Global vs. s-t Vertex Connectivity Beyond Sequential: Almost-Perfect Reductions and Near-Optimal Separations*

Joakim Blikstad
CWI
Amsterdam, Netherlands
KTH Royal Institute of Technology
Stockholm, Sweden
Blikstad@kth.se

Yonggang Jiang
MPI-INF
Saarbrücken, Germany
Saarland University
Saarbrücken, Germany
yjiang@mpi-inf.mpg.de

Sagnik Mukhopadhyay
University of Birmingham
Birmingham, United Kingdom
s.mukhopadhyay@bham.ac.uk

Sorrachai Yingchareonthawornchai
ETH Zürich
Zürich, Switzerland
Hebrew University of Jerusalem
Jerusalem, Israel
sorrachai.cp@gmail.com

Abstract

A recent breakthrough by [LNPSY STOC'21] showed that solving s-t vertex connectivity is sufficient (up to polylogarithmic factors) to solve (global) vertex connectivity in the sequential model. This raises a natural question: What is the relationship between s-t and global vertex connectivity in other computational models? In this paper, we demonstrate that the connection between global and s-t variants behaves very differently across computational models.

In parallel and distributed models, we obtain almost tight reductions from global to s-t vertex connectivity. In PRAM, this leads to a $n^{\omega+o(1)}$-work and $n^{o(1)}$-depth algorithm for vertex connectivity, improving over the 35-year-old $\tilde{O}(n^{\omega+1})$-work $O(\log^2 n)$-depth algorithm by [LLW FOCS'86], where ω is the matrix multiplication exponent and n is the number of vertices. In CONGEST, the reduction implies the first sublinear-round vertex connectivity algorithm when the diameter is moderately small. This answers an open question in [JM STOC'23].

In contrast, we show that global vertex connectivity is strictly harder than s-t vertex connectivity in the two-party communication setting, requiring $\widetilde{\Theta}(n^{1.5})$ bits of communication. The s-t variant was known to be solvable in $\tilde{O}(n)$ communication [BvdBEMN FOCS'22]. Our results resolve open problems raised by [MN STOC'20, BvdBEMN FOCS'22, AS SOSA'23].

At the heart of our results is a new graph decomposition framework we call common-neighborhood clustering, which can be applied in multiple models. Finally, we observe that global vertex connectivity cannot be solved without using s-t vertex connectivity by proving an s-t to global reduction in dense graphs in the PRAM and communication models.

*The full version is available at https://arxiv.org/abs/2503.20366.

STOC '25, Prague, Czechia

ACM ISBN 979-8-4007-1510-5/25/06
https://doi.org/10.1145/3717823.3718316

CCS Concepts

• **Theory of computation → Distributed algorithms**; **Lower bounds and information complexity**; **Parallel algorithms**; **Graph algorithms analysis**.

Keywords

Algorithmic Graph Theory, Vertex Connectivity, Parallel Computation, Distributed Computation

ACM Reference Format:
Joakim Blikstad, Yonggang Jiang, Sagnik Mukhopadhyay, and Sorrachai Yingchareonthawornchai. 2025. Global vs. s-t Vertex Connectivity Beyond Sequential: Almost-Perfect Reductions and Near-Optimal Separations. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25), June 23–27, 2025, Prague, Czechia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3717823.3718316

1 Introduction

The problem of finding (global) vertex connectivity of a simple undirected graph aims to find the minimum number of vertices whose removal disconnects the graph (or the graph becomes a singleton). This problem, along with a very closely related problem of (global) edge connectivity (which asks for the minimum number of edges to be removed to disconnect the graph), are classic and fundamental graph problems that have drawn the attention of researchers for the last fifty years [8, 12, 18, 19, 23–25, 27, 28, 33, 34, 38, 41, 44, 49, 54, 58]. Both problems can be easily solved by n^2 calls to their s-t variants[1], which is in addition given two vertices s, t and asking the minimum number of vertices (or edges) whose removal disconnects s and t. This simple idea is not efficient for two reasons. Firstly, s-t vertex (or edge) connectivity is known to be equivalent to max flow with unit vertex capacities (or edge capacities), which is a hard problem.[2] Besides, n^2 calls add an even larger polynomial

[1]Throughout this paper, we use n to denote the number of vertices and m to denote the number of edges of the input graph.

[2]While there now exist almost linear time maximum flow algorithms in the sequential setting [16], these algorithms are far from simple. Morover, in other models of computation it remains largely open how efficiently one can solve (even unit-vertex-capacitated) maximum flow.

factor to their s-t variancts running time. Thus, researchers focus on two questions: is there an algorithm to circumvent their s-t variants? Can we reduce the number of calls to their s-t variants?

For edge connectivity, the answer to the first question is affirmative. In the conventional unit-cost RAM (i.e., sequential) model, a long line of work spanning over many decades in the last century was concluded by a near-linear time algorithm [36], which does not require s-t edge connectivity computation, but instead uses a technique called 'tree-packing'. This technique has been proven to be versatile enough to lead to near-optimal algorithms in various computational models [22, 29, 42, 46]. One can view these developments through the lens of the *cross-paradigm algorithm* design—a relatively new research direction where the general theme is to come up with schematic algorithms that can be implemented in several computational models without much trouble [11, 29, 46, 50, 57]. The second question was also answered affirmatively recently by the technique 'isolating cuts' [40, 47], which reduces the number of calls to polylogarithmic.

For the vertex connectivity problem, in the sequential model, the second question was solved very recently [39] which reduces the number of calls to polylogarithmic. However, unlike the tree packing framework for edge connectivity which has been successfully implemented in various models, the reduction in [39] is highly sequential and suffers from several bottlenecks which make it only suitable for the sequential model (we discuss more about these bottlenecks in section 2). Moreover, most the known algorithms for vertex connectivity [8, 18, 19, 24, 27, 34, 39, 41] This is in contrast with the progress on edge connectivity, which is known to be easier to handle than its s-t variant. Thus, a central question this paper aims to answer is:

Is solving s-t vertex connectivity necessary and/or sufficient for solving
(global) vertex connectivity in various computational models?

In this paper, we show a trifecta of such connections:

Sufficiency: We show that s-t vertex connectivity is *sufficient* for vertex connectivity on *parallel and distributed models* by providing an almost-perfect reduction that uses only $n^{o(1)}$ calls. This reduction is built on a new framework using *common-neighborhood clustering*, which is of independent interest.

Insuffiency: In contrast, we show that s-t vertex connectivity is *not sufficient* for vertex connectivity in the *communication model*, by showing a near-optimal lower bound separating the complexity of s-t vertex connectivity and vertex connectivity. We also provide a matching upper bound, thus settling the communication complexity of vertex connectivity.

Necessity: Unlike the case for edge-connectivity, we observe that s-t vertex connectivity is indeed *necessary* for vertex connectivity, at least in dense graphs. This follows from a straightforward reduction from s-t to global vertex connectivity in dense graphs. The reduction works in the PRAM and communication models, but when applied in the CONGEST model it adds additional communication edges. To the best of our knowledge, this simple reduction is novel. This observation implies that any algorithm developed for global vertex connectivity can be adapted to solve s-t vertex connectivity on dense graphs (and hence also unit-vertex-capacitated maximum flow). This explains the lack of vertex connectivity algorithms that work without calling its s-t variant, unlike what is the case for edge connectivity.

1.1 Our Results: Reductions and Separations

Parallel and Distributed Computing. We use the standard PRAM and CONGEST models for parallel and distributed computing respectively (see section 3.2 for the detailed definitions). Our first results are almost perfect reduction algorithms in both models.

Theorem 1.1. *In* PRAM *model, if s-t vertex connectivity can be solved in $W(m,n)$ work[3] and $D(m,n)$ depth where $W(m,n)$ is superadditive[4] on m, then vertex connectivity can be solved in $W(m,n) \cdot n^{o(1)}$ work and $D(m,n) \cdot n^{o(1)}$ depth.*

Theorem 1.2. *In* CONGEST *model, if S-T vertex connectivity[5] can be solved in $R(m,n,D)$ rounds,[6] then vertex connectivity can be solved in $R(m,n,D) \cdot n^{o(1)}$ rounds.*

Our main technical contribution for showing Theorems 1.1 and 1.2 is a new algorithmic framework based on graph decomposition for vertex connectivity. We introduce a graph property called *common-neighborhood* and show that if the graph satisfies this property, then the reduction can be handled in PRAM and CONGEST. For the general graph, we develop an efficient algorithmic framework that decomposes the graph into *common-neighborhood* clusters. The decomposition algorithm, as well as its efficient implementation in both PRAM and CONGEST[7], might be of independent interest.

Our approach is similar to recent advances in graph algorithms, where problems are first solved on graphs with "nice properties" and then extended to general graphs through decomposition techniques (for example, low-diameter decomposition [1, 9, 20, 26], expander decomposition [56, 59, 60], length-constraint expander decomposition [31, 32]) . Certain graph decompositions are also proven to be highly parallelizable [3, 13, 14, 45]; this is also the case for our common-neighborhood clustering.

Two-Party Communication. Given the positive results in the sequential, PRAM, and CONGEST models, one might expect a perfect reduction from vertex connectivity to its s-t variant in other models as well. Perhaps surprisingly, we show that this is not always the case. In particular in the classic *two-party communication* model, we show a lower bound separating the communication complexity of vertex connectivity and s-t vertex connectivity by $\tilde{\Theta}(\sqrt{n})$.[8] We also

[3] We assume all the complexity functions in this paper are reasonably smooth in the sense that $W(Cm, Cn) = \text{poly}(C) \cdot W(m,n)$.

[4] $W(m,n)$ is superadditive on m means $W(m_1+m_2,n) \geq W(m_1,n) + W(m_2,n)$. For example, $W(m,n) = m\sqrt{n}$ is superadditive, while $W(m,n) = n^{\omega}$ would not be superadditive.

[5] S-T vertex connectivity is given two disjoint vertex sets S, T and asking the minimum vertex cut separating S and T. In other models we can simply contract S and T into single vertices. However, in CONGEST, this is not allowed as it would change the underlying communication network. This variant does not incur new challenges as commonly both s-t or S-T vertex connectivity are solved by maximum bipartite matching.

[6] D is the diameter of the input network.

[7] More precisely, our algorithm works in the recently introduced *minor-aggregation model* [57] in $n^{o(1)}$ rounds.

