions [17, 18, 23] is at least $O(\rho)$ for a hopset of size $n\rho$. This is a major obstacle for efficient shortest paths algorithms, since, based on recent existential results, there are no hopsets where both β and ρ are polylogarithmic [1]. Nevertheless, we show that our new distance tools build a hopset in a time that does not depend on its size. In particular, we show how to implement a variant of the recent hopset construction of in [18] in $O(\frac{\log^2 n}{\epsilon})$ rounds. The size of our hopset is $\widetilde{O}(n^{3/2})$, hence constructing it using previous algorithms requires at least $\widetilde{O}(\sqrt{n})$ rounds.

Applying the Distance Tools. As a direct application of our source detection and hopset algorithms, we obtain an algorithm for computing distances from k sources at once (k-SSP). This is the first sub-polynomial result, with such approximations, for polynomial k.

Theorem 5. *There is a deterministic $(1+\epsilon)$-approximation algorithm for weighted undirected k-SSP that takes*

$$O\Big(\Big(\frac{k^{2/3}}{n^{1/3}}+\log n\Big)\cdot\frac{\log n}{\epsilon}\Big)$$

rounds in the Congested Clique*, where S is the set of sources. In particular, the complexity is $O(\frac{\log^2 n}{\epsilon})$ as long as $k = O(\sqrt{n}\cdot(\log n)^{3/2})$.*

In turn, this forms the basis of our $(3 + \epsilon)$-approximation for weighted APSP. To obtain a $(2+\epsilon)$-approximation for unweighted APSP, the idea is to deal separately with paths containing a high-degree node and paths without. A crucial ingredient is showing that in sparser graphs we can efficiently compute distances to a larger set S.

Theorem 6. *There is a deterministic $(2+\epsilon)$-approximation algorithm for unweighted undirected APSP in the* Congested Clique *model that takes $O(\frac{\log^2 n}{\epsilon})$ rounds.*

Our approximation is almost tight for sub-polynomial algorithms in the following sense. As noted by [26], a $(2 - \epsilon)$-approximate APSP in unweighted undirected graphs is essentially equivalent to fast matrix multiplication, so obtaining a better approximation in complexity below $O(n^{0.158})$ would result in a faster algorithm for non-sparse matrix multiplication in the Congested Clique. Likewise, a sub-polynomial-time algorithm with *any* approximation ratio for directed graphs would give a faster matrix multiplication algorithm [14].

As stated above in the overview, this concludes the first sub-polynomial constant-factor APSP approximation, showing an *exponential* speedup over previous results which all required polynomial rounds for such an approximation. Following this work and using the above described distance tools, [16] bring the complexity down to $O(\text{poly} \log \log n)$. In weighted graphs, an $O(\log^{1+\epsilon} n)$-approximation to APSP is known in a similar round complexity [8].

$O(\text{poly} \log n)$ APSP approximation in $O(1)$ rounds

In [15], we ask whether it is possible to spend *just* $O(1)$ rounds in order show a good approximation of APSP. We answer this by showing poly-logarithmic approximations to APSP in such a round complexity.

Spanners. A k-spanner is a sparse subgraph, preserving distances up to a factor of k. Any graph has a $(2k - 1)$-spanner with $O(n^{1+1/k})$ edges, which is assumed to be tight by the Erdős Girth Conjecture [19]. We aim for spanners with $O(n)$ edges, requiring $k = \Theta(\log n)$. The classic algorithm of [6] builds $(2k - 1)$-spanners with $O(kn^{1+1/k})$ edges in expectation, and is simulated in $O(k)$ rounds. Faster, $\text{poly}(\log k)$-round algorithms appear in [8, 31]. Specifically, [31] shows a randomized (deterministic) construction of $(2k-1)$-spanners ($O(k)$-spanners) with $\widetilde{O}(n^{1+1/k})$ edges ($O(kn^{1+1/k})$ edges). For weighted undirected graphs, [8] show a randomized construction of $O(k^{1+o(1)})$-spanners with $O(n^{1+1/k} \cdot \log k)$ edges.

