Distributed Computing on Core-Periphery Networks: Axiom-based Design

Chen Avin\textsuperscript{1,\dagger,\ddagger}, Michael Borokhovich\textsuperscript{1,\ddagger}, Zvi Lotker\textsuperscript{1,\ddagger}, and David Peleg\textsuperscript{2,\ddagger}

\textsuperscript{1} Ben-Gurion University of the Negev, Israel. \{avin,borokhom,zvilo\}@cse.bgu.ac.il

\textsuperscript{2} The Weizmann Institute, Israel. david.peleg@weizmann.ac.il

Abstract. Inspired by social networks and complex systems, we propose a core-periphery network architecture that supports fast computation for many distributed algorithms and is robust and efficient in number of links. Rather than providing a concrete network model, we take an axiom-based design approach. We provide three intuitive (and independent) algorithmic axioms and prove that any network that satisfies all axioms enjoys an efficient algorithm for a range of tasks (e.g., MST, sparse matrix multiplication, etc.). We also show the minimality of our axiom set: for networks that satisfy any subset of the axioms, the same efficiency cannot be guaranteed for any deterministic algorithm.

1 Introduction

A fundamental goal in distributed computing concerns finding a network architecture that allows fast running times for various distributed algorithms, but at the same time is cost-efficient in terms of minimizing the number of communication links between the machines and the amount of memory used by each processor.

For illustration, let’s consider three basic networks topologies: a star, a clique and a constant degree expander. The star graph has only a linear number of links and can compute every computable function after only one round of communication. But clearly, such an architecture has two major disadvantages: the memory requirements of the central node do not scale, and the network is not robust. The complete graph, on the other hand, is very robust and can support extremely fast performance for tasks such as information dissemination, distributed sorting and minimum spanning tree, to name a few [123]. Also, in a complete graph the amount of memory used by a single processor is minimal. But obviously, the main drawback of that architecture is the high number of links it uses. Constant degree expanders are a family of graphs that support efficient computation for many tasks. They also have linear number of links and can effectively balance the workload between many machines. But the diameter of these graphs

\dagger Part of this work was done while the author was a visitor at ICERM, Brown university.

\ddagger Supported in part by the Israel Science Foundation (grant 1549/13).
is lower bounded by \( \Omega(\log n) \) which implies similar lower bound for most of the interesting tasks one can consider.

A natural question is therefore whether there are other candidate topologies with guaranteed good performance. We are interested in the best compromise solution: a network on which distributed algorithms have small running times, memory requirements at each node are limited, the architecture is robust to link and node failures, and the total number of links is minimized (preferably linear).

To try to answer this question we adopt in this paper an axiomatic approach to the design of efficient networks. In contrast to the direct approach to network design, which is based on providing a concrete type of networks (by deterministic or random construction) and showing its efficiency, the axiomatic approach attempts to abstract away the algorithmic requirements that are imposed on the concrete model. This allows one to isolate and identify the basic requirements that a network needs for a certain type of tasks. Moreover, while usually the performance of distributed algorithms is dictated by specific structural properties of a network (e.g., diameter, conductance, degree, etc.), the axioms proposed in this work are expressed in terms of desired algorithmic properties that the network should have.

The axioms proposed in the current work are motivated and inspired by the core-periphery structure exhibited by many social networks and complex systems. A core-periphery network is a network structured of two distinct groups of nodes, namely, a large, sparse, and weakly connected group identified as the periphery, which is loosely organized around a small, cohesive and densely connected group identified as the core. Such a dichotomic structure appears in many areas of our life, and has been observed in many social organizations including modern social networks [4]. It can also be found in urban and even global systems (e.g., in global economy, the wealthiest countries constitute the core which is highly connected by trade and transportation routes) [5,6,7]. There are also peer-to-peer networks that use a similar hierarchical structure, e.g., FastTrack [8] and Skype [9], in which the supernodes can be viewed as the core and the regular users as the periphery.

The main technical contribution of this paper is proposing a minimal set of simple core-periphery-oriented axioms and demonstrating that networks satisfying these axioms achieve efficient running time for various distributed computing tasks while being able to maintain linear number of edges and limited memory use. We identify three basic, abstract and conceptually simple (parameterized) properties, that turn out to be highly relevant to the effective interplay between core and periphery. For each of these three properties, we propose a corresponding axiom, which in our belief captures some intuitive aspect of the desired behavior expected of a network based on a core-periphery structure. Let us briefly describe our three properties, along with their “real life” interpretation, technical formulation, and associated axioms.

---

3 One may ask whether the properties we define qualify as “axioms”. Our answer is that the axiomatic lens helps us focus attention on the fundamental issues of minimality, independence and necessity of our properties.
| Task                              | Running time on CP networks | Lower bounds for All Axioms | Lower bounds for Any 2 Axioms |
|----------------------------------|-----------------------------|-----------------------------|------------------------------|
| MST *                            | $O(\log^2 n)$              | $\Omega(1)$                 | $\Omega(\sqrt[4]{n})$       |
| Matrix transposition             | $O(k)$                     | $\Omega(k)$                 | $\Omega(n)$                 |
| Vector by matrix multiplication  | $O(k)$                     | $\Omega(k/\log n)$         | $\Omega(n/\log n)$          |
| Matrix multiplication            | $O(k^2)$                   | $\Omega(k^2)$               | $\Omega(n/\log n)$          |
| Find my rank                     | $O(1)$                     | $\Omega(1)$                 | $\Omega(n)$                 |
| Find median                      | $O(1)$                     | $\Omega(1)$                 | $\Omega(\log n)$            |
| Find mode                        | $O(1)$                     | $\Omega(1)$                 | $\Omega(n/\log n)$          |
| Find number of distinct values   | $O(1)$                     | $\Omega(1)$                 | $\Omega(n/\log n)$          |
| Top $r$ ranked by areas          | $O(r)$                     | $\Omega(r)$                 | $\Omega(r\sqrt{n})$         |

$k$ - maximum number of nonzero entries in a row or column. * - randomized algorithm

Table 1: Summary of algorithms for core-periphery networks.

The three axioms are: (i) clique-like structure of the core, (ii) fast convergecast from periphery to the core and (iii) balanced boundary between the core and periphery. The first property deals with the flow of information within the core. It is guided by the key observation that to be influential, the core must be able to accomplish fast information dissemination internally among its members. The corresponding Axiom $A_E$ postulates that the core must be a $\Theta(1)$-clique emulator (to be defined formally later). Note that this requirement is stronger than just requiring the core to possess a dense interconnection subgraph, since the latter permits the existence of “bottlenecks”, whereas the requirement of the axiom disallows such bottlenecks.

The second property focuses on the flow of information from the periphery to the core and measures its efficiency. The core-periphery structure of the network is said to be a $\gamma$-convergecaster if this data collection operation can be performed in time $\gamma$. The corresponding Axiom $A_C$ postulates that information can flow from the periphery nodes to the core efficiently (i.e., in constant time). Note that one implication of this requirement is that the presence of periphery nodes that are far away from the core, or bottleneck edges that bridge between many periphery nodes and the core, is forbidden.

The third and last property concerns the “boundary” between the core and the periphery and claim that core nodes are “effective ambassadors”. Ambassadors serve as bidirectional channels through which information flows into the core and influence flows from the core to the periphery. However, to be effective as an ambassador, the core node must maintain a balance between its interactions with the “external” periphery and its interactions with the other core members, serving as its natural “support”; a core node which is significantly more connected to the periphery than to the core becomes ineffective as a channel of influence. In distributed computing terms, a core node that has many connections to the periphery has to be able to distribute all the information it collected from them to other core nodes. The corresponding Axiom $A_B$ states that the core must have a $\Theta(1)$-balanced boundary (to be defined formally later).