[8] Throughout this paper, we use $\tilde{O}(\cdot)$ to hide $\text{poly}\log n$ factors, and $\widehat{O}(\cdot)$ to hide $n^{o(1)}$ factors.

show a matching upper bound, thus resolving the communication complexity of vertex connectivity up to polylogarithmic factors.

Theorem 1.3. *The two-party communication complexity of vertex connectivity is* $\widetilde{\Theta}\left(n^{1.5}\right)$.

It is known from previous work [10] that the communication complexity of s-t vertex connectivity is $\widetilde{\Theta}(n)$. Thus, Theorem 1.3 provides a strict separation of these two problems. Previously, a lower bound of $\Omega(n^2)$ is only known to hold on multigraph [6].

In contrast, we note that for *edge-connectivity*, seemingly the s-t variant is harder than the global variant: it is known that the global variant admits an $\tilde{O}(n)$ communication protocol [46], while it remains an an open question if this is also the case for the s-t edge-connectivity.

1.2 Our Results: Implications

Implications in PRAM *and* CONGEST. By folklore reductions, s-t vertex connectivity is known to be equivalent to *maximum bipartite matching* (BMM)[9], which is an extensively studied problem. For completeness, we summarize the current state-of-the-art results for BMM in PRAM and CONGEST in the full version. By combining with our reduction Theorems 1.1 and 1.2, we directly get the following results.

Theorem 1.4. *In* PRAM *model, vertex connectivity can be solved in* $mn^{0.5+o(1)}$ *work and* $n^{0.5+o(1)}$ *depth.*

Theorem 1.5. *In* CONGEST *model, vertex connectivity can be solved in* $n^{o(1)} \cdot (n^{3/8}D^{1/2} + n^{1/2}D + n^{7/10}D^{7/10})$ *rounds, which is sublinear as long as* $D = n^{3/7-\epsilon}$ *for some constant* $\epsilon > 0$.

The superadditive requirement in Theorem 1.1 might prevent us from reduction to BMM algorithm in matrix multiplication work and depth [37, 43, 48]. However, although not directly implied by Theorem 1.1, our reduction framework is versatile enough to obtain vertex connectivity in matrix multiplication work and depth as well.

Theorem 1.6. *In* PRAM *model, vertex connectivity can be solved in* $n^{\omega+o(1)}$ *work and* $n^{o(1)}$ *depth, where* n^{ω} *is the optimal work for matrix multiplication in* $n^{o(1)}$ *depth.*

Remark 1.7. The $n^{\omega+o(1)}$ work and $n^{o(1)}$ depth can be improved to $\widetilde{O}(n^{\omega})$ work and $\widetilde{O}(1)$ depth if we apply a common-neighborhood cover algorithm with $\widetilde{O}\left(n^2\right)$ work and $\widetilde{O}(1)$ depth instead of $\widetilde{O}(m)$ work and $n^{o(1)}$ depth. However, for consistency throughout the paper across different models, we do not optimize this here.

Previous Work. Parallel vertex connectivity was mostly studied for small κ (the vertex connectivity of the input graph) [17, 19, 30], which have at least $\Omega(n)$ depths for large κ. Despite the long-known fact that BMM can be solved in matrix multiplication work and depth [37, 43, 48], the only known sublinear depth algorithm for vertex connectivity in the general case is by reduction to matrix multiplication [41] in $\widetilde{O}\left(n^{\omega+1}\right)$ work and $O(\log^2 n)$ depth. Our result Theorem 1.6 improves this 35-year-old result in the subpolynomial depth regime to almost matrix multiplication time. Moreover, in the sublinear depth regime, our result Theorem 1.4 is faster than matrix multiplication time on moderately sparse graphs.

In the CONGEST model, vertex connectivity was either studied for small κ [35, 52, 53, 55, 61], or by $O(\log n)$ approximation [12]. For exact computation and large κ, all of the mentioned results cost at least $\Omega(n)$ rounds, even for a graph with constant diameter. Our result Theorem 1.5 gives the first sublinear round algorithm for exact large vertex connectivity as long as D is moderately small.

Implications for Streaming. We use the standard graph streaming model (formally defined in section 3.2). It is a well-known fact that an efficient streaming protocol yields an efficient communication protocol. Thus, our communication lower bound in Theorem 1.3 immediately implies the following streaming lower bound.

Theorem 1.8 (Streaming lower bound). *Any* $P(n)$*-pass randomized streaming algorithm of vertex connectivity needs* $\Omega(n^{1.5}/P(n))$ *space.*

Our algorithmic ideas can be used in the streaming model as well and give the following reduction theorem. In the randomized streaming model, we say a reduction from vertex connectivity to maximum bipartite matching is an (α, β)-reduction if vertex connectivity can be solved in $\alpha S(n)$ space and $\beta P(n)$ passes as long as maximum bipartite matching can be solved in $S(n)$ space and $P(n)$ passes.

Theorem 1.9 (Streaming upper bound). *In the randomized streaming model, there is a* $(\widetilde{O}\left(n^{0.5}\right), O(1))$*-reduction from vertex connectivity to maximum bipartite matching.*

A natural question is whether the reduction of Theorem 1.9 is optimal or not. Although we cannot show a lower bound, there is a simple observation that any better reduction gives a strong streaming lower bound for maximum bipartite matching.

Corollary 1.10. *If an* (α, β)*-reduction from vertex connectivity to maximum bipartite matching exists for* $\alpha \cdot \beta < n^{0.5-\epsilon}$*, then in the semi-streaming model, maximum bipartite matching cannot be solved in* $o(n^{\epsilon})$ *passes.*

The implied BMM lower bound of corollary 1.10 would be a breakthrough in semi-streaming lower bound [2, 5, 15]. Hence, if we are to believe the conjecture that maximum bipartite matching can be solved in $\widetilde{O}(n)$ space and $n^{o(1)}$ passes, then according to corollary 1.10, the reduction of Theorem 1.9 can not be improved by a polynomial factor. Alternatively, corollary 1.10 can be viewed as a new potential way to obtain polynomial pass semi-streaming lower bound for bipartite matching through developing better reductions for vertex connectivity.

At last, by combining Theorem 1.9 with the current progress on BMM [4], we get the following result.

Corollary 1.11. *There is a randomized streaming algorithm for vertex connectivity with* $\widetilde{O}\left(n^{1.5}\right)$ *space and* $n^{0.75+o(1)}$ *passes.*

2 Technical Overview

Given an undirected graph $G = (V, E)$, a partition (L, S, R) of V (where L, S, R are all non-empty) is a *vertex cut* (or simply *cut* in this work) if there are no edges between L and R. The size of the cut (L, S, R) is defined as $|S|$, and we say S is a *separator* or more

[9] They are equivalent for the value version, i.e., outputting the size of the cut and the size of the maximum matching (see in the full version).

precisely, (s,t)-separator for any $s \in L, t \in R$. We say a cut (L,S,R) is a *t-sink cut* if $t \in R$. Throughout this paper, we will focus on solving the following single-sink version of vertex connectivity.

Definition 2.1 (SSVC (single-sink vertex connectivity))**.** *Given an undirected graph $G = (V,E)$ and a vertex $t \in V$, outputs a minimum t-sink cut.*

There is a simple randomized reduction from the general vertex connectivity problem to SSVC (we defer to the full version for the reduction), and fixing a sink makes the algorithm easier to state. So we focus on the single-sink version.

Another useful observation is that to solve SSVC, it suffices to solve the following Min-Neighbor problem. Given a vertex $u \in V$, define $N_G(u)$ (or $N(u)$ when the context is clear) to be the set of neighboring vertices of u, i.e., vertices that share an edge with u. Let $N[u] := N(u) \cup \{u\}$. For a vertex set $V' \subseteq V$, define $N(V') = (\cup_{u \in V'} N(u)) - V'$ and $N[V'] = N(V') \cup V'$.

Definition 2.2 (Min-Neighbor)**.** *Given an undirected graph $G = (V,E)$ and a vertex set $V' \subseteq V$, find $L \subseteq V'$ such that $|N_G(L)|$ is minimized.*

If we have an algorithm for Min-Neighbor, to solve SSVC, we set $V' = V - N[t]$ and call the Min-Neighbor algorithm to get $L \subseteq V'$ such that $|N(L)|$ is minimized. One can verify that $(L, N(L), V - L - N(L))$ is a valid t-sink cut for any $L \subseteq V'$. Moreover, every minimum t-sink cut (L,S,R) satisfies that $L \subseteq V'$ and $S = N(L)$ (otherwise we can set $S = N(L)$ which decrease the size of the cut).

We can also show the reversed reductions by simple arguments, so Min-Neighbor is, essentially, equivalent to vertex connectivity. Throughout this overview, we focus on solving Min-Neighbor.

Bottlenecks of [39] beyond the sequential model. We first discuss the bottlenecks of implementing the sequential algorithm of [39] in PRAM, CONGEST, and two-party communication models.

(1) The algorithm is inherently sequential in the sense that it uses a breadth-first-search with unbounded depth as a subroutine. This makes it hard to be implemented in situations where parallelism is a requirement, such as PRAM and CONGEST.
(2) Very informally, the algorithm reduces a vertex connectivity instance on a graph G with n vertices and m edges to a bipartite maximum matching instance on a graph G' with $\widetilde{O}(m)$ edges and $\widetilde{O}(n^2)$ vertices. In other words, when n is the parameter, the reduction is not efficient. In many situations, n is the parameter for measuring the complexity, such as CONGEST and communication, or in the matrix multiplication work of PRAM.

As mentioned in the introduction, we develop a new framework based on common-neighborhood clustering to sidestep these bottlenecks. The organization of this overview is as follows.

(1) In section 2.1, we define the common-neighborhood property of a graph, and show how to reduce Min-Neighbor to s-t vertex connectivity preserving the number of edges (but not vertices) in a *common-neighborhood cluster* (defined in section 2.1). The algorithm is described in PRAM model for the sake of simplicity.
(2) In section 2.2, we show our decomposition algorithm for common-neighborhood clustering in PRAM, which can also be implemented efficiently in various models like CONGEST and communication. Common-neighborhood clustering is a way to reduce solving Min-Neighbor on a general graph to solving Min-Neighbor on common-neighborhood clusters. Combined with section 2.1, this proves Theorem 1.1 (not preserving the number of vertices is fine here because we assumed $W(m,n)$ is superadditive on m in Theorem 1.1).
(3) For the CONGEST model, to prove Theorem 1.2, we do not have the guarantee that $R(m,n,D)$ is superadditive. Thus, it suffers from the second bottleneck of [39]. We show that it can be resolved nearly perfectly in section 2.3 and prove Theorem 1.2.
(4) To prove that vertex-connectivity can be solved in matrix multiplication work in PRAM (Theorem 1.6), only preserving the number of edges does not suffice. We show how to leverage our framework to circumvent this bottleneck in section 2.4.
(5) Section 2.5 gives an overview of our results in the two-party communication setting (Theorem 1.3). We show that the number of vertices cannot be perfectly preserved by proving a near-optimal lower bound for vertex connectivity separating it from s-t vertex connectivity. The near-optimality of the lower bound is shown by an upper bound.