The Locality Barrier. Intuitively, these take $\text{poly}(\log k)$ rounds as the locality of spanners is linear in k. In the LOCAL model, where a node can learn its entire t-neighborhood in t rounds, constructing $(2k-1)$-spanners with $O(n^{1+1/k})$ edges takes $\Omega(k)$, assuming the Erdős Girth Conjecture [13]. The connection between LOCAL and Congested Clique is captured via graph exponentiation, whereby nodes learn their 2^i-neighborhoods in round i. This requires much bandwidth, nevertheless, is sometimes used with other ideas, leading to $O(\log t)$ algorithms in Congested Clique based on t-round algorithms in LOCAL (e.g. [16, 31]). In variants of MPC, this approach is conditionally tight [22], for certain algorithms.

We break through the locality barrier, by utilizing techniques from our distances computations (above), our partition trees tool (see **??**), as well as recent developments for computing a minimum spanning tree (MST) in Congested Clique in $O(1)$ rounds [25, 30].

A powerful ingredient is our sparsification tool, Theorem 7, based on our partition trees (**??**), that finds spanners for graphs F connecting some nodes in G.

Theorem 7. *Let $G = (V, E)$ be a* Congested Clique*, and $F = (V_F, E_F)$ a graph with $V_F \subseteq V$, $|V_F| = N$ nodes, $|E_F| = M$ edges, and maximum degree Δ_F. There is an $O(\frac{M^{1/3} \cdot N^{2/3}}{n} + 1)$-round algorithm in G that finds a $(2k-1)$-spanner for F with $O(M^{1/3} \cdot N^{2/3+1/k})$ edges.*

In a recent work on MST computation, given $F = (V_F, E_F)$, $|V_F| = N$, $|E_F| = M$, [30] uses the following. Let $d = 2M/N$ and partition V_F into $S_1, \cdots, S_{\sqrt{d}}$, where $|S_i| = O(N/\sqrt{d})$ and $|E(S_i, S_j)| = O(N)$. Then, the edges $E(S_i, S_j)$ are sent to some node, which replaces them with an MST, with $|S_i| + |S_j|$ edges, on the graph induced by $S_i \times S_j$. Since $|S_i \cup S_j| = O(N/\sqrt{d})$, each MST has $O(N/\sqrt{d})$ edges, and all MSTs have $d \cdot O(N/\sqrt{d}) = O(\sqrt{M \cdot N})$ edges in total.

We show Theorem 7 by partitioning into $d^{1/3}$ sets, and follow the notion where each node sparsifies the edges it gets, by returning a spanner (instead of an MST). Our partitioning in [15] is randomized, and using our partition trees technique (see **??**), we show a deterministic version. The key behind this is that applying our partition trees technnique on the "subgraph" $H = (V_H, E_H)$ which is simply one edge (i.e. $V_H = \{a, b\}, E_H = \{\{a, b\}\}$) redistributes the edges of $G = (V, E)$ such that every v knows some set of edges E_v, where $\{E_v | v \in V\}$ is a partition of E, and the amount of nodes incident to each E_v is rather small. Based on this approach, we show a very powerful partitioning which leads to Theorem 7.

Unweighted Graphs We apply Theorem 7 on cluster graphs generated by a smart sampling procedure. Let $d = 2m/n$. We construct a cluster graph C. Find a hitting set $D \subseteq V$ (every node is either in D or has at least one neighbor in D) of size $O(n \log d/d)$. Then, each node in D is denoted the center of a cluster in C, and each node in $V \setminus D$ joins the cluster of one of its neighbors in D. For any two clusters C_1 and C_2 in C, connect them with an edge if, in G, any node in C_1 is connected to any node in C_2. As C has $M \leq m$ edges but only $N = O(n \log d/d)$ nodes, we can apply Theorem 7 on C, to get a spanner with $\ell = O(M^{1/3} \cdot N^{2/3+1/k}) = O((dn)^{1/3} \cdot (n \log d/d)^{2/3+1/k}) = O(n^{1+1/k})$ edges in $O(\frac{M^{1/3} \cdot N^{2/3}}{n}) = O(1)$ rounds.

Any α-spanner H_C of C with ℓ edges can be translated to an $O(\alpha)$-spanner H for G with $\ell + n$ edges. Replace each edge in H_C by an edge in G that connects two nodes in the corresponding clusters. Then, for each $v \in V$, add to H the edge that connects it to the center of the cluster v belongs to. H is an $O(\alpha)$-spanner for G, as any path P_C of length α in H_C translates to a path P in H through $\alpha + 1$ clusters. As the radius of each cluster is 1, P has length $O(\alpha)$.