To support and justify our selection of axioms, we examine their usefulness for effective distributed computations on core-periphery networks. We consider a
collection of different types of tasks, and show that they can be efficiently solved on core-periphery networks, by providing a distributed algorithm for each task and bounding its running time. Moreover, for each task we argue the necessity of all three axioms, by showing that if at least one of the axioms is not satisfied by the network under consideration, then the same efficiency cannot be guaranteed by any algorithm for the given task.

Table 1 provides an overview of the main tasks we studied along with the upper and lower bounds on the running time when the network satisfies our axioms and a worst case lower bound on the time required when at least one of the axioms is not satisfied. For each task we provide an algorithm and prove formally its running time and the necessity of the axioms. As it turns out, some of the necessity proofs make use of an interesting connection to known communication complexity results.

The most technically challenging part of the paper is the distributed construction of a minimum-weight spanning tree (MST), a significant task in both the distributed systems world (cf. [10,11,12]) and the social networks world [13,14,15]. Thus, the main algorithmic result of the current paper is proving that MST can be computed efficiently (in $O(\log^2 n)$ rounds) on core-periphery networks. To position this result in context we recall that for the complete graph $G = K_n$, an MST can be constructed distributively in $O(\log \log n)$ time [1]. For the wider class of graphs of diameter at most 2, this task can still be performed in time $O(\log n)$. In contrast, taking the next step, and considering graphs of diameter 3, drastically changes the picture, as there are examples of such graphs for which any distributed MST construction requires $\Omega(\sqrt{n})$ time [10].

The rest of the paper is organized as follows. Section 2 formally describes core-periphery networks, the axioms and their basic structural implications. Section 3 provides an overview of the MST algorithm and Section 4 an overview of the rest of the task we study. Due to lack of space we defer many of the technical details and proofs to the Appendix.

2 Axiomatic design for core-periphery networks

Preliminaries Let $G(V,E)$ denote our (simple, undirected) network, where $V$ is the set of nodes, $|V| = n$, and $E$ is the set of edges, $|E| = m$. The network can be thought of as representing a distributed system. We assume the synchronous CONGEST model (cf. [12]), where communication proceeds in rounds and in each round each node can send a message of at most $O(\log n)$ bits to each of its neighbors. Initially each node has a unique ID of $O(\log n)$ bits.

For a node $v$, let $N(v)$ denote its set of neighbors and $d(v) = |N(v)|$ its degree. For a set $S \subseteq V$ and a node $v \in S$, let $N_{in}(v,S) = N(v) \cap S$ denote its set of neighbors within $S$ and denote the number of neighbors of $v$ in the set $S$ by $d_{in}(v,S) = |N_{in}(v,S)|$. Analogously, let $N_{out}(v,S) = N(v) \cap V \setminus S$ denote $v$’s set of neighbors outside the set $S$ and let $d_{out}(v) = |N_{out}(v,S)|$. For two subsets $S,T \subseteq V$, let $\partial(S,T)$ be the edge boundary (or cut) of $S$ and $T$, namely
the set of edges with exactly one endpoint in $S$ and one in $T$ and $|\partial(S,T)| = \sum_{v \in S} |N_{out}(v, S) \cap T|$. Let $\partial(S)$ denote the special case where $T = V \setminus S$.

**Core-periphery networks** Given a network $G(V, E)$, a $(\mathcal{C}, \mathcal{P})$-partition is a partition of the nodes of $V$ into two sets, the core $\mathcal{C}$ and the periphery $\mathcal{P}$. Denote the sizes of the core and the periphery by $n_c$ and $n_p$ respectively. To represent the partition along with the network itself, we denote the partitioned network by $G(V, E, \mathcal{C}, \mathcal{P})$.

Intuitively, the core $\mathcal{C}$ consists of a relatively small group of strong and highly connected machines designed to act as central servers, whereas the periphery $\mathcal{P}$ consists of the remaining nodes, typically acting as clients. The periphery machines are expected to be weaker and less well connected than the core machines, and they may perform much of their communication via the dense interconnection network of the core. In particular, a central component in many of our algorithms for various coordination and computational tasks is based on assigning each node $v$ a representative core node $r(v)$, essentially a neighbor acting as a “channel” between $v$ and the core. The representative chosen for each periphery node is fixed.

For a partitioned network to be effective, the $(\mathcal{C}, \mathcal{P})$ partition must possess certain desirable properties. In particular, a partitioned network $G(V, E, \mathcal{C}, \mathcal{P})$ is called a core-periphery network, or CP-network for short, if the $(\mathcal{C}, \mathcal{P})$-partition satisfies three properties, defined formally later on in the form of three axioms.

**Core-periphery properties and axioms** We first define certain key parameterized properties of node groups in networks that are of particular relevance to the relationships between core and periphery in our partitioned network architectures. We then state our axioms, which capture the expected behavior of those properties in core-periphery networks, and demonstrate their independence and necessity. Our three basic properties are:

**$\alpha$-Balanced Boundary.** A subset of nodes $S$ is said to have an $\alpha$-balanced boundary iff $\frac{d_{out}(v, S)}{d_{in}(v, S) + 1} = O(\alpha)$ for every node $v \in S$.

**$\beta$-Clique Emulation.** The task of clique emulation on an $n$-node graph $G$ involves delivering a distinct message $M_{v,w}$ from $v$ to $w$ for every pair of nodes $v, w$ in $V(G)$. An $n$-node graph $G$ is a $\beta$-clique-emulator if it is possible to perform clique emulation on $G$ within $\beta$ rounds (in the CONGEST model).

**$\gamma$-Convergecast.** For $S, T \subseteq V$, the task of $(S, T)$-convergecast on a graph $G$ involves delivering $|S|$ distinct messages $M_v$, originated at the nodes $v \in S$, to some nodes in $T$ (i.e., each message must reach at least one node in $T$). The sets $S, T \subset V$ form a $\gamma$-convergecaster if it is possible to perform $(S, T)$-convergecast on $G$ in $\gamma$ rounds (in the CONGEST model).

Consider a partitioned network $G(V, E, \mathcal{C}, \mathcal{P})$. We propose the following set of axioms concerning the core $\mathcal{C}$ and periphery $\mathcal{P}$.

**$A_B$. Core Boundary.** The core $\mathcal{C}$ has a $\Theta(1)$-balanced boundary.
**A_E. Clique Emulation.** The core $C$ is a $\Theta(1)$-clique emulator.

**A_C. Periphery-Core Convergecast.** The periphery $P$ and the core $C$ form a $\Theta(1)$-convergecaster.

Let us briefly explain the axioms. Axiom $A_B$ talks about the boundary between the core and periphery. Think of core nodes with a high out-degree (i.e., with many links to the periphery) as "ambassadors" of the core to the periphery. Axiom $A_B$ states that while not all nodes in the core must serve as ambassadors, if a node is indeed an ambassador, then it must also have many links within the core. Axiom $A_E$ talks about the flow of information within the core, and postulates that the core must be dense, and in a sense behave almost like a complete graph: "everyone must know everyone else". The clique-emulation requirement is actually stronger than just being a dense subgraph, since the latter permits the existence of bottlenecks nodes, which a clique-emulator must avoid. Axiom $A_C$ also concerns the boundary between the core and periphery, but in addition it refers also to the structure of the periphery. It postulates that information can flow efficiently from the periphery to the core. For example, it forbids the presence of periphery nodes that are far away from the core, or bottleneck edges that bridge between many periphery nodes and the core. Fig. 1 (I) provides an example for $CP$-network satisfying the three axioms.

We next show that the axioms are independent. Later, we prove the necessity of the axioms for the efficient performance of a variety of computational tasks.