2.1 Building the Framework: Common-Neighborhood Cluster

Recall that the input to Min-Neighbor is an undirected graph $G = (V,E)$ and a vertex set $V' \subseteq V$. Throughout this overview, we always use $L^\star$ to denote an arbitrary one of the minimizers of $\min_{L' \subseteq V'} |N(L')|$. Wlog., we will assume $G[L^\star]$ is connected, otherwise we can take a connected component of $G[L^\star]$ denoted as L', we have $N(L') \subseteq N(L^\star)$ so we can use L' as the minimizer instead.

Our goal is to find $L^\star$, and let us assume that we know its size.[10] Now we are ready to define common-neighborhood clusters. For two sets A, B, define $A \triangle B = (A - B) \cup (B - A)$ as their *symmetric difference*.

Definition 2.3. *Given an undirected graph $G = (V,E)$ and a vertex set $C \subseteq V$, we say C has* neighborhood difference *d if for any $u, v \in C$, $|N(u) \triangle N(v)| \le d$.*

We say C is a *common-neighborhood cluster* if C has neighborhood difference at most $\widehat{O}(|L^\star|)$. In this section, we will assume V' is a common-neighborhood cluster and show a PRAM algorithm solving Min-Neighbor. We will use the following property of a common-neighborhood cluster.

Lemma 2.4 (Informal)**.** *If the input vertex set V' is a common-neighborhood cluster, then one of the following two cases happens.*

(1) $|N_G(L^\star) \cap V'| = \widehat{O}(|L^\star|)$.
(2) $G[V']$ is almost a clique. For this overview, assume V' is indeed a clique for convenience.

[10] This assumption can be easily removed by guessing the size of $L^\star$ by powers of 2 which only adds a logarithmic factor to the complexity.

Now we show the rough idea of how to find $L^\star$ in each case. The formal proofs are in section 5.

Case 1: $|N(L^\star) \cap V'| = \widehat{O}(|L^\star|)$. In this case, we can employ the *isolating cuts* technique [39], that, given an independent set T, finds the minimum vertex cut separating exactly one node in T from the other nodes in T. For our case, we let T be an independent sample by including each node $v \in V'$ in T with probability $1/\widehat{O}(|L^\star|)$. This means that with probability $\widehat{\Omega}(1)$, T has exactly one node in $L^\star$ and no nodes in $N(L^\star)$.

Case 2: $G[V']$ *is a clique.* In this case, we are trying to solve Min-Neighbor given that (i) V' is a clique, and (ii) every two nodes in V' has their neighbors in G differ by at most $\widehat{O}(|L^\star|)$ nodes. We call this problem MinNeighbor-NearClique-CommonNeighborhood, or MinNNCC for short.

For a vertex $s \in V'$, define the *s-source* Min-Neighbor problem as finding $\arg\min_{s \in L' \subseteq V'} N(L')$. Note that it can be solved in one s-t vertex connectivity call: add a super node t connecting to $N(V')$ and call s-t vertex connectivity. Given this, the algorithm for dealing with Case 2 contains two steps.

(1) Sample $|V'|/|L^\star|$ nodes uniformly at random so that one of them is in $L^\star$.
(2) For every sample node, solve s-source Min-Neighbor on a *sparsified graph* (which we will define later) with $|V'| \cdot |L^\star|$ edges.

The cumulative number of edges for calling s-t vertex connectivity is $|V'|^2$, which is proportional to the number of edges in G (since V' is a clique). Thus, the reduction preserves the total number of edges. Moreover, if the work function $W(m,n)$ for solving s-t connectivity is superadditive on m, then the work of solving MinNNCC can be written as $O(W(|V'| \cdot |L^\star|, n)) \cdot |V'|/|L^\star| \le O(W(|V'| \cdot |L^\star| \cdot (|V'|/|L^\star|), n)) = O(W(m,n))$.

The remaining question is how to run s-source Min-Neighbor on a *sparsified graph with* $|V'| \cdot |L^\star|$ *edges*. This is because we can delete all edges from $V' - \{s\}$ to neighbors of s, and deleting them will not change the set $N(L')$ for any set $L' \ni s$. Recall that V' has neighborhood difference at most $\widehat{O}(|L^\star|)$, and hence $u \in V'$ can have at most $\widehat{O}(|L^\star|)$ remaining edges outside $N(s)$.

Building the Framework. To summarize, once we have a common-neighborhood cluster V', the problem reduces to one of (i) finding a minimum isolating cut, or (ii) solving MinNNCC. If V' is not a common-neighborhood cluster, the natural idea is to decompose it into common-neighborhood clusters, which we call *common-neighborhood clustering* defined as follows.

Definition 2.5 (Common-neighborhood clustering)**.** *Given $G = (V,E)$ and $V' \subseteq V$, output a set of vertex sets $\mathcal{C}$ such that*

(1) each vertex $u \in V'$ is contained it at most $\widehat{O}(1)$ clusters in $\mathcal{C}$,
(2) for any $C \in \mathcal{C}$, C is a common-neighborhood cluster and $C \subseteq V'$,
(3) $L^\star \subseteq C$ for some $C \in \mathcal{C}$ with probability at least $1/n^{o(1)}$.

Our framework can be thus summarized as follows.

Step 1: Compute a *common-neighborhood clustering* denoted as $\mathcal{C}$.
Step 2: For every $C \in \mathcal{C}$, solve *minimum isolating cut* and MinNNCC to get non-empty $L_C \subseteq C$ minimizing $N(L_C)$.
Step 3: Output the minimum $N(L_C)$ among all $C \in \mathcal{C}$.

2.2 A Decomposition Algorithm for Common-Neighborhood Clustering

In this section, we show a PRAM algorithm for common-neighborhood clustering which can be implemented in different models. One crucial property of $L^\star$ that we state here is that $L^\star$ has neighborhood difference at most $2|L^\star|$—we defer the formal proof of this statement to lemma 3.1.

A proof of existence. We first give a simple, but inefficient, algorithm that shows the existence of common-neighborhood clustering. Define the neighborhood-difference graph $G' = (V', E')$ as follows: for any pair of vertices $(u,v) \in V' \times V'$, $(u,v) \in E'$ if and only if $|N_G(u) \triangle N_G(v)| \le 2|L^\star|$.

We also need the notion of the sparse neighborhood cover that is studied in the literature [7]. We define it as follows.

Definition 2.6 (Sparse neighborhood cover (SNC))**.** *Given an undirected graph $G = (V,E)$, the* sparse neighborhood cover *of G is a set of vertex sets $\mathcal{C}$ (called clusters) such that*

(1) (sparse) each vertex $v \in V$ is contained in at most $\widetilde{O}(1)$ clusters in $\mathcal{C}$,
(2) (low diameter) the diameter of $G[C]$ for any $C \subseteq \mathcal{C}$ is at most $O(\log n)$,
(3) (cover) for every $v \in V$, there exists a cluster $C \subseteq \mathcal{C}$ such that $N_G[v] \subseteq C$.

It is also proved in [7] that SNC can be constructed efficiently. We claim that applying SNC on G' gives us a common-neighborhood clustering. Let us verify each property of common-neighborhood clustering one by one.

(1) Each vertex $u \in V'$ is contained in at most $\widetilde{O}(1)$ clusters in $\mathcal{C}$ according to the *(sparse)* condition.
(2) For any $C \in \mathcal{C}$ and any $u, v \in C$, there exists a path from u to v in G' with length $O(\log n)$ according to *low diamter*, denoted by $(u, v_1, v_2, ..., v)$. Notice that an edge $(u, v_1) \in E'$ means we can delete and add $2|L^\star|$ nodes from $N_G(u)$ to get $N_G(v_1)$. We can repeat this argument along the path, which shows that $|N_G(u) \triangle N_G(v)| = O(|L^\star| \log n)$. This proves that C has neighborhood difference at most $O(|L^\star| \log n)$, i.e., it is a common-neighborhood cluster.
(3) $L^\star$ has neighborhood difference at most $2|L^\star|$ according to lemma 3.1, $L^\star$ must be a clique in G' according to the definition of G' (actually here we do not require $G[L^\star]$ to be connected, this additional property is to assist a faster algorithm which we will explain later). Thus, take an arbitrary $v \in L^\star$, and note $L^\star \subseteq N_{G'}[v]$, which means $L^\star$ is contained in some cluster due to property *(cover)*.

This completes the existential proof. However, construction G' is inefficient and runs in $O(n^3)$ time in a trivial way. □

Speeding up the construction of G': approximating the neighborhood differences. One reason for the slow computation of G' is that to decide if an edge (u,v) exists in G' or not, we need to go over the neighbors of both u and v, which in the worst case use time n, resulting in a total time $O(n^3)$. This is unavoidable if we want

to know the exact size of $|N_G(u) \triangle N_G(v)|$. However, we will show that an approximate size of $|N_G(u) \triangle N_G(v)|$ suffices.

We can build G' in the following way: (i) if $(u, v) \in E'$, then $|N_G(u) \triangle N_G(v)| \leq 3|L^\star|$, (ii) if $(u, v) \notin E'$, then $|N_G(u) \triangle N_G(v)| > 2|L^\star|$. Such G' can be constructed by only knowing an approximate value of $|N_G(u) \triangle N_G(v)|$ for any $u, v \in V$. One can verify that applying SNC on this G' does not make too much difference and can still give us a common-neighborhood clustering. Moreover, computing an approximate value of $|N_G(u) \triangle N_G(v)|$ only requires $\widetilde{O}(1)$ time by standard sampling and sketching techniques. This is proved in corollary 3.4.

Assumption 2.7. In the rest of this overview, we use $\mathrm{apd}(u, v)$ to denote a 1.1-approximation value of $|N_G(u) \triangle N_G(v)|$, which can be found using corollary 3.4.