However, it is hard to find a hitting set with $|D| = O(n \log d/d)$. In graphs with minimum degree d, a sampling procedure suffices, yet d is our average degree. Thus, we bucket G, where bucket i contains edges E_i, incident to nodes with degree in $[2^i, 2^{i+1})$, and run the above, for bucket i[1] computing C_i and H_{C_i}. The size of all spanners, where Δ is the maximum degree, is:

$$|\bigcup_{i=0}^{\log \Delta} H_i| = |H_0| + \sum_{i=1}^{\log \Delta} \mathrm{O}((ni/2^i)^{2/3+1/k}(n2^i)^{1/3}) \leq \mathrm{O}(n) + \sum_{i=1}^{\log \Delta} \mathrm{O}(n^{1+1/k}) i^{2/3}/2^{i/3} = \mathrm{O}(n^{1+1/k})$$

Finally, we convert H_{C_i} to H_i which is a spanner for edges E_i of G, and $H = \bigcup_i H_i$ is a spanner of G, as required. However, as above, $|H_i| \leq |H_{C_i}| + n$, where the $+n$ is due to connecting each node its cluster center. This may result in $|H| = O(n^{1+1/k} + n \log \Delta)$, instead of $O(n^{1+1/k})$. To solve this, we construct the graphs C_i such that for each $v \in V$, all clusters v belongs to have the same center. Hence, each node adds at most only one edge in total, which means that across all H_i at most n edges are added, and not n edges per H_i.

[1]Except for $i = 0$, where we define H_{C_i} as all edges incident to a node of degree at most 2.

Theorem 8. *Given k and an undirected unweighted graph G, there is an $O(1)$-round* Congested Clique *algorithm constructing an $O(k)$-spanner for G with $O(n^{1+1/k})$ edges, w.h.p.*

Choosing $k = \Theta(\log n)$ gives an $O(\log n)$-spanner of size $O(n)$, thus implying the following.

Theorem 9. *Given an undirected unweighted graph G, there is an $O(1)$-round* Congested Clique *algorithm computing an $O(\log n)$-approximation of APSP w.h.p.*

Weighted Graphs We develop two additional tools to extend our results to weighted graphs. First, we split the edges into buckets of exponentially increasing weights $B_l = \{e \mid 2^l \leq w(e) < 2^{l+1}\}$, and construct, in parallel, a spanner for each bucket using the unweighted algorithm. The union of these spanners is an $O(k)$-spanner for G, yet with a total of $O(n^{1+1/k} \log n)$ edges. To obtain a spanner with $O(n^{1+1/k})$ edges (yet $O(k \log n)$ stretch), we draw a connection to MSTs. We construct an MST using [30] and use it to contract the graph to $n/\log n$ nodes, while preserving distances up to a factor of $O(\log n)$. This is done by replacing low diameter subtrees of the MST, each with a single node, and due to the property that the edges of the MST cannot be the heaviest along a cycle, we show that graph distances are stretched only by $O(\log n)$. Then, we execute the above spanner algorithm on the contracted graph, and as the number of nodes is $n/\log n$, the resulting spanner has $O((n/\log n)^{1+1/k} \log(n/\log n)) = O(n^{1+1/k})$ edges.

Theorem 10. *Given an undirected weighted graph G, there is an $O(1)$-round algorithm in the* Congested Clique *model that computes an $O(\log^2 n)$-approximation of APSP, w.h.p.*

Conclusion

The research community has recently made rapid progress in APSP approximations in the Congested Clique model, mainly based on algorithms solving variants of matrix multiplication. Currently, the state-of-the-art is a constant approximation in $O(\text{poly} \log \log n)$ rounds, or an $O(\log n)$ approximation in constant rounds ($O(\text{poly} \log n)$ if the graph is weighted). We hypothesize that the community is close to achieving a constant approximation in constant rounds barring a few additional insights. This is certain to lead to several new exciting research directions in the upcoming years.

Concretely, it is interesting to see whether the techniques behind the two extreme results detailed above (constant approximation in $O(\text{poly} \log \log n)$ rounds or $O(\text{poly} \log n)$ approximation in constant rounds) can be merged in order to show a better overall result. These are active research directions which the community is currently investigating.