**Theorem 1.** Axioms $A_B$, $A_E$, $A_C$ are independent, namely, assuming any two of them does not imply the third.

We prove this theorem by considering three examples of partitioned networks, described next, each of which satisfies two of the axioms but not the third (hence they are not $CP$-networks), implying independence.

*The lollipop partitioned network $L_n$: (Fig. 1 (II)(a)) The lollipop graph consists of a $\sqrt{n}$-node clique and an $n - \sqrt{n}$-node line attached to some node of the clique. Set the core $C$ to be the clique and the periphery $P$ to be the line. Observe that Axioms $A_E$ and $A_B$ hold but $A_C$ is not satisfied since the long line will require linear time for periphery to core convergecast.*

*The sun partitioned network $S_n$: (Fig. 1 (II)(b)) The sun graph consists of an $n/2$-node cycle with an additional leaf node attached to each cycle node. Set the core $C$ to be the cycle and the periphery $P$ to contain the $n/2$ leaves. Axioms $A_C$ and $A_B$ hold but Axiom $A_E$ does not, since the distance between diametrically opposing nodes in the cycle is $n/4$, preventing fast clique emulation.*

*The dumbbell partitioned network $D_n$: (Fig. 1 (II)(c)) The dumbbell graph is composed of two stars, each consisting of a center node connected to $n/2 - 1$ leaves, whose centers are connected by an edge. Set the core $C$ to be the two centers, and the periphery $P$ to contain all the leaves. Then Axioms $A_E$ and $A_C$ hold while Axiom $A_B$ does not.*
Fig. 1: (I) An example for a 36-node CP-network that satisfies all three axioms. The 6 core nodes (in gray) are connected in clique. In this example every core node is also an ambassador with equal number of edges to the core and outside from the core. The core and periphery form a convergecaster since the periphery can send all its information to the core in one round. (II) Networks used in proofs: (a) The lollipop partitioned network \( L_{25} \). (b) The sun partitioned network \( S_{16} \). (c) The dumbbell partitioned network \( D_{16} \).

**Structural implications of the axioms** The axioms imply a number of simple properties of the network structure.

**Theorem 2.** If \( G(V, E, C, P) \) is a core-periphery network (i.e., it satisfies Axioms \( A_B \), \( A_E \) and \( A_C \)), then the following properties hold:

1. The size of the core satisfies \( \Omega(\sqrt{n}) \leq n_C \leq O(\sqrt{m}) \).
2. Every node \( v \) in the core satisfies \( d_{\text{out}}(v, C) = O(n_C) \) and \( d_{\text{in}}(v, C) = \Omega(n_C) \).
3. The number of outgoing edges from the core is \( |\partial(C)| = \Theta(n_C^2) \).
4. The core is dense, i.e., the number of edges in it is \( \sum_{v \in C} d_{\text{in}}(v, C) = \Theta(n_C^2) \).

**Proof.** Axiom \( A_E \) necessitates that the inner degree of each node \( v \) is \( d_{\text{in}}(v, C) = \Omega(n_C) \) (or else it would not be possible to complete clique emulation in constant time), implying the second part of claim 2. It follows that the number of edges in the core is \( \sum_{v \in C} d_{\text{in}}(v, C) = \Theta(n_C^2) \), hence it is dense; claim 4 follows. Since also \( \sum_{v \in C} d_{\text{in}}(v, C) \leq 2m \), we must have the upper bound of claim 1, that is, \( n_C = O(\sqrt{m}) \). Axiom \( A_B \) yields that for every \( v \), \( d_{\text{out}}(v, C) = O(n_C) \), so the first part of claim 2 follows. Note that \( |\partial(C)| = \sum_{v \in C} d_{\text{out}}(v, C) = O(n_C^2) \), so the upper bound of claim 3 follows. To give a lower bound on \( n_C \), note that by Axiom \( A_C \) we have \( |\partial(C)| = \Omega(n - n_C) \) (otherwise the information from the \( n - n_C \) nodes of \( P \) could not flow in \( O(1) \) time to \( C \)), so \( n_C = \Omega(\sqrt{n}) \) and the lower bounds of claims 1 and 3 follow.

An interesting case for efficient networks is where the number of edges is linear in the number of nodes. In this case we have the following corollary.

**Corollary 1.** In a core-periphery network where \( m = O(n) \), the following properties hold:
1. The size of the core satisfies \( n_c = \Theta(\sqrt{n}) \)
2. The number of outgoing edges from the core is \( |\partial(C)| = \Theta(n) \).
3. The number of edges in the core is \( \sum_{v \in C} d_{in}(v,C) = \Theta(n) \).

Now we show a key property relating our axioms to the network diameter.

Claim 1. If the partitioned network \( G(V,E,C,\mathcal{P}) \) satisfies Axioms \( \mathcal{A}_E \) and \( \mathcal{A}_C \) then its diameter is \( \Theta(1) \).

The following claim shows that the above conditions are necessary.

Claim 2. For \( X \in \{E,C\} \), there exists a family of \( n \)-node partitioned networks \( G_X(V,E,C,\mathcal{P}) \) of diameter \( \Omega(n) \) that satisfy all axioms except \( \mathcal{A}_X \).

3 MST on a Core-Periphery Network

In this section we present a time-efficient randomized distributed algorithm for computing a minimum-weight spanning tree (MST) on a core-periphery network. In particular, we consider an \( n \)-node core periphery network \( G(V,E,C,\mathcal{P}) \), namely, a partitioned network satisfying all three axioms, and show that an MST can be distributedly computed on such a network in \( O(\log^2 n) \) rounds with high probability. Upon termination, each node knows which of its edges belong to the MST. Formally, we state the following theorem.

Theorem 3. On a CP-network \( G(V,E,C,\mathcal{P}) \), Algorithm CP-MST constructs an MST in \( O(\log^2 n) \) rounds with high probability.

We also show that Axioms \( \mathcal{A}_B \), \( \mathcal{A}_E \), and \( \mathcal{A}_C \) are indeed necessary for our distributed MST algorithm to be efficient.

Theorem 4. For each \( X \in \{B,E,C\} \) there exists a family \( \mathcal{F}_X = \{G_X(V,E,C,\mathcal{P})(n)\} \) of \( n \)-node partitioned networks that do not satisfy Axiom \( \mathcal{A}_X \) but satisfy the other two axioms, and the time complexity of any distributed MST algorithm on \( \mathcal{F}_X \) is \( \Omega(n^{\alpha_X}) \) for some constant \( \alpha_X > 0 \).

The formal proof of Theorem 4 can be found in Appendix B, but the idea of the proof is as following. For each case of Theorem 4 we show a graph in which, for a certain weight assignment, there exist two nodes \( s \) and \( r \) such that in order to decide which of the edges incident to \( r \) belong to the MST, it is required to know the weights of all the edges incident to \( s \). Thus, at least \( \deg(s) \) (i.e., degree of \( s \)) messages have to be delivered from \( s \) to \( r \) in order to complete the MST task, which implies a lower bound on any distributed MST algorithm.

Now let us give a high level description of the algorithm. Our CP-MST algorithm is based on Boruvka’s MST algorithm [17], and runs in \( O(\log n) \) phases, each consisting of several steps. The algorithm proceeds by maintaining a forest of tree fragments (initially singletons), and merging fragments until the forest converges to a single tree. Throughout the execution, each node has two officials,
namely, core nodes that represent it. In particular, recall that each node $v$ is assigned a representative core neighbor $r(v)$ passing information between $v$ and the core. In addition, $v$ is also managed by the leader $l(i)$ of its current fragment $i$. An important distinction between these two roles is that the representative of each node is fixed, while its fragment leader may change in each phase (as its fragment grows). At the beginning of each phase, every node knows the IDs of its fragment and its leader. Then, every node finds its minimum weight outgoing edge, i.e., the edge with the second endpoint belonging to the other fragment and having the minimum weight. This information is delivered to the core by the means of the representative nodes, which receive the information, aggregate it (as much as possible) and forward it to the leaders of the appropriate fragments. The leaders decide on the fragment merging and inform all the nodes about new fragments IDs.