Thus, G' can be constructed in time $\widetilde{O}(n^2)$. However, this is still slow if our goal is $\widetilde{O}(m)$ work. More severely, computing G' is not a 'localized' procedure since it requires two very far away nodes to decide whether they have an edge in G' or not, posing a challenge for distributed computing. Thus, our algorithm will not compute G', or an SNC of G', but instead involves an entirely new idea.

A fast algorithm with large depth. Now we describe an algorithm that has a large depth but in $\widetilde{O}(m)$ work in PRAM. Given that $L^\star$ has neighborhood difference at most $2|L^\star|$ and $G[L^\star]$ is connected, if we know a vertex $u \in L^\star$, we can construct a common-neighborhood cluster C with $L^\star \subseteq C$ in the following way: build a BFS tree on G with root u, such that a vertex v is included in the BFS tree only if $\mathrm{apd}(u, v) \leq 3|L^\star|$. Let us denote the vertices in this BFS tree as $B'\left[3|L^\star|\right](u)$. It can be equivalently defined as follows.

Definition 2.8. *Define $B'[\ell](u)$ as the largest possible set such that (i) $G[B'[\ell](u)]$ is connected, (ii) for any $v \in B'[\ell](u)$ we have $apd(u, v) \leq \ell$.*

In this way, we can guarantee that the neighborhood difference of C is at most $O(|L^\star|)$, $L^\star \subseteq C$, and the construction time of C is proportional to the number of adjacent edges of C.

Forgetting what $L^\star$ is. In general, we do not know a vertex in $L^\star$. A natural idea is to sample vertices with probability $1/|L^\star|$ and run the above algorithm for each sampled node and add all the resulting clusters into $\mathcal{C}$. In this way, with good probability one of the clusters in $\mathcal{C}$ will contain $L^\star$. However, it is easy to see that the *(sparse)* property does not hold, i.e., each vertex could appear in as many as $n/|L^\star|$ clusters (this also results in a large work). The reason for the large overlap is that clusters can intersect with each other. A naive way to reduce the overlap would be to delete every cluster that intersects with others. Unfortunately, this does not work because the cluster that contains $L^\star$ could be deleted by this procedure. Hence, we set our goal as follows: sample some vertices u and build the cluster $B'[\widehat{O}(|L^\star|)](u)$, such that (i) at least one sampled vertex is in $L^\star$, and (ii) the cluster containing $L^\star$ does not intersect other clusters. If these conditions are satisfied, we can let $\mathcal{C}$ be all the clusters that do not intersect with others, and we will be done. The work is also guaranteed to be $\widetilde{O}(m)$ since if two BFS trees meet together at one node, they can stop growing and they are both excluded from being in $\mathcal{C}$.

To achieve the aforementioned non-overlap requirement (ii), an ideal situation would be the following. There are two vertex sets A, B with $L^\star \subseteq A \subseteq B$ such that:

(1) the size of B is roughly equal to the size of A,
(2) B has neighborhood difference at most $\widehat{O}(|L^\star|)$, and
(3) Any vertex outside B has large neighborhood difference from the vertices in A (e.g., at least $10d_A$ where the neighborhood difference of A is d_A).

If we have such A and B, then we can sample with probability $1/|B|$ such that, with good probability, one of the sampled nodes will be in A while all other sampled nodes are outside B. Now we can build $B'[2d_A](u)$ for each sampled node u, and A will be contained in one of the clusters, and the cluster containing A will not overlap with other clusters.

Now we discuss how to prove the existence of such A, B. For a vertex $v \in L^\star$, $L^\star \subseteq B'[3|L^\star|](v)$. In what follows we assume v is an arbitrary vertex in $L^\star$. If we can find a parameter $r > 2$ such that $|B'[99r|L^\star|](v)| \leq n^{o(1)}|B'[r|L^\star|](v)|$ holds, then we can set $A = B'[r|L^\star|](v)$ and $B = B'[99r|L^\star|](v)$ and all the three conditions for A and B hold. Our goal is to find the smallest such r starting from $r = 2$ and increasing r to $99r$ iteratively. Notice that each repetition increases the size of $B'[r|L^\star|](v)$ by a factor of $n^{o(1)}$ but only increases r by a factor of 99, by setting the parameter correct we can make sure $r = \widehat{O}(1)$ in the end. This proves the existence of A and B. As long as A, B exists, we can guess the size of B by powers of 2, so that we can sample with probability $1/|B|$ as the algorithm suggests.

A side remark is that the algorithm only guarantees $L^\star \subseteq C \in \mathcal{C}$ with probability $1/n^{o(1)}$. This probability can be boosted by repeating the algorithm $n^{o(1)}$ and add all the found clusters to $\mathcal{C}$.

Reducing the depth by graph shrinking. Unfortunately, the above idea still requires BFS with unbounded depth, which is unsuitable for parallel and distributed models. The main bottleneck is about finding $B'[\ell](u)$ for many different u and some parameter ℓ. If $B'[\ell](u)$ contains less than $n^{o(1)}$ nodes, then it can be done in $n^{o(1)}$ depth since the BFS can include at most $n^{o(1)}$ nodes. Thus, let us assume $B'[\ell](u)$ contains much more than $n^{o(1)}$ nodes.

The idea is to shrink the graph size by a factor of $n^{o(1)}$ while not losing too much information about $B'[\ell](u)$. We sample vertices with probability $1/n^{o(1)}$ and denote the sampled vertex set as X. With good probability we have (i) $X \cap B'[\ell](u) \neq \emptyset$, (ii) size of X decrease the size of V by a factor of $1/n^{o(1)}$. We decompose the graph by growing vertex disjoint BFS trees simultaneously from every vertex $v \in X$, such that a vertex w is included in the BFS tree of v only if $\mathrm{apd}(v, w) \leq 3\ell$. After that, we contract each BFS tree and call them *super nodes*. The graph size decreases by a factor of $1/n^{o(1)}$. Moreover, all vertices in $B'[\ell](u)$ are preserved in some super nodes since X contains at least one node $v \in B'[\ell](u)$ and all nodes in $B'[\ell](u)$ are eligible to join the BFS tree of v.

We can repeat the above operation as long as the number of super nodes intersecting $B'[\ell](u)$ is at least $n^{o(1)}$: sample super nodes with probability $1/n^{o(1)}$ to make sure one sampled super node intersects $B'[\ell](u)$; from each super node v grows a BFS tree only including another supernode w such that there exist original nodes $v' \in v, w' \in w$ with $\mathrm{apd}(v, w) \leq 3\ell$; contract each BFS tree.

Notice that the neighborhood difference of each super node grows by a constant factor in each iteration.

We repeat until the number of super nodes intersecting $B'[\ell](u)$ becomes $n^{o(1)}$, then we can find all of them by a simple BFS of depth $n^{o(1)}$. The only problem is that a super node might contain extra nodes besides nodes in $B'[\ell](u)$. This is fine due to the following reason: in each iteration the neighborhood difference of each super node grows by a constant factor, while the number of nodes of the graph shrink by a factor of $1/n^{o(1)}$; by setting the parameter correctly, the final neighborhood difference of each super node is at most $n^{o(1)}\ell$. Thus, all the nodes in the super modes intersecting $B'[\ell](u)$ are contained in $B'[n^{o(1)}\ell](u)$. This $n^{o(1)}$ factor is not too large and a slight change in the algorithm can handle it.

The algorithm and analysis are described in details in section 4.

2.3 Distributed Algorithms

Recall the framework for our CONGEST algorithms: we first apply common-neighborhood cluster described in section 2.2 so that we can assume V' is a common-neighborhood cluster as defined in section 2.1, then we use the observations in section 2.1 to reduce the problem to (i) minimum isolating cut, and (ii) MinNNCC. Both problems can be reduced to s-t vertex connectivity while preserving the total number of edges m when doing the reduction, as explained in section 2.1.

Now, we describe the key obstacles to implementing distributed CONGEST and outline our solution to overcome the technical obstacles.

2.3.1 Bottlenecks of Proving Theorem 1.2: Minimum Isolating Cut. The following isolating cuts lemma can almost handle our case.

Lemma 2.9 (Lemma 4.2 of [39]). *Given a graph $G = (V, E)$ and an independent set $T \subseteq V$ of size at least 2, there is an algorithm that outputs for each $v \in T$ a $(v, T-\{v\})$-min-separator C_v. The algorithm makes calls to s-t vertex mincut on graphs with $O(m)$ total number of vertices and $O(m)$ total number of edges and takes $\tilde{O}(m)$ additional time.*

This version of the isolating cuts lemma works perfectly fine for sequential settings as they focus on bounding the total number of edges. Their algorithm can also be easily parallelized. However, the number of vertices can be as large as $O(m)$ which is not suitable for distributed computing.

To handle this situation, we prove a more efficient version of the isolating cuts lemma that also guarantees $\tilde{O}(n)$ total number of vertices and $O(m)$ total number of edges.

Lemma 2.10. *Given a graph $G = (V, E)$ and an independent set $T \subseteq V$ of size at least 2, there is an algorithm that outputs for each $v \in T$ a $(v, T - \{v\})$-min-separator C_v. The algorithm makes calls to s-t vertex mincut on graphs with $\tilde{O}(n)$ total number of vertices and $O(m)$ total number of edges and takes $\tilde{O}(m)$ additional time.*

To elaborate for minimum isolating cut, by following the proof of Lemma 2.9 (formally described in Lemma 4.2 of [39]), we remove a set of vertices X from the graph so that each connected components C of the remaining graph contains at most one terminal $t \in T$. For each connected component C with one terminal $t \in T$ inside, we try to find $\min_{s \in L' \subseteq C} N_G(L')$. Notice that this minimization problem only depends on all the edges adjacent to C, which are non-overlapping for different C, so the total number of edges is bounded by m. However, it depends on all vertices in $N_G[C]$, which could intersect at lot for different C.

To obtain the improvement as in Lemma 2.10, we briefly explain how to deal with this issue. For every C where we wish to find $\min_{s \in L' \subseteq C} N_G(L')$, we prove that it suffices to keep $\widetilde{O}(|C|)$ vertices in $N_G(C)$ and deleting the others, while preserving $\min_{s \in L' \subseteq C} N_G(L')$. This is done by finding a maximum bipartite matching between C and $N_G(C)$. After that, the total number of vertices is upper bounded by $\sum_C \tilde{O}(|C|) = \tilde{O}(n)$. Given Lemma 2.10, we can easily obtain PRAM and CONGEST algorithms where the number of vertices involved in the isolating cuts lemma is $\tilde{O}(n)$.