References

[1] Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM Journal on Computing*, 47(6):2203–2236, 2018.

[2] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. 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.

[3] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Communication complexity of prams. *Theoretical Computer Science*, 71(1):3–28, 1990. doi:https://doi.org/10.1016/0304-3975(90)90188-N.

[4] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539, 2021. doi:10.1137/1.9781611976465.32.

[5] Grey Ballard, Aydin Buluç, James Demmel, Laura Grigori, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. Communication optimal parallel multiplication of sparse random matrices. In *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA)*, pages 222–231, 2013. URL: http://doi.acm.org/10.1145/2486159.2486196, doi:10.1145/2486159.2486196.

[6] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

[7] Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *SIAM Journal on Computing*, 50(3):815–856, 2021. doi:10.1137/19M1286955.

[8] Amartya Shankha Biswas, Michal Dory, Mohsen Ghaffari, Slobodan Mitrovic, and Yasamin Nazari. Massively parallel algorithms for distance approximation and spanners. *SPAA 2021*, 2021.

[9] Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing*, 2020. doi:10.1007/s00446-020-00380-5.

[10] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Computing*, 32(6):461–478, 2019. doi:10.1007/s00446-016-0270-2.

[11] Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. *Theoretical Computer Science*, 809:45–60, 2020. doi:10.1016/j.tcs.2019.11.006.

[12] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

[13] Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the locality of distributed sparse spanner construction. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing (PODC)*, pages 273–282, 2008.

[14] Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[15] Michal Dory, Orr Fischer, Seri Khoury, and Dean Leitersdorf. Constant-round spanners and shortest paths in congested clique and mpc. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, page 223–233, 2021. doi:10.1145/3465084.3467928.

[16] Michal Dory and Merav Parter. Exponentially faster shortest paths in the congested clique. In *PODC '20: ACM Symposium on Principles of Distributed Computing, August 3-7, 2020*, pages 59–68. ACM, 2020. doi:10.1145/3382734.3405711.

[17] Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. *SIAM J. Comput.*, 48(4):1436–1480, 2019. doi:10.1137/18M1166791.

[18] Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in RNC. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 333–341, 2019. doi:10.1145/3323165.3323177.

[19] Paul Erdős. Extremal problems in graph theory. In *Theory Of Graphs And Its Applications, Proceedings of Symposium Smolenice*, pages 29–36. Publ. House Cszechoslovak Acad. Sci., Prague, 1964.

[20] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. ISSAC 2014*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

[21] François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, pages 57–70, 2016. doi:10.1007/978-3-662-53426-7_5.

[22] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1650–1663. IEEE, 2019.

[23] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. *SIAM J. Comput.*, 50(3), 2021. doi:10.1137/16M1097808.

[24] Stephan Holzer and Nathan Pinsker. Approximation of distances and shortest paths in the broadcast congest clique. In *Proc. OPODIS 2015*, 2015. doi:10.4230/LIPIcs.OPODIS.2015.6.

[25] Janne H. Korhonen. Deterministic MST sparsification in the congested clique. *CoRR*, abs/1605.02022, 2016.

[26] Janne H. Korhonen and Jukka Suomela. Towards a complexity theory for the congested clique. In *Proc. SPAA 2018*, pages 163–172, 2018. doi:10.1145/3210377.3210391.

[27] Dean Leitersdorf. *Fast Distributed Algorithms via Sparsity Awareness*. PhD thesis, Technion - Israel Institute of Technology, Israel, 2022. URL: https://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2022/PHD/PHD-2022-09.

[28] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log\log n)$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005. doi:10.1137/S0097539704441848.

[29] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. STOC 2014*, pages 565–573. ACM, 2014.

[30] Krzysztof Nowicki. A deterministic algorithm for the MST problem in constant rounds of congested clique. *STOC 2021*, 2021.

[31] Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners. In *Proc. DISC 2018*, pages 40:1–40:18, 2018. doi:10.4230/LIPIcs.DISC.2018.40.

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

[33] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. doi:10.1007/BF02165411.

[34] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012. URL: http://doi.acm.org/10.1145/2213977.2214056, doi:10.1145/2213977.2214056.