The correctness of the algorithm follows from emulating Boruvka’s algorithm and the correctness of the fragments merging procedure, described in Appendix B. The main challenges in obtaining the proof were in bounding the running time, which required careful analysis and observations. There are two major sources of problems that can cause delays in the algorithm. The first involves sending information between officials (representatives to leaders and vice versa). Note that there are only $O(\sqrt{m})$ officials, but they may need to send information about $m$ edges, which can lead to congestion. For example, if more than $\alpha \cdot \sqrt{m}$ messages need to be sent to an officials of degree $\sqrt{m}$, then this will take at least $\alpha$ rounds. We use randomization of leaders and the property of clique emulation to avoid this situation and make sure that officials do not have to send or receive more than $O(\sqrt{m} \log m)$ messages in a phase. The second source for delays is the fragments merging procedure. This further splits into two types of problems. The first is that a chain of fragments that need to be merged could be long, and in the basic distributed Boruvka’s algorithm will take long time (up to $n$) to resolve. This problem is overcome by using a modified pointer jumping technique similar to [16]. The second problem is that the number of fragments that need to be merged could be large, resulting in a large number of merging messages that contain, for example, the new fragment ID. This problem is overcome by using randomization and by reducing the number of messages needed for resolving a merge. Full description of the algorithm along with the proofs of correctness and running time can be found in Appendix B.

4 Additional Algorithms in Core-Periphery Networks

In addition to MST, we have considered a number of other distributed problems of different types, and developed algorithms for these problems that can be efficiently executed on core-periphery networks. In particular, we dealt with the following set of tasks related to matrix operations. (M1) Sparse matrix transposition. (M2) Multiplication of a sparse matrix by a vector. (M3) Multiplication of two sparse matrices.
We then considered problems related to calculating aggregate functions of initial values initially stored one at each node in $V$. In particular, we developed efficient algorithms for the following problems. (A1) Finding the rank of each value, assuming the values are ordered. (As output, each node should know the rank of the element it stores.) (A2) Finding the median of the values. (A3) Finding the (statistical) mode, namely, the most frequent value. (A4) Finding the number of distinct values stored in the network. Each of these problems requires $\Omega(Diam)$ rounds on general networks, whereas on a $CP$-network it can be performed in $O(1)$ rounds.

An additional interesting task is defined in a setting where the initial values are split into disjoint groups, and requires finding the $r$ largest values of each group. This task can be used, for example, for finding the most popular headlines in each area of news. Here, there is an $O(r)$ round solution on a $CP$-network, whereas in general networks the diameter is still a lower bound.

In all of these problems, we also establish the necessity of all 3 axioms, by showing that there are network families satisfying 2 of the 3 axioms for which the general lower bound holds. Due to space limitation, we discuss in this section only one of these problems, namely, multiplication of a vector by a sparse matrix. Our results for the other problems can be found in Appendix D.

A few definitions are in place. Let $A$ be a matrix in which each entry $A(i,j)$ can be represented by $O(\log n)$ bits (i.e., it fits in a single message in the CONGEST model). Denote by $A_{i,*}$ (respectively, $A_{*,i}$) the $i$th row (resp., column) of $A$. Denote the $i$th entry of a vector $s$ by $s(i)$. We assume that the nodes in $C$ have IDs $[1,\ldots,n_C]$ and this is known to all of them. A square $n \times n$ matrix $A$ with $O(k)$ nonzero entries in each row and each column is hereafter referred to as an $O(k)$-sparse matrix.

Let $s$ be a vector of size $n$ and $A$ be a square $n \times n$ $O(k)$-sparse matrix. Initially, each node in $V$ holds one entry of $s$ (along with the index of the entry) and one row of $A$ (along with the index of the row). The task is to distributively calculate vector $s' = sA$ and store its entries at the corresponding nodes in $V$, such that the node that initially stored $s(i)$ will store $s'(i)$. We start with a claim on the lower bound (the proof can be found in Appendix D).

Claim 3. The lower bound for any algorithm for multiplication of a vector by a sparse matrix on any network is $\Omega(D)$, and on a $CP$-network is $\Omega(k/\log n)$.

Algorithm 1. The following algorithm solves the task in $O(k)$ rounds on a $CP$-network $G(V,E,C,P)$.

1. Each $u \in V$ sends the entry of $s$ it has (along with the index of the entry) to its representative $r(u) \in C$ (recall that if $u \in C$ then $r(u) = u$).
2. $C$ nodes redistribute the $s$ entries among them so that the node with ID $i$ stores indices $[1 + (n/n_C)(i - 1),\ldots,(n/n_C)i]$ (assume $n/n_C$ is integer).
3. Each $u \in V$ sends the index of the row of $A$ it has to $r(u) \in C$.
4. Each representative requests the $s(i)$ entries corresponding to rows $A_{i,*}$ that it represents from the $C$ node storing it.
5. Each representative gets the required elements of $s$ and sends them to the nodes in $P$ it represents.

6. Each $u \in V$ sends the products $\{A(i,j)s(i)\}_{j=1}^{n}$ to its representative.

7. Each representative sends each nonzero value $A(i,j)s(i)$ it has (up to $O(kn_c)$ values) to the representative responsible for $s(j)$, so it can calculate $s'(j)$.

8. Now, each node $u \in V$ that initially stored $s(i)$, requests $s'(i)$ from its representative. The representative gets the entry from the corresponding node in $C$ and sends it back to $u$.

We state the following results regarding the running time of Algorithm 1.

**Theorem 5.** On a CP-network $G(V,E,C,P)$, the multiplication of a $O(k)$-sparse matrix by a vector can be completed in $O(k)$ rounds w.h.p.

Before we start with the proof, we present the following theorem from [2].

**Theorem 6 ([2]).** Consider a fully connected system of $n_c$ nodes. Each node is given up to $M_s$ messages to send, and each node is the destination of at most $M_r$ messages. There exists algorithm that delivers all the messages to their destinations in $O\left(\frac{M_s + M_r}{n_c}\right)$ rounds w.h.p.

This theorem will be extensively used by our algorithms since it gives running time bound on messages delivery in a core that satisfies Axiom $A_E$. The result of the theorem holds with high probability which implies that it exploits a randomized algorithm. Nevertheless, our algorithms can be considered deterministic in the sense that all the decisions they make are deterministic. The randomness of the information delivery algorithm of Theorem 6 does not affect our algorithms since the decisions when and what message will be sent along with the message source and destination, are deterministically controlled by our algorithms.

**Proof of Theorem 5.** Consider Algorithm 1 and the CP-network $G(V,E,C,P)$. At Step 1, due to $A_B$ and $A_C$, each representative will obtain $O(n_c)$ entries of $s$ in $O(1)$ rounds. For Step 2, we use Theorem 6 with the parameters: $M_s = O(n_c)$ and $M_r = O(n/n_c)$, and thus such a redistribution will take $O((n_c + n/n_c)/n_c) = O(1)$ rounds. At Step 3, due to $A_B$ and $A_C$ each representative will obtain $O(n_c)$ row indices of $A$ in $O(1)$ rounds.

For Step 4, we again use Theorem 6 with the parameters: $M_s = O(n_c)$ (indices of rows each representative has), $M_r = O(n/n_c)$ (number of entries of $s$ stored in each node in $C$), and obtain the running time for this step: $O((n_c + n/n_c)/n_c) = O(1)$ rounds. At Step 5, each representative gets the required elements of $s$ which takes running time is $O(1)$ due to Theorem 6 and then sends them to the nodes in $P$ it represents which also takes $O(1)$ due to $A_C$.