2.3.2 Bottlenecks of Proving Theorem 1.2: Solving MinNNCC. It is easy to see that the algorithm described in section 2.1 for MinNNCC does not preserve the number of edges. In fact, it contains $|V'|/|L^\star|$ many instances, where each instance contains $|V'| \cdot |L^\star|$ edges and n vertices.

Now we briefly explain how this is solved in CONGEST. We will use the fact that V' is a clique. For each instance with $|V'|/|L^\star|$, we map those edges uniformly at random into the clique $G[V']$, and we solve the instance by an oracle call to s-t vertex connectivity where communication happens only on the mapped edges. Since there are in total at most $\widetilde{O}(|V'|^2)$ edges, and $G[V']$ is a clique, in expectation each edge is only included in $\widetilde{O}(1)$ many instances.

Random mapping of the communication inside the clique. Here we elaborate on the random mapping procedure. Our solution is to offload the communication of each instance to the cliques by a simple random load-balancing strategy.

Given each instance G'_s, which is the sparsified graph with $\widetilde{O}(|V'| \cdot |L^\star|)$ edges as described in section 2.1, we define a random (public) function $f_s : N(V') \to V'$ that maps a node $v \in N(V')$ to a random node in V'. For each $u \in N(V')$, our goal is to have $f(u) \in V'$ simulate u (in other words, $f(u)$ acts as a proxy for u), i.e., $f(u)$ acts as if it is node u. In order for $f(u)$ to simulate u, $f(u)$ must learn the following information:

- The algorithm description of node u for executing s-t vertex mincut algorithm in G'_s. This can be done by having u send its code to $f(u)$.
- $f(u)$ learns every node $N_G(u) \cap V'$. This can be done by letting every neighbor $v \in N_G(u) \cap C$ send its id to $f(u)$.

In this case, $f(u)$ can act as if it is node u. So, any communication between $v \in V'$ to $u \in N(V')$ can be moved to the same communication between $v \in V'$ and $f(u) \in V'$. Therefore, the communication between every edge (v, u) where $v \in C, u \in N(V')$ will map to a $(v, f(u))$ where u is a random node. Using the fact of graph G'_s that every node $v \in C - \{s\}$ has at most $\widehat{O}(|L^\star|)$ edges to $N(V')$, we conclude that every node $v \in C - \{s\}$ in the instance G'_s communicates using random $\widehat{O}(|L^\star|)$ edges adjacent to V'.

In summary, fix a node $v \in V'$, and the algorithm will communicate from v to $\tilde{O}(|L^\star|)$ random vertices in V' for each instance G'_s. Since every $\tilde{O}(\frac{|V'|}{|L^\star|})$ instance will use $\tilde{O}(|L^\star|)$ random neighbors from v to in V', and v has degree $|V'| - 1$ inside V', in expectation, the load of each edge in V' incident to v is $\tilde{O}(1)$. This means every

edge e in V' will have $\tilde{O}(1)$ number of instances G'_s use e to communicate. We can now run all instances with $\tilde{O}(1)$ congestion. We present the full algorithm in the full version.

2.4 Parallel Algorithm in Matrix Multiplication Work

Recall that in the last section, we explain how to solve minimum isolating cut and MinNNCC by reductions to s-t vertex connectivity while preserving both the number of edges and vertices, in CONGEST model. For the case of minimum isolating cut, one can easily check that the idea for preserving the number of vertices described in the last section also works in the PRAM model, and thus we can solve s-t vertex connectivity using a reduction to bipartite matching which can be solved in $O(n^\omega)$ work.

We now focus on solving the case of MinNNCC. We will use the same local graph $G_{V'}$. Recall that the algorithm that sparsifies $G_{V'}$ into G'_s can result in $\widehat{O}(m) = O(n^2)$ vertices in total, which is not suitable for our purpose.

In this section, we explain how to solve MinNNCC in matrix multiplication time by utilizing *convex embedding* introduced in [41]. The details can be found in the full version. Throughout this section, we let $n_0 = |N_G[V']| + 1$ be the number of vertices in $G_{V'}$. The goal is to solve MinNNCC in $\tilde{O}(n_0^\omega)$ work (and polylog depth). For simplicity, we assume that $|N(V')| \le |V'|$. Later we show how to remove this assumption.

Let $\mathbb{F}$ be a finite field. For $k \ge 0$, the space $\mathbb{F}^k$ is a k-dimensional linear space over $\mathbb{F}$. Let $X = \{x_1, \ldots, x_n\}$ be a finite set of points within $\mathbb{F}^k$. The *affine hull* of X is defined as

$$\mathrm{aff}(X) = \left\{ \sum_{i=1}^{k} c_i x_i : x_i \in X \text{ and } \sum_{i=1}^{k} c_i = 1 \right\}.$$

The rank of X, represented as $\mathrm{rank}(X)$, is one plus the dimension of $\mathrm{aff}(X)$. Specifically, if $\mathbb{F} = \mathbb{R}$, we refer to the *convex hull* of X, denoted $\mathrm{conv}(X)$. For any sets V, W, any function $f : V \to W$, and any subset $U \subseteq V$, we denote $f(U) := \{f(u) : u \in U\}$.

Definition 2.11 (Convex X-embedding [41]). *For any $X \subset V$, a convex X-embedding of a graph $G = (V, E)$ is a function $f : V \to \mathbb{R}^{|X|-1}$ such that for each $v \in V \setminus X$, $f(v) \in conv(f(N_G(v)))$.*

We defer the details of construction to the full version. For now, let us assume that we are given $N_{G_{V'}}(t)$-convex embedding f_t in graph $G_{V'}$ such that for all $x \in V'$, $\kappa_{G_{V'}}(s,t) = \mathrm{rank}(f_t(N_{G_{V'}}(x)))$. The task now is to compute the rank of $f_t(N_{G_{V'}}(x))$, which can be done in $\tilde{O}(n_0^\omega)$ time. In total, it would take $\tilde{O}(\frac{|V'|}{|L^\star|} \cdot n_0^\omega)$ work to compute $\kappa_{G_{V'}}(s,t)$ for all instances of s-source Min-Neighbor via this approach.

Here is the speedup we can exploit: Since V' has neighborhood difference at most $\widehat{O}(|L^*|)$, and thus for any $s, s' \in V'$, $|N(s) \triangle N(s')| \le \widehat{O}(|L^*|)$. Therefore, $\mathrm{rank}(f_t(N_{G_{V'}}(s')))$ can be obtained from *low-rank* updates of $\mathrm{rank}(f_t(N_{G_{V'}}(s)))$ given we preprocess the matrix representing $f_t(N_{G_{V'}}(s))$. Here, low-rank updates correspond to changing $\widehat{O}(|L^*|)$ columns of the matrix representing $f_t(N_{G_{V'}}(s))$ since $|N(s) \triangle N(s')| \le \widehat{O}(|L^*|)$.

In terms of low-rank updates, we can use a *dynamic matrix data structure* to support this operation. Namely, we can preprocess the matrix M_s representing $f_t(N_{G_{V'}}(s))$ in $\tilde{O}(n_0^\omega)$ time so that given $s' \in V'$, we can decide if $\mathrm{rank}(f_t(N_{G_{V'}}(s'))) \ge k$ in $T_{\mathrm{MM}}(|L^\star|, n_0, |L^\star|)$ time where $T_{\mathrm{MM}}(n,k,r)$ be the number of field operations needed to multiply an $n \times k$ matrix with a $k \times r$ matrix in $O(\log^2 n)$ depth.

To summarize, using the matrix data structure, the total work to compute $\kappa_{G_{V'}}(s,t)$ for $\tilde{O}(\frac{|V'|}{|L^\star|})$ instances of s-source Min-Neighbor is at most

$$\tilde{O}(n_0^\omega) + \tilde{O}\left(\frac{|V'|}{|L^\star|}\right) \cdot T_{\mathrm{MM}}(|L^\star|, n_0, |L^\star|) \le \tilde{O}(n_0^\omega) = \tilde{O}(|V'|^\omega).$$

2.5 Communication Complexity

When implementing our framework in the communication model, common-neighborhood clustering (as described in section 2.3) works well. The problem we face is solving MinNNCC: the best s-t vertex connectivity algorithm [10] takes n as the parameter, but in the worst case we need to run n sparse instances, each with n vertices. One might hope that there is a way to solve this issue with an almost-perfect reduction just like we did in CONGEST. However, we proved that this is not true by showing an $\Omega(n^{1.5})$ lower bound.

An $\Omega(n^{1.5})$ lower bound. We will reduce the following problem to vertex connectivity. Alice and Bob will get $\sqrt{n}$ instances of *subset-tribes*, each containing $\sqrt{n}$ instances of *subset* problems with length $\sqrt{n}$. The inputs to Alice and Bob are denoted as $A_1^{(i)}, A_2^{(i)}, ..., A_{\sqrt{n}}^{(i)} \subseteq [\sqrt{n}]$ and $B_1^{(i)}, B_2^{(i)}, ..., B_{\sqrt{n}}^{(i)} \subseteq [\sqrt{n}]$ from $i = 1$ to $i = \sqrt{n}$, where it is guaranteed that $|A_j^{(i)}| = \sqrt{n}/2$ for any i, j. They want to determine whether there exists i such that $B_j^{(i)} \subseteq A_j^{(i)}$ for any j. This problem has randomized communication lower bound $\Omega(n^{1.5})$. Now we show how to reduce this problem to vertex connectivity.

We let $G = (V_1 \cup V_2 ... \cup V_{\sqrt{n}} \cup U, E)$ where V_i is a clique with $\sqrt{n}$ vertices, and U is a clique with n vertices. We connect V_i to U based on $A_1^{(i)}, ..., A_{\sqrt{n}}^{(i)}$ and $B_1^{(i)}, ..., B_{\sqrt{n}}^{(i)}$, such that $N(V_i)$ has size n iff. $B_j^{(i)} \subseteq A_j^{(i)}$ for any j, while making sure that any other cut has size at least $n+1$. The way of connecting V_i to U is as follows: split U into $\sqrt{n}$ pieces $U_1, ...U_{\sqrt{n}}$, and connect the j-th vertex in V_i (denoted by $v_j^{(i)}$) to U_j according to $B_j^{(i)}$; connect the j-th vertex in V_i to U_k for any $k \ne j$ according to $A_k^{(i)}$.

A simple nearly tight upper bound. Now we describe a simple way to solve Min-Neighbor when V' is a common-neighborhood cluster in $\widetilde{O}(n^{1.5})$ communication. We break it into two cases based on the size of $L^\star$.