Step 6 takes $O(k)$ rounds since $A$ has up to $k$ nonzero entries in each row. Step 7 again uses Theorem 6 with parameters $M_s = O(kn_c)$, $M_r = O(n/n_c)$, and thus the running time is $O(kn_c/n_c^2) = O(k)$.

At Step 8, a single message is sent by each node to its representative (takes $O(1)$ due to $A_C$), then the requests are delivered to the appropriate nodes in
and the replies with the appropriate entries of \( s' \) are received back by the representatives. All this takes \( O(1) \) rounds due to the Axiom \( A_E \) and Theorem \( \text{□} \). Then the entries of \( s' \) are delivered to the nodes that have requested them. Due to \( A_C \) this will also take \( O(1) \) rounds.

The following theorem shows the necessity of the axioms for achieving \( O(k) \) running time. The proof of the theorem can be found in Appendix C.

**Theorem 7.** For each \( X \in \{B, E, C\} \) there exists a family \( \mathcal{F}_X = \{G_X(V, E, C, P)(n)\} \) of \( n \)-node partitioned networks that do not satisfy Axiom \( A_X \) but satisfy the other two axioms, and input matrices of size \( n \times n \) and vectors of size \( n \), for every \( n \), such that the time complexity of any algorithm for multiplying a vector by a matrix on the networks of \( \mathcal{F}_X \) with the corresponding-size inputs is \( \Omega(n/\log n) \).

**References**

1. Lotker, Z., Patt-Shamir, B., Pavlov, E., Peleg, D.: Minimum-weight spanning tree construction in \( o(\log \log n) \) communication rounds. SIAM J. Computing **35**(1) (2005) 120–131
2. Lenzen, C., Wattenhofer, R.: Tight bounds for parallel randomized load balancing. In: STOC. (2011) 11–20
3. Lenzen, C.: Optimal deterministic routing and sorting on the congested clique. In: PODC. (2013) 42–50
4. Avin, C., Lotker, Z., Pignolet, Y.A., Turkel, I.: From caesar to twitter: An axiomatic approach to elites of social networks. CoRR abs/1111.3374 (2012)
5. Fujita, M., Krugman, P.R., Venables, A.J.: The spatial economy: Cities, regions, and international trade. MIT press (2001)
6. Krugman, P.: Increasing Returns and Economic Geography. The Journal of Political Economy **99**(3) (1991) 483–499
7. Holme, P.: Core-periphery organization of complex networks. Physical Review E **72** (Oct 2005) 046111
8. Liang, J., Kumar, R., Ross, K.W.: The fasttrack overlay: A measurement study. Computer Networks **50** (2006) 842 – 858
9. Basit, S., Schulzrinne, H.: An analysis of the skype peer-to-peer internet telephony protocol. In: INFOCOM. (2006) 1–11
10. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations and Advanced Topics. McGraw-Hill (1998)
11. Lynch, N.: Distributed Algorithms. Morgan Kaufmann (1995)
12. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM (2000)
13. Adamic, L.: The small world web. Research and Advanced Technology for Digital Libraries (1999) 852–852
14. Bonanno, G., Caldarelli, G., Lillo, F., Mantegna, R.: Topology of correlation-based minimal spanning trees in real and model markets. Phys. Rev. E **68** (2003)
15. Chen, C., Morris, S.: Visualizing evolving networks: Minimum spanning trees versus pathfinder networks. In: INFOVIS. (2003) 67–74
16. Lotker, Z., Patt-Shamir, B., Peleg, D.: Distributed MST for constant diameter graphs. Distributed Computing **18**(6) (2006) 453–460
17. Nesetril, J., Milkova, E., Nesetrilova, H.: Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. Discrete Mathematics 233(1 - 3) (2001) 3 – 36
18. Wyllie, J.: The complexity of parallel computations. Technical report. Dept. of Computer Science, Cornell University (1979)
19. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge Univ. Press (2005)
20. Soltys, K.: The hardness of median in the synchronized bit communication model. In: Proc. TAMC. (2011) 409–415
APPENDIX

A Proofs for Section 2

Claim 1 (restated). If the partitioned network $G(V,E,C,P)$ satisfies Axioms $A_E$ and $A_C$ then its diameter is $\Theta(1)$.

Proof. Suppose the partitioned network $G(V,E,C,P)$ satisfies Axioms $A_E$ and $A_C$. Let $u,v$ be two nodes in $V$. There are 3 cases to consider. (1) Both $u,v \in C$: Then Axiom $A_E$ ensures $O(1)$ time message delivery and thus $O(1)$ distance. (2) $u \in P$ and $v \in C$: Axiom $A_C$ implies that there must be a node in $w \in C$ such that $d(u,w) = O(1)$. By Axiom $A_E$, $d(w,v) = O(1)$, thus $d(u,v) \leq d(u,w) + d(w,v) = O(1)$. (3) Both $u,v \in P$: Then there must be $w,x \in C$ such that $d(u,w) = O(1)$ and $d(x,v) = O(1)$. Since $d(w,x) = O(1)$ by Axiom $A_E$, it follows that $d(u,v) \leq d(u,w) + d(w,x) + d(x,v) = O(1)$. Hence $d(u,v) = O(1)$ for any $u,v \in V$, so the diameter of $G(V,E)$ is constant. □

Claim 2 (restated). For $X \in \{E,C\}$, there exists a family of $n$-node partitioned networks $G_X(V,E,C,P)$ of diameter $\Omega(n)$ that satisfy all axioms except $A_X$.

Proof. For $X = C$, let $G_C(V,E,C,P)$ be the lollipop partitioned network $L_n$. As mentioned before, for this network Axiom $A_C$ is violated while the other are not. Also note that the diameter of $G_X$ is $\Omega(n)$.

For $X = E$, let $G_E(V,E,C,P)$ be the sun partitioned network $S_n$. As mentioned before, for this network Axiom $A_E$ is violated while the other are not. Also note that the diameter of $G_E$ is $\Omega(n)$. □

B Algorithm for MST

B.1 Description of the $\mathcal{CP}$-MST algorithm

Let us now describe the phases of our algorithm.

Phase 0 – Initialization

1. Obtaining a Representative. Each node $u \in V$ obtains a representative $r(u) \in C$ in the core.
2. Renaming. Each node $u \in V$ receives a unique ID, $\text{id}(u) \in [1, \ldots, n]$.
3. Fragment ID Initialization. Each node $u \in V$ forms a singleton fragment with its unique $\text{id}(u)$.
4. Obtaining a Leader. Each fragment $i$ obtains a (random) leader $l(i) \in C$ in the core.
5. Fragment State Initialization. Each leader keeps a state (active/frozen/root/waiting) for each of its fragments. The initial state of all fragments is active.
Before describing the subsequent phases of the algorithm a few definitions are in place. Throughout, let $f(u)$ denote the fragment that $u$ belongs to. Dually, let $V^i$ denote the set of nodes in fragment $i$ and let $V^i(w)$ denote the subset of $V^i$ consisting of the nodes that are represented by $w$. For a representative $w \in \mathcal{C}$, let $F_{rep}(w)$ be the set of fragments that $w$ represents, namely, $F_{rep}(w) = \{i \mid V^i(w) \neq \emptyset\}$, and let $F_{lead}(w)$ be the set of fragments that $w$ leads, namely, $F_{lead}(w) = \{i \mid l(i) = w\}$. For a set of nodes $S^i$ belonging to the same fragment $i$, an outgoing edge is one whose second endpoint belongs to a different fragment. Let $\text{mwoe}(S^i)$ be the minimum weight outgoing edge of $S^i$. For a node $u$, a fragment $i$ and a representative $w$, we may occasionally refer to the fragment’s $\text{mwoe}$ as either $\text{mwoe}(u)$, $\text{mwoe}(V^i)$ or $\text{mwoe}(V^i(w))$. The merge-partner of fragment $i$, denoted $\text{mp}(i)$, is the fragment of the second endpoint of the edge $\text{mwoe}(V^i)$. Define $F_{lead}^j(w) \subseteq F_{lead}(w)$ to be the set of fragments led by $w$ that attempt to merge with the fragment $j$, i.e., $F_{lead}^j(w) = \{i \mid i \in F_{lead}(w) \land \text{mp}(i) = j\}$. Define a speaker fragment $\text{spk}(w) = \text{min}(F_{lead}(w))$, that is responsible for sending merge-requests on behalf of all the fragments in $F_{lead}^j(w)$ and updating them upon the reception of merge-replies. We now proceed with the description of the algorithm.

Phase $b \in [1\ldots B]$ (similar to Boruvka’s phases):

1. **Finding $\text{mwoe}$**. Each $u \in V$ finds an edge $(u,v) = \text{mwoe}(u)$ and obtains $f(v)$ and $l(f(v))$.

2. **Periphery to Representatives**. Each node $u \in V$ sends to its representative $r(u) \in \mathcal{C}$ its $(u,v) = \text{mwoe}(u)$, $f(u)$, $l(f(u))$, $f(v)$ and $l(f(v))$.

3. **Representatives to Leaders**. Each representative $w \in \mathcal{C}$, for each fragment $i \in F_{rep}(w)$, sends $(u,v) = \text{mwoe}(V^i(w))$, $i$, $f(v)$, and $l(f(v))$ to the leader $l(i)$ of $i$.

4. **Leaders Merge Fragments**. Each leader $w \in \mathcal{C}$, for each fragment $i \in F_{lead}(w)$, finds $(u,v) = \text{mwoe}(V^i)$ and $\text{mp}(i) = f(v)$, and then executes $\text{MERGEFRAGS}(i)$.

5. **Leaders to Representatives**. Each leader $w \in \mathcal{C}$, for each active fragment $i \in F_{lead}(w)$, sends an update message with the new fragment name $\text{newID}(i)$, the new leader node $l(\text{newID}(i))$ and the edge to add to the MST to all the representatives of the nodes in $V^i$. If $w \neq l(\text{newID}(i))$, then the fragment $i$ is removed from $F_{lead}(w)$.

6. **Representatives to Periphery**. Each representative $w \in \mathcal{C}$, for each $i \in F_{rep}(w)$ for which the update message with $\text{newID}(i)$ and $l(\text{newID}(i))$ was received, forwards it to all the nodes $V^i(w)$.

### B.2 MergeFrags procedure

The $\text{MERGEFRAGS}$ procedure is the essential part of our algorithm, executed at each phase $b$. The procedure is executed by each leader $w \in \mathcal{C}$ for each fragment $i \in F_{lead}(w)$. For a fragment $i$, its leader maintains a state parameter $\text{state}(i) \in \{\text{active, frozen, root, waiting}\}$. Each fragment $i$ attempts to merge with some
other fragment \( mp(i) \). Towards that, the leader of \( i \) initiates a merge-request to a leader of the fragment \( mp(i) \) (the fragment at the other end of \( mwoe(i) \)). Since these requests do not have to be reciprocal, merge requests usually form a *merge-tree* whose nodes are fragments and whose directed edges represent merge-requests (see Figure 2 for illustration). In order to minimize the number of merge-request messages sent by fragment leaders we propose to designate, for each set of fragments sharing the same leader that attempt to merge with the same target fragment, a *speaker* fragment that will act on behalf of all the fragments in the set, and update all of them upon reception of merge-replies.

The root of that tree is a fragment that received a reciprocal merge-request (actually there are two such fragments, so the one with the smaller ID is selected as a root). However, since merge-requests are not sent by every fragment but only by *speakers*, the root node is detected by the *speaker* and not the root fragment itself (except for the case when the root is the *speaker*). For example, in Figure 2 fragment 4 sends a merge request to fragment 10 and gets a merge-reply with the *next* pointer of 10, which is 8 (the *next* pointer of fragment \( i \) is always set initially to \( mp(i) \)). Fragment 4 then realizes that 8 belongs to \( F_{lead(w)}^{10} \) and thus identifies the reciprocity between 8 and 10. Fragment 4 (the *speaker*) then notifies 8 that it should be the root (7 does not notify 10 since \( 8 < 10 \)). For a detailed description of the root finding procedure see Algorithm 3 in the
Appendix. So, when a fragment \(i\) that is led by \(w\) is in the active state and attempts to merge with another fragment \(\text{mp}(i)\), it first tries to find the root using the procedure \textsc{FindRoot}. By the end of the \textsc{FindRoot} procedure, \(i\) may not find a root, in which case its state will become frozen; \(i\) may find that the root is another fragment \(k\) in \(F_{\text{lead}}(w)\), and then \(i\) will notify \(k\), but \(i\)'s state will become frozen; \(i\) may find that it is a root by itself, in which case its state will become root; and finally, \(i\) may be notified by a speaker of \(F_{\text{lead}}(w)\) and \(i\)'s state will become root.

Once a fragment enters the root state, it starts waiting for all the tree fragments to send it merge-requests. These merge-requests are sent by each fragment using the pointer-jumping procedure \textsc{PJ} (see Algorithm 2 in the Appendix), while it is in the frozen state. Once the requests of all the tree fragments reach the root (using pointer-jumping), it chooses a new random ID (\textit{newID}) for the fragment among all the fragments in the tree and a random Core node to be the new leader (\textit{newLead}) for this fragment, and sends this information back to all of them. At this point, the merge-tree is considered to be resolved and all its fragments (including the root) change their state to active. The simple state diagram of Algorithm \textsc{MergeFrags} can be found in Figure 3 and a detailed pseudocode in Algorithm 1 in the Appendix.

![Fig. 3: State diagram of Algorithm \textsc{MergeFrags}. The algorithm is executed by every leader for every fragment it leads. Each title indicates an event and the text below it is the action performed upon that event.](image)

Now we briefly describe the pointer-jumping approach used to resolve fragment trees, or simply \textit{merge-trees}. Pointer-jumping is a technique (developed
Algorithm 1 MergeFrags(i)

Executed every phase by every leader \( w \in C \) for each fragment \( i \in F_{\text{lead}}(w) \).

1: if \( \text{state}(i) = \text{active} \) then
2: \( \text{next} \leftarrow \text{mp}(i) \)
3: \( \text{state}(i) \leftarrow \text{frozen} \)
4: \( \text{[isFound, rootFrag]} \leftarrow \text{FindRoot}(i) \)
5: if \( \text{isFound} = \text{true} \) then
6: \( \text{state(rootFrag)} \leftarrow \text{root} \)
7: if \( \text{state}(i) = \text{frozen} \) then
8: \( \text{[isFinished, next]} \leftarrow \text{PJ}(i, \text{next}, 2) \)
9: \( \text{next}(F_{\text{mp}}(i)) \leftarrow \text{next} \)
10: if \( \text{isFinished} = \text{true} \) then
11: \( \text{state}(F_{\text{mp}}(i)) \leftarrow \text{waiting} \)
12: if \( \text{state}(i) = \text{waiting} \) then
13: receive merge-requests
14: send merge-replies with \( \text{next} \) (which now points to the \text{root})
15: wait for FIN msg from \text{root} with \text{newID} and \text{newLead}
16: if FIN msg received then
17: \( \text{newID}(F_{\text{mp}}(i)) \leftarrow \text{newID} \)
18: \( \text{state}(F_{\text{mp}}(i)) \leftarrow \text{active} \)
19: if \( \text{state}(i) = \text{root} \) then
20: wait for incoming merge-requests
21: store the sources of the requests
22: reply on all requests with \text{NULL}
23: if num of requests = 0 and size of merge-tree \( \leq 2^{2+\text{phase}} \) then
24: \( \text{newID} \leftarrow \text{random ID among all fragments in the merge-tree} \)
25: \( \text{newLead} \leftarrow \text{random node in } C \)
26: send FIN msg with \text{newID} and \text{newLead} to all the stored sources
27: \( \text{state}(i) \leftarrow \text{active} \)