When $|L^\star| > n^{0.5}$, we can sample $\widetilde{O}(\sqrt{n})$ nodes and one of them will hit $L^\star$. Recall that if we know a node in $L^\star$, we can find $L^\star$ in one s-t vertex connectivity call. Thus, in this case, the communication complexity os $\widetilde{O}(n^{1.5})$.

If $|L^\star| < n^{0.5}$, for a common-neighborhood cluster V', we can recover all edges adjacent to V' by using $O(n^{1.5})$ communications in the following way: choose an arbitrary vertex $u \in V'$, Alice and Bob learn the set $N_G(u)$ by $O(n)$ communication, then for every

$v \in V' - \{u\}$, Alice and Bob can learn $N_G(v)$ by $O(n^{0.5})$ communication since $|N(u) \triangle N(v)| \leq n^{0.5}$. Knowing all edges adjacent to V' suffices to solve MinNNCC.

3 Preliminaries

3.1 Terminologies

Basic graph terminologies. An (undirected) graph is denoted by $G = (V, E)$ where V is the vertex (or node) set and $E \subseteq \{\{x, y\} \mid x, y \in V, x \neq y\}$ is the edge set[11]. Following the convention, we abuse the notations a bit and will also use (x, y) to denote an edge, which is a set $\{x, y\}$. x, y are the *endpoints* of the edge (x, y). $N_G(u)$ is the set of neighbors of u defined by $\{v \in V \mid \{u, v\} \in E\}$. We omit G in the subscript if G is clear from the context. We also define $N[u] = N(u) \cup \{u\}$. The *degree* of a vertex u is defined as $|N(u)|$. The *minimum degree* of G is defined as $\min_{v \in V} |N(v)|$. Define $E(A, B) = \{\{x, y\} \in E \mid x \in A, y \in B\}$.

For a vertex set V', $G[V']$ is called the induced subgraph of G on V' defined by $G[V'] = (V', \{(u, v) \in E \mid u, v \in V'\})$.

Throughout the paper, we use n to denote $|V|$ and m to denote $|E|$. We write $\widetilde{O}(f) = O(f \cdot \log^c n)$ for some constant c, and $\widehat{O}(f) = f \cdot n^{o(1)}$. It is important that n represents $|V|$ here, which is roughly the input size.

Set terminologies. We define $[z] = \{1, 2, ..., z\}$. For two sets A, B, we use $A - B$ or $A \backslash B$ to denote $\{a \in A \mid a \notin B\}$. We define $A \triangle B$ as the symmetric difference $(A - B) \cup (B - A)$. We say $(A_1, ..., A_z)$ is a partition of A if $\cup_{i \in [z]} A_i = A$ and $A_i \cap A_j = \emptyset$ for any $i \neq j$.

Vertex cut. A *vertex cut* (or simply *cut* in this paper) is a partition of V denoted by (L, S, R) where (1) $L \neq \emptyset, R \neq \emptyset, E(L, R) = \emptyset$ or (2) $|S| \geq n - 1$. The size of a vertex cut (L, S, R) is defined as $|S|$. The *minimum* vertex cut refers to the vertex cut with the smallest size. For a vertex $s \in V$, The *minimum t-sink vertex cut* is defined as a vertex cut (L, S, R) with $t \in R$ which minimizes $|S|$.

We say a vertex cut (L, S, R) as a (u, v)-vertex cut if $u \in L, v \in R$. For two vertex sets $A, B \subseteq V$, we say (L, S, R) is an (A, B)-vertex cut if $A \subseteq L, B \subseteq R$. In this case, we say S is a (u, v)-separator or (A, B)-separator. S is called a *separator* (or a *vertex cut*) if S is a (u, v)-separator for some $u, v \in V$. We use $\kappa_G(u, v)$ to denote the size of the smallest u, v-separator in G, and $\kappa(G)$ to denote the smallest separator in G. When there are no (u, v)-separator or no separator in G, we define $\kappa_G(u, v)$ or $\kappa(G)$ as $n - 1$. In this case, we say any $n - 1$ nodes is a separator.

For a vertex set $X \subseteq V$, we say X has *neighborhood difference* d if for any $u, v \in X$ we have $|N_G(u) \triangle N_G(v)| \leq d$. The following lemma is crucial to our algorithm.

Lemma 3.1. *Suppose (L, S, R) is a minimum vertex cut, then the neighborhood difference of L is at most $2|L|$.*

3.2 Computational Models

PRAM *model.* In PRAM model, we have a set of processors and a shared memory space. Each processor can independently read and write on the shared memory space or do other unit operations. The input is given initially on the shared memory space, and the processors are required to jointly compute a specific problem given the input. The complexity is measured by *work* and *depth*, where work is measured by the **total** amount of unit operations performed by all the processes, and depth is measured by the time consumed before the output is generated.

CONGEST *model.* In the distributed CONGEST model, we have an initial graph $G = (V, E)$ of n nodes with diameter D where each node has a unique ID and unlimited computational power. Initially, each node only knows its neighbors. The algorithm runs in synchronous rounds. For each round, each node can exchange $O(\log n)$ bits of information to its neighbors. The goal is to design an algorithm that minimizes the number of rounds needed to determine the output, for example, the vertex connectivity of G.

Two-party communication model. In the two-party communication model (or simply communication model), the edge set E is arbitrarily partitioned into two sets $E = E_1 \cup E_2$ where $E_1 \cap E_2 = \emptyset$. Two players Alice and Bob both know the vertex set V, and Alice knows the set E_1, and Bob knows the set E_2. Their goal is to exchange as few bits as possible to find some intrinsic property of the graph $G = (V, E)$.

Graph streaming model. In the graph streaming model (or simply streaming model), a graph $G = (V, E)$ is given to the algorithm as an arbitrarily ordered stream of edges. The algorithm has limited space and can read this stream in sequential passes. The goal is to minimize both the space usage and the number of passes. *Semi-streaming* refers to the case when the space is restricted to $\widetilde{O}(n)$.

Remark 3.2 (public and private randomness). In the two-party communication model and CONGEST model, *public randomness* refers to the case when every party (or node) can access the same random bits, *private randomness*, in contrast, does not allow parties (or nodes) to share the randomness. According to Newman's Theorem [51], to simulate public randomness using private randomness, parties only need to share an additional $O(\log n)$ bits of message (which can be broadcasted in CONGEST model).

3.3 Problem Definitions

We define the problems considered in this paper as follows. We say a (randomized) algorithm solves a problem if it outputs the correct answer with high probability (or simply w.h.p.), i.e., with probability at least $1 - \frac{1}{n^c}$ for an arbitrarily large constant c. All algorithms in this paper are randomized except with explicit clarification.

*Vertex connectivity (*VC*).* Given an undirected graph $G = (V, E)$, output a minimum vertex cut.

*Single sink unidirected vertex connectivity (*SSVC*).* Given a undirected graph $G = (V, E)$ and a sink vertex $t \in V$, output a minimum t-sink vertex cut.

s-t vertex connectivity (s-t VC*)..* Given an undirected graph $G = (V, E)$ and $s, t \in V$, output a minimum (s, t)-vertex cut.

S,T-vertex connectivity (S-T VC*).* Given an undirected graph $G = (V, E)$, and two vertex sets $A, B \subseteq V$, compute the minimum (A, B)-vertex cut.

[11] All graphs in this paper are simple, i.e., without multi-edges and self-loops, as written in the definition.

Subgraph problems in CONGEST. For subgraph S-T vertex connectivity (or other subgraph problems) in CONGEST, we are given a network along with a subgraph H where every node knows its adjacent edges in H, and we want to solve the problem on H while other edges of G are only used for communication purposes.

*Bipartite maximum matching (*BMM*)..* Given a bipartite graph (A, B, E) (which is defined as $E \subseteq A \times B$ where $A \cup B$ is the vertex set), output the maximum matching (a matching is defined as a set of edges with mutually disjoint endpoints).

*Bipartite minimum vertex cover (*BMC*).* Given a bipartite graph (A, B, E), output the minimum vertex cover (a vertex cover is defined as a set of vertices such that every edge has at least one endpoint in this set).

3.4 Sketching

We will use the standard linear sketching algorithms from [21].

Theorem 3.3 (Section 3 in [21]). *For any numbers n and s, there is an algorithm that preprocesses in $\widetilde{O}(s)$ work and $\widetilde{O}(1)$ depth and then, given any vector $v \in \{-1, 0, 1\}^n$, return a sketch $sk_s(v) \in \mathbb{Z}^{\widetilde{O}(s)}$ in $\widetilde{O}\left(\|v\|_2^2\right)$[12] work and $\widetilde{O}(1)$ depth and guarantees the following w.h.p. (as long as the number of recovery operations is poly(n)).*

- *If $\|v\|_2^2 \le s$, then we can recover v from $sk_s(v)$ in $\widetilde{O}(s)$ time and $\widetilde{O}(1)$ depth. (More specifically, we obtain all non-zero entries of v together with their indices).*
- *Otherwise, if $\|v\|_2^2 > s$, then the algorithm returns $\perp$.*

Moreover, the sketch is linear*, i.e. $sk_s(u+v) = sk_s(v) + sk_s(u)$ for any $u, v \in \mathbb{Z}^n$.*

The following corollary is from Theorem 3.3 combined with sampling.

Corollary 3.4. *For any number n, there is an algorithm that preprocesses in $\widetilde{O}(n)$ time and then, given any vector $v \in \{-1, 0, 1\}^n$, return a sketch $sk_\approx(v) \in \mathbb{R}^{\widetilde{O}(1)}$ in $\widetilde{O}\left(\|v\|_2^2\right)$ time such that whp. (as long as the number of recovery operations is poly(n)) we can recover an approximate size of $\|v\|_2^2$ (a value between $0.9\|v\|_2^2$ and $1.1\|v\|_2^2$) in $\widetilde{O}(1)$ work. Moreover, the sketch is* linear*, i.e. $sk_\approx(u+v) = sk_\approx(v) + sk_\approx(u)$ for any $u, v \in \mathbb{R}^n$.*

4 Parallel and Distributed Common-Neighborhood Clustering

We state the common-neighborhood clustering result.