in [18] and often used in parallel algorithms) for contracting a given linked list of length \( n \), causing all its elements to point to the last element, in \( O(\log n) \) steps. We use the pointer-jumping approach for resolving fragments trees, viewing the fragments as the nodes in a linked list with each fragment \( i \) initially pointing at \( \text{mp}(i) \). Each fragment chain can be of length at most \( O(n) \), and thus can be contracted in \( \log n \) rounds, resulting in \( \log n \) time overhead per phase. In order to overcome this, we use a technique first introduced in [16], called “amortized pointer-jumping”, in which the contraction of long chains is deferred to later phases, while in each phase only a small constant number of pointer-jumps is performed. The argument for the correctness of this approach is that if the chain (or tree) is large, then the resulting fragment, once resolved, is large
Algorithm 2 PJ(i, next, iter) (pointer jumping)

Executed by each fragment i in the frozen state

Input: next – first fragment to try, iter – how many pointer-jumps to perform

Output 1: indication whether the root was reached
Output 2: pointer to the root or to the next fragment in the chain

1: while iter > 0 do
2: if i = spk(i)(w) then
3: send merge-request to next
4: receive merge-requests
5: send merge-replies with next
6: if i = spk(i)(w) then
7: receive merge-reply with next'
8: if next' = NULL then
9: return [TRUE, next]
10: next ← next'
11: iter ← iter − 1
12: return [FALSE, NULL]

Algorithm 3 FindRoot(i)

1: if i = spk(i)(w) then
2: send merge-request to mp(i)
3: receive merge-requests
4: send merge-replies with mp(i)
5: if i = spk(i)(w) then
6: receive merge-reply with next'
7: if next' ∈ Flead(i)(w) then
8: if next' ≤ mp(i) then
9: return [TRUE, next']
10: return [FALSE, NULL]

enough to satisfy the fragment growth rate needed to complete the algorithm in

B = O(log n) phases (see Claim 7).

B.3 Correctness of the CP-MST Algorithm

We now show that our CP-MST algorithm is correct, i.e., it results in an MST.
The following claim shows that the MergeFrags algorithm indeed resolves a merge-tree.
Claim 4. Once a fragment tree becomes active, all the (old) fragments in the tree have the same (new) fragment ID.

Proof. This follows directly from the description above, and the observation that in the pointer-jumping procedure, at every step at least one more node points to the root. Thus, if at some phase the root of the fragment tree does not receive any merge-request, then every other fragment in the tree is in waiting state, i.e., points to the root. Consequently, the root knows all the fragments in the tree and can inform their leaders about the new fragment ID, newID, and the new leader node l(newID).

Claim 5. The CP-MST algorithm emulates Boruvka’s MST algorithm and thus constructs an MST for the network.

Proof. In Boruvka’s algorithm, fragment merges can be performed in any order. What’s important is that a merge between any two fragments will occur if and only if they share an edge that is an mwoe for at least one of the fragments. Since our algorithm satisfies this property, it results in an MST.

B.4 Running time analysis of the CP-MST algorithm

We now analyze the run-time of Algorithm CP-MST in a Core-Periphery network.

Theorem 3 (restated). On a CP-network G(V,E,C,P), Algorithm CP-MST terminates in O(log^2 n) rounds with high probability.

In order to prove this theorem, we analyze each part of the algorithm separately. The Theorem will follow directly by proving the following Claims 6, 7 and 8. We start with the initialization phase.

Claim 6. The initialization phase (Phase 0) takes O(1) rounds.

Proof. In the first step of the initialization phase, each node u ∈ V has to obtain a representative r(u) ∈ C. If u ∈ C, it represents itself, i.e., r(u) = u. Each periphery node u ∈ P sends a “representative-request” message towards the core C with its ID. This is done in parallel using a γ-covercast protocol on P and C, which ensures that each such message is received by some node in C. Once a node w ∈ C receives such a message, it replies to u on the same route and u sets r(u) = w. Due to Axiom AC, this covercast process requires O(1) rounds.

Next consider the renaming step. This step can be performed in the following simple way: each node sends to its representative its ID, and each representative sends its own ID and the number of nodes it represents to all core members. Note that this can be done in O(1) rounds due to Axiom AE. Now, every core member can sort the core IDs and reserve a sufficiently large range of IDs for each representative. Each node in the core can now set its own new ID and send
unique new IDs in the range \([1 \ldots n]\) to the nodes it represents. We assume nodes in the core \(C\) take IDs \([1 \ldots n_C]\).

All nodes use their IDs to set their initial fragment number. Each initial fragment \(i = f(u)\) (which is a singleton at this phase) obtains a leader by asking the representative \(r(u)\) of node \(u\) to select a random core member \(w\) uniformly at random and declare it as a leader of \(i\), \(l(i) = w\). This is done in a balanced way by picking a random permutation and assigning leaders according to it. This operation requires \(O(1)\) steps and every node in \(C\) becomes the leader of \(O(n_c)\) fragments.

Now we show that the number of Boruvka phases needed to accomplish \(CP\)-MST is \(B = O(\log n)\).

**Claim 7.** Our MST algorithm takes \(O(\log n)\) phases, i.e., \(B = O(\log n)\).

**Proof.** The proof is by induction. Assume that every active fragment \(f\) at phase \(x \leq i\) has size (in nodes) \(|f| > \min(2^x, n)\). We show that in the phase \(j > i\) at which \(f\) becomes active again, its size will be at least \(\min(2^j, n)\). In phase \(i\), \(f\) joins a fragments tree that was created at some phase \(k \leq i\), and according to induction assumption, every fragment in this tree has size at least \(\min(2^k, n)\). That tree will be resolved in phase \(j\), i.e., after \(j - k\) phases. Let \(D\) be the diameter of the tree in phase \(j\). Since the algorithm uses pointer jumping with 2 iterations at each phase, it follows that \(j - k \leq \frac{\log D}{2}\). The size of the resolved tree is at least \(\min(2^k, D)\) since it comprises of at least \(D\) fragments each of size at least \(\min(2^k, n)\). Clearly,

\[
2^k D = 2^{k + \log D} \geq 2^k + \left\lceil \frac{\log D}{2} \right\rceil \geq 2^j,
\]

and thus \(|f| \geq \min(2^j, n)|. So each active fragment at phase \(j\) is of size at least \(\min(2^j, n)\). If in phase \(\log n\) there are no active fragments, then the algorithm waits for at most \(\log n\) time, which is sufficient to resolve any fragments tree, and then, the size of the fragment is \(\min(2^{\log n}, n) = n\), which means that the algorithm has terminated. 

Finally, we analyze the steps performed in phases \(b \in [1, \ldots, B]\).

**Claim 8.** For every phase \(b \in [1, \ldots, B]\), the run-times of the 6 main steps are bounded as follows:

1. **Finding own mwoe** – \(O(1)\).
2. **Periphery to Representatives** – \(O(1)\).
3. **Representatives to Leaders** – \(O(\log n)\).
4. **Leaders Merge Fragments** – \(O(\log n)\).
5. **Leaders to Representatives** – \(O(\log n)\).
6. **Representatives to Periphery** – \(O(1)\).