Lemma 4.1 (common neighborhood clustering)**.** *There exists a randomized algorithm that given an undirected graph G and an integer ℓ, return $\mathcal{C}$ satisfying the following properties*

(1) (sparse) for every $v \in V$, v is included in at most $\widetilde{O}(1)$ clusters in $\mathcal{C}$,
(2) (common neighborhood) any $C \in \mathcal{C}$ has neighborhood difference $\ell \cdot 2^{\log^{0.7} n}$ with high probability,
(3) (cover) for every $L \subseteq V$ such that $G[L]$ is connected and L has neighborhood difference ℓ, we have $\Pr[\exists C \in \mathcal{C}, L \subseteq C] \ge \frac{1}{n^{o(1)}}$.

[12]Notice that $\|v\|_2^2$ is basically the number of non-zero entries of v.

The algorithm can be implemented in $\widetilde{O}(m)$ work and $\widehat{O}(1)$ depth in PRAM *model, or $\widehat{O}\left(\sqrt{n} + D\right)$ rounds in* CONGEST *model.*

5 A Schematic Reduction for Vertex Connectivity

In this section, we show a general framework for solving SSVC, which can be implemented in different models, and prove its correctness. It essentially uses common-neighborhood clustering to reduce the problem to solving *isolating cuts* and solving min-neighbor in a *near-clique common-neighborhood* graph, which we define as follows. For convenience, we write the second problem MinNNCC.

Definition 5.1 (Isolating Cuts Problem)**.** *In the isolating cuts problem, we are given an undirected graph $G = (V, E)$, vertex sets $C, T \subseteq V$ where $T \subseteq C$ is an independent set inside C, and the goal is to solve the following minimization problem.*

$$\min_{L \subseteq C} \quad |N_G(L)| \quad s.t. \quad |L \cap T| = 1 \quad \text{and } N_G(L) \cap T = \emptyset.$$

The algorithm returns a minimizer L^* along with its neighbors' size $|N_G(L^*)|$.

Definition 5.2 (Minimum Neighbor in a Near-Clique Cluster (MinNNCC))**.** *In this problem, we are given an undirected graph $G = (V, E)$, a vertex set $C \subseteq V$ where $N_G[C] \ne V$, and an integer ℓ. The goal is to find a non-empty vertex set $L \subseteq C$ minimizing $|N_G(L)|$ and also return $|N_G(L)|$. The inputs have the following guarantees:*

(1) (correct estimate). There exists $L \subseteq C$ which is a minimizer of $\min_{L' \subseteq C : L' \ne \emptyset} |N_G(L')|$ such that

$$|L| \le \ell \le 2|L|.$$

(2) (near clique). For any $u \in C$,

$$|C - N_G(u)| \le 2^{\log^{0.8} n} \cdot \ell, \tag{1}$$

(3) (common neighborhood). the cluster C has neighborhood difference $2^{\log^{0.7} n} \cdot \ell$. That is, for all $u, v \in C$,

$$|N_G(u) \triangle N_G(v)| \le 2^{\log^{0.7} n} \cdot \ell \tag{2}$$

If the input guarantees are unsatisfied, the algorithm returns an arbitrary $L \subseteq C$ and $|N_G(L)|$.

5.1 The Reduction

Let us denote the algorithm solving isolating cuts by IsolatingCuts(G, C, T) and the algorithm solving MinNNCC by MinNNCC(G, C, ℓ).

The Schematic Algorithm. The framework for solving SSVC is presented in algorithm 1.

The following lemma shows the correctness of algorithm 1 based on the correctness of IsolatingCuts and MinNNCC.

Lemma 5.3 (Correctness of the framework)**.** *Given that IsolatingCuts and MinNNCC correctly output according to definition 5.1 and definition 5.2, algorithm 1 returns a valid vertex cut, which is a minimum t-sink vertex cut with probability at least $1/n^{o(1)}$.*

Algorithm 1: $S \leftarrow \text{SSVC}(G, t)$

Input: An undirected graph $G = (V, E)$, a vertex t.
Output: A minimum t-sink vertex cut (L, S, R).

foreach $i = 0, 1, ..., \log n$ **do**
  $\ell \leftarrow 2^i$;
  $\mathcal{C} \leftarrow$ ComNeiClustering$(G, 2\ell)$;
  Let $C \leftarrow C - N_G[t]$ for every $C \in \mathcal{C}$;
  foreach $C \in \mathcal{C}$ **do**
    Let T include each node in C independently at random with probability $\frac{1}{\ell \cdot 2^{\log^{0.8} n}}$;
    Let G' be a subgraph of G only containing edges adjacent to C;
    $L_{C,1}, s_{C,1} \leftarrow$ IsolatingCuts(G', C, T);
    $L_{C,2}, s_{C,2} \leftarrow$ MinNNCC(G', C, ℓ);
  Return $N_G(L_{C,i})$ where $s_{C,i}$ is minimized among all $C \in \mathcal{C}$ and $i \in \{1, 2\}$;

5.2 Implementation in CONGEST and PRAM Models

As applications of the reduction, we present algorithms that solve IsolatingCuts and MinNNCC in PRAM and CONGEST models, which are summarized as in the following two lemmas.

Lemma 5.4. *Given a IsolatingCuts instance (G, C, T) where the input graph has m edges and n vertices, there is an algorithm solving it correctly with high probability in*

(1) PRAM *model, the algorithm can be implemented to run in work $\widetilde{O}(W(m, |C|))$ and depth $\widetilde{O}(D(m, |C|))$ where $W(m, n)$ and $D(m, n)$ are the work and depth of s-t vertex connectivity.*

(2) CONGEST *model, the algorithm can be implemented to run in $\tilde{O}(R(m, n, D))$ rounds where $R(m, n, D)$ is the round complexity of subgraph S-T vertex connectivity and D is the diameter of G; furthermore, the algorithm only communicates via the set of edges that is incident to C.*

Lemma 5.5. *Given a* MinNNCC *instance (G, C, ℓ) (Definition 5.2), it can be solved with high probability in*

(1) PRAM *model in $\widetilde{O}(W(m, n))$ work and $\widetilde{O}(D(m, n))$ depth where $W(m, n)$ and $D(m, n)$ are the work and depth of s-t vertex connectivity, satisfying that $W(m, n)$ is superadditive on m (meaning $W(m_1 + m_2, n) \geq W(m_1, n) + W(m_2, n)$ for every m_1, m_2, n),*

(2) CONGEST *model in $\tilde{O}(R(m, n, O(1))$ rounds where $R(m, n, D)$ is the round complexity of subgraph S-T vertex connectivity; furthermore, the algorithm only communicates via the set of edges that is incident to C.*

To solve IsolatingCuts and MinNNCC problems, in addition to sparsification techniques from [39], we introduce novel vertex sparsification lemmas that leverage the structure of the common-neighborhood property.

Lemma 5.6. *If, in* PRAM *model, the s-t vertex connectivity problem can be solved in $W(m, n)$ work and $D(m, n)$ depth where $W(m, n)$ is superadditive on m, then* SSVC *can be solved in $W(m, n) \cdot n^{o(1)}$ work and $D(m, n) \cdot n^{o(1)}$ depth.*

Proof. The correctness of algorithm 1 is proved by lemma 5.3. Now we calculate the total work and depth for one run of algorithm 1. There are $O(\log n)$ iterations of the outer loop of i. Common-neighborhood clustering uses $\widehat{O}(m)$ work and $\widehat{O}(1)$ depth. For every $C \in \mathcal{C}$, we solve the problems IsolatingCuts and MinNNCC simultaneously. Write E_C as the edge set of edges in G adjacent to C. Each one of them on C cost work $\widetilde{O}(W(|E_C|, n))$ and depth $\widetilde{O}(D(|E_C|, n))$ according to lemma 5.4 and lemma 5.5. Notice that each edge can be contained in at most $\widetilde{O}(1)$ different C according to lemma 4.1. Thus, the total work is $\sum_{C \in \mathcal{C}} \widetilde{O}(W(|E_C|), n)) \leq \widetilde{O}\left(W(\sum_{C \in \mathcal{C}} |E_C|, n)\right) = \widetilde{O}(W(m, n))$ where the inequality is due to W is superadditive on m, and the depth is $\widetilde{O}(D(m, n))$ since they are run simultaneously. □

Lemma 5.7. *If, in* CONGEST *model, the subgraph S-T vertex connectivity problem can be solved in $R(m, n, D)$ rounds, then* SSVC *can be solved in $R(m, n, D) \cdot n^{o(1)}$ depth.*

Proof. The correctness of algorithm 1 is proved by lemma 5.3. Notice that to boost the correct probability to w.h.p., we repeat the algorithm $\widehat{O}(1)$ times and choose the smallest vertex cut as the output (the size of each vertex cut is also output according to definitions 5.1 and 5.2), which only increases the work and depth by at most a $\widehat{O}(1)$ factor. Now we calculate the total rounds for one run of algorithm 1.

There are $O(\log n)$ iterations of the outer loop of i. For each iteration, common-neighborhood clustering uses $\widehat{O}\left(\sqrt{n} + D\right)$ rounds. For every $C \in \mathcal{C}$, we solve the problems MinNNCC simultaneously. Notice that each edge can be adjacent to at most $\widetilde{O}(1)$ different C according to lemma 4.1. Thus, each edge is involved in at most $\widetilde{O}(1)$ different algorithms of IsolatingCuts and MinNNCC, which results in total congestion of $\widetilde{O}(R(m, n, O(1)))$ according to lemma 5.4 and lemma 5.5. So the total number of rounds caused by MinNNCC is at most $\widetilde{O}(R(m, n, O(1)))$. We defer the implementation of IsolatingCuts for every cluster $C \in \mathcal{C}$ by one call to lemma 5.4 in the full version. □

Acknowledgements

The project is partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 759557–ALGOCom), and the ERC Starting Grant (CODY 101039914), Dr. Max Rössler, the Walter Haefner Foundation, and the ETH Zürich Foundation. This work was done partly while SY was visiting the Simons Institute for the Theory of Computing, UC Berkeley. We thank Danupon Nanongkai for the helpful discussions and the anonymous reviewers for their helpful suggestions on the writing.

References

[1] Ittai Abraham, Yair Bartal, and Ofer Neiman. 2008. Nearly Tight Low Stretch Spanning Trees. In *FOCS*. IEEE Computer Society, 781–790.
[2] Kook Jin Ahn and Sudipto Guha. 2011. Linear Programming in the Semi-streaming Model with Application to the Maximum Matching Problem. In *ICALP (2) (Lecture Notes in Computer Science, Vol. 6756)*. Springer, 526–538.
[3] Vikrant Ashvinkumar, Aaron Bernstein, Nairen Cao, Christoph Grunau, Bernhard Haeupler, Yonggang Jiang, Danupon Nanongkai, and Hsin-Hao Su. 2024. Parallel, Distributed, and Quantum Exact Single-Source Shortest Paths with Negative Edge Weights. In *ESA (LIPIcs, Vol. 308)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:15.

[4] Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. 2022. Semi-Streaming Bipartite Matching in Fewer Passes and Optimal Space. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 627–669.

[5] Sepehr Assadi and Ran Raz. 2020. Near-Quadratic Lower Bounds for Two-Pass Graph Streaming Algorithms. In *FOCS*. IEEE, 342–353.

[6] Sepehr Assadi and Vihan Shah. 2023. Tight Bounds for Vertex Connectivity in Dynamic Streams. In *SOSA*. SIAM, 213–227.

[7] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. 1998. Near-Linear Time Construction of Sparse Neighborhood Covers. *SIAM J. Comput.* 28, 1 (1998), 263–277.

[8] Michael Becker, W. Degenhardt, Jürgen Doenhardt, Stefan Hertel, Gerd Kaninke, W. Kerber, Kurt Mehlhorn, Stefan Näher, Hans Rohnert, and Thomas Winter. 1982. A Probabilistic Algorithm for Vertex Connectivity of Graphs. *Inf. Process. Lett.* 15, 3 (1982), 135–136.

[9] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. 2022. Negative-Weight Single-Source Shortest Paths in Near-linear Time. In *FOCS*. IEEE, 600–611.

[10] Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. 2022. Nearly Optimal Communication and Query Complexity of Bipartite Matching. *arXiv preprint arXiv:2208.02526* (2022).

[11] Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. 2022. Nearly Optimal Communication and Query Complexity of Bipartite Matching. In *FOCS*. IEEE, 1174–1185.

[12] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. 2014. Distributed connectivity decomposition. In *PODC*. ACM, 156–165.

[13] Yi-Jun Chang and Thatchaphol Saranurak. 2019. Improved Distributed Expander Decomposition and Nearly Optimal Triangle Enumeration. In *PODC*. ACM, 66–73.

[14] Yi-Jun Chang and Thatchaphol Saranurak. 2020. Deterministic Distributed Expander Decomposition and Routing with Applications in Distributed Derandomization. In *FOCS*. IEEE, 377–388.

[15] Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh R. Saxena, Zhao Song, and Huacheng Yu. 2021. Almost optimal super-constant-pass streaming lower bounds for reachability. In *STOC*. ACM, 570–583.

[16] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2022. Maximum Flow and Minimum-Cost Flow in Almost-Linear Time. In *FOCS*. IEEE, 612–623.

[17] Joseph Cheriyan, Ming-Yang Kao, and Ramakrishna Thurimella. 1993. Scan-First Search and Sparse Certificates: An Improved Parallel Algorithms for k-Vertex Connectivity. *SIAM J. Comput.* 22, 1 (1993), 157–174.

[18] Joseph Cheriyan and John H. Reif. 1994. Directed *s-t* Numberings, Rubber Bands, and Testing Digraph *k*-Vertex Connectivity. *Combinatorica* 14, 4 (1994), 435–451. Announced at SODA'92.

[19] Joseph Cheriyan and Ramakrishna Thurimella. 1991. Algorithms for Parallel k-Vertex Connectivity and Sparse Certificates (Extended Abstract). In *STOC*. ACM, 391–401.

[20] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. 2014. Solving SDD linear systems in nearly $m\log^{1/2} n$ time. In *STOC*. ACM, 343–352.

[21] Graham Cormode and Donatella Firmani. 2014. A unifying framework for ℓ_0-sampling algorithms. *Distributed Parallel Databases* 32, 3 (2014), 315–335.

[22] Michal Dory, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. 2021. Distributed weighted min-cut in nearly-optimal time. In *STOC*. ACM, 1144–1153.

[23] Abdol-Hossein Esfahanian and S. Louis Hakimi. 1984. On computing the connectivities of graphs and digraphs. *Networks* 14, 2 (1984), 355–366.

[24] Shimon Even. 1975. An Algorithm for Determining Whether the Connectivity of a Graph is at Least k. *SIAM J. Comput.* 4, 3 (1975), 393–396.

[25] Shimon Even and Robert Endre Tarjan. 1975. Network Flow and Testing Graph Connectivity. *SIAM J. Comput.* 4, 4 (1975), 507–518.

[26] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. 2004. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.* 69, 3 (2004), 485–497.

[27] Harold N. Gabow. 2006. Using expander graphs to find vertex connectivity. *J. ACM* 53, 5 (2006), 800–844. Announced at FOCS'00.

[28] Zvi Galil. 1980. Finding the Vertex Connectivity of Graphs. *SIAM J. Comput.* 9, 1 (1980), 197–199.

[29] Mohsen Ghaffari and Goran Zuzic. 2022. Universally-Optimal Distributed Exact Min-Cut. In *PODC*. ACM, 281–291.

[30] Lukas Gianinazzi and Torsten Hoefler. 2020. Parallel Planar Subgraph Isomorphism and Vertex Connectivity. In *SPAA*. ACM, 269–280.

[31] Bernhard Haeupler, D. Ellis Hershkowitz, Jason Li, Antti Roeyskoe, and Thatchaphol Saranurak. 2024. Low-Step Multi-commodity Flow Emulators. In *STOC*. ACM, 71–82.

[32] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. 2022. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *STOC*. ACM, 1325–1338.

[33] Monika Rauch Henzinger. 1997. A Static 2-Approximation Algorithm for Vertex Connectivity and Incremental Approximation Algorithms for Edge and Vertex Connectivity. *J. Algorithms* 24, 1 (1997), 194–220.

[34] Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. 2000. Computing Vertex Connectivity: New Bounds from Old Techniques. *J. Algorithms* 34, 2 (2000), 222–250. Announced at FOCS'96.

[35] Yonggang Jiang and Sagnik Mukhopadhyay. 2023. Finding a Small Vertex Cut on Distributed Networks. In *STOC*. ACM, 1791–1801.

[36] David R. Karger. 2000. Minimum cuts in near-linear time. *J. ACM* 47, 1 (2000), 46–76. announced at STOC'96.

[37] Richard M. Karp, Eli Upfal, and Avi Wigderson. 1986. Constructing a perfect matching is in random NC. *Comb.* 6, 1 (1986), 35–48.

[38] D Kleitman. 1969. Methods for investigating connectivity of large graphs. *IEEE Transactions on Circuit Theory* 16, 2 (1969), 232–233.

[39] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. 2021. Vertex connectivity in poly-logarithmic max-flows. In *STOC*. ACM, 317–329.

[40] Jason Li and Debmalya Panigrahi. 2020. Deterministic Min-cut in Poly-logarithmic Max-flows. In *FOCS*. IEEE, 85–92.

[41] Nathan Linial, László Lovász, and Avi Wigderson. 1988. Rubber bands, convex embeddings and graph connectivity. *Combinatorica* 8, 1 (1988), 91–102. Announced at FOCS'86.

[42] Andrés López-Martínez, Sagnik Mukhopadhyay, and Danupon Nanongkai. 2021. Work-Optimal Parallel Minimum Cuts for Non-Sparse Graphs. In *SPAA*. ACM, 351–361. doi:10.1145/3409964.3461806

[43] László Lovász. 1979. On determinants, matchings, and random algorithms. In *FCT*. Akademie-Verlag, Berlin, 565–574.

[44] David W. Matula. 1987. Determining Edge Connectivity in O(nm). In *FOCS*. IEEE Computer Society, 249–251.

[45] Gary L. Miller, Richard Peng, and Shen Chen Xu. 2013. Parallel graph decompositions using random shifts. In *SPAA*. ACM, 196–203.

[46] Sagnik Mukhopadhyay and Danupon Nanongkai. 2020. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *STOC*. ACM, 496–509.

[47] Sagnik Mukhopadhyay and Danupon Nanongkai. 2021. A Note on Isolating Cut Lemma for Submodular Function Minimization. *CoRR* abs/2103.15724 (2021).

[48] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. 1987. Matching is as easy as matrix inversion. *Comb.* 7, 1 (1987), 105–113.

[49] Hiroshi Nagamochi and Toshihide Ibaraki. 1992. A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph. *Algorithmica* 7, 5&6 (1992), 583–596.

[50] Danupon Nanongkai. 2024. Cross-Paradigm Graph Algorithms (Invited Talk). In *ICALP (LIPIcs, Vol. 297)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 3:1–3:1.

[51] Ilan Newman. 1991. Private vs. Common Random Bits in Communication Complexity. *Inf. Process. Lett.* 39, 2 (1991), 67–71.

[52] Merav Parter. 2020. Distributed Planar Reachability in Nearly Optimal Time. In *34th International Symposium on Distributed Computing (DISC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[53] Merav Parter and Asaf Petruschka. 2022. Near-Optimal Distributed Computation of Small Vertex Cuts. In *36th International Symposium on Distributed Computing (DISC 2022)*.

[54] VD Podderyugin. 1973. An algorithm for finding the edge connectivity of graphs. *Vopr. Kibern* 2 (1973), 136.

[55] David Pritchard and Ramakrishna Thurimella. 2011. Fast computation of small cuts via cycle space sampling. *ACM Trans. Algorithms* 7, 4 (2011), 46:1–46:30.

[56] Harald Räcke, Chintan Shah, and Hanjo Täubig. 2014. Computing Cut-Based Hierarchical Decompositions in Almost Linear Time. In *SODA*. SIAM, 227–238.

[57] Václav Rozhon, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. 2022. Undirected $(1+\epsilon)$-shortest paths via minor-aggregates: near-optimal deterministic parallel and distributed algorithms. In *STOC*. ACM, 478–487.

[58] Thatchaphol Saranurak and Sorrachai Yingchareonthawornchai. 2022. Deterministic Small Vertex Connectivity in Almost Linear Time. In *FOCS*. IEEE, 789–800.

[59] Jonah Sherman. 2013. Nearly Maximum Flows in Nearly Linear Time. In *FOCS*. IEEE Computer Society, 263–269.

[60] Daniel A. Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*. ACM, 81–90.

[61] Ramakrishna Thurimella. 1997. Sub-Linear Distributed Algorithms for Sparse Certificates and Biconnected Components. *J. Algorithms* 23, 1 (1997), 160–179. Announced at PODC'95.

Received 2024-11-04; accepted 2025-02-01