Thus, every phase \(b\) takes \(O(\log n)\) rounds.
Proof. Before embarking on the proof, we give the following auxiliary lemma. The result of this lemma is well known and its proof is analogous to the proof of Lemma 5.1 in [10].

**Lemma 1.** When up to $O(n \log n)$ balls are thrown independently and uniformly at random into $\Omega(n)$ bins, the maximum loaded bin has $O(\log n)$ balls with probability at least $1 - \frac{1}{n^8}$.

**Proof.** Let $X_i$ be the random variable representing the number of balls in bin $i$, and $k_1, k_2, k_3, k_4,$ and $\alpha$ are constants. Then, assuming that number of balls is $k_1 n \log n$ and number of bins is $k_3 n$:

\[
\Pr(X_i \geq M) \leq \left(\frac{k_1 n \log n}{M}\right) \cdot \left(\frac{1}{k_3 n}\right)^M \leq \frac{(k_1 n \log n)^M}{M!} \cdot \frac{1}{(k_3 n)^M} \leq \frac{(k_4 \log n)^M}{M!} \cdot \frac{1}{(k_3 n)^M}, \quad \text{where } k_4 = k_1/k_3
\]

(2) \hspace{2cm} (3) \hspace{2cm} (4)

By taking $M = \alpha \log n$ we obtain:

\[
\Pr(X_i \geq \alpha \log n) \leq \left(\frac{k_4 e}{\alpha}\right)^{\alpha \log n}.
\]

(5)

If $\alpha = k_2 e(k_4 e + 1)$:

\[
\Pr(X_i \geq k_2 e(k_4 e + 1) \log n) \leq \left(\frac{1}{e}\right)^{k_2 e(k_4 e + 1) \log n} = \frac{1}{e^{k_2 e}}.
\]

(6) \hspace{2cm} (7)

By taking union bound over all the $n$ bins we obtain that the probability that any bin has at most $k_2 e(k_4 e + 1) \log n = O(\log n)$ balls with probability of at most $n \cdot \frac{1}{n^8} \leq \frac{1}{n^8}$ for $k_2 \geq 4$. \qed

Now we are ready to proceed with the proof.

1. **Finding own mwoe.** In this step every node sends a single message to all its neighbors, so the running time is $O(1)$.

2. **Informing Representatives.** Each node $u \in V$ sends $\text{mwoe}(u)$ to $r(u) \in C$ using $\gamma$-covercast. By Axiom $A_C$, the running time is $O(\gamma) = O(1)$.

3. **Informing Leaders.** Since the network satisfies Axiom $A_E$, one may assume that $C$ is a clique. To derive the running time of this step we have to calculate how many messages are sent between a representative $u$ and a leader $v$ in $C$. It suffices to look only at Core edges, since this step involves communication only between nodes in $C$ (representatives and leaders). Using Theorem 2 we
have that $d_{out}(u) = O(n_c)$, and since $\mathcal{P}$ and $\mathcal{C}$ form a $\Theta(1)$-convergecaster, it follows that on each edge towards $\mathcal{P}$, $u \in \mathcal{C}$ receives a constant number of “representative-requests” at the initialization phase. The last claim implies that $u$ represents $O(n_c)$ nodes and thus at most $O(n_c)$ fragments. Hence every representative node has to send $O(n_c)$ messages, each destined to a leader of a specific fragment. Since the “leadership” on a fragment is assigned independently at random to the nodes in $\mathcal{C}$, sending messages from representative to leaders is analogous to throwing $O(n_c)$ balls into $n_c$ bins. Hence using Lemma 1 it follows that the most loaded edge (bin) from a representative $u$ to some leader $v$ handles $O(\log n_c)$ messages (balls), with probability at least $1 - \frac{1}{n_c^2}$. Applying the union bound over all $O(n_c)$ representative nodes and all $O(\log n) = O(\log n_c)$ phases of the algorithm, we get that the most loaded edge in step 3 (Informing Leaders) is at most $O(\log n)$ with probability at least $1 - \frac{1}{n}$. Thus, this step takes $O(\log n)$ time.

4. Leaders Merge Fragments. Every execution of Procedure MergeFrags requires sending/receiving merge-request/reply messages from every leader $u \in \mathcal{C}$, for each fragment $i \in F_{lead}(u)$. For each merge-request there is exactly one merge-reply, thus it suffices to count only merge-requests. Moreover, if there are multiple fragments that reside at the same leader node and need to send a merge-request to the same other fragment, only one message will actually be sent by the speaker fragment. The last observation implies that every request message sent by a leader is destined to a different fragment (i.e., to its leader). As in the analysis of the previous step, since “leadership” is assigned independently at random to the nodes in $\mathcal{C}$, sending messages from leaders to leaders is analogous to throwing $A$ balls into $n_c$ bins, where $A$ is the number of fragments that the node $u$ leads. Using the similar “balls and bins” argument, $A$ can be bounded w.h.p. by $O(\sqrt{n})$ – up to $n$ fragments (balls) are assigned to $n_c = \Omega(\sqrt{n})$ Core nodes (bins).

We now apply Lemma 1 with $O(\sqrt{n})$ balls (fragments led by a node) and $n_c = \Omega(\sqrt{n})$ bins (edges towards other Core nodes), and conclude that the most loaded edge from a leader $u$ to some other leader $v$ carries $O(\log n)$ messages, with probability at least

$$1 - \frac{1}{(\sqrt{n})^8} = 1 - \frac{1}{n^4}.$$

Applying the union bound over all the $O(\sqrt{n})$ leaders and all $O(\log n)$ phases of the algorithm, we get that the most loaded edge in the process of sending merge-requests carries at most $O(\log n)$ messages with probability at least $1 - \frac{1}{n}$.

The last part of step 4 (Leaders Merge Fragments) is when the root fragment sends the FIN (“finish”) message to all the merge-tree members. the size of each active fragment at the beginning of phase $j$ is at least $2^j$ (see Claim 7) and at most $2^{j+1}$ (as the root does not release a tree at phase $j - 1$ if it is too large). Thus, the number of merge-trees resolved at phase $i$ is at most $\frac{n}{2^i}$ (every resolved tree becomes an active fragment at the next phase). In
case $2^{i+1} \leq \sqrt{n}$, it follows from the “balls and bins” argument that a leader node $u \in C$ has at most $O(\sqrt{n \log n})$ roots. For each root, a leader has to send a message for each fragment in its tree. The number of fragments in the tree is bounded by the number of nodes in the tree, which is $2^{i+1}$ (this is because the tree becomes an active fragment at the beginning of the next phase $j = i + 1$ and its size is limited by $2^{i+1}$). Thus, a leader has to send $O(\sqrt{n \log n}) \cdot 2^{i+2} = O(\sqrt{n \log n})$ messages. Each message is destined to a leader of some fragment which is located at the randomly chosen node in $C$. So the same “balls and bins” arguments yields that the most loaded edge carries $O(\log n)$ messages with high probability.

In case $2^{i+1} > \sqrt{n}$, the “balls and bins” argument yields that a leader $u \in C$ has at most $O(\log n)$ roots. Now, since a root has to send at most one message to each leader (even if the node is a leader of multiple fragments of the tree), the total number of messages needed to be sent by a leader is $O(n_c \log n)$. Since every message is destined to a random leader, we obtain a bound of $O(\log n)$ on the maximum edge load, with high probability.

Thus, step 4 (Leaders Merge Fragments) takes $O(\log n)$ time.

5. **Informing Representatives.** Obviously, this step takes the same time ($O(\log n)$) as step Informing Leaders, since it involves the transfer of the same amount of information (except in the opposite direction).

6. **Informing Periphery.** This step takes the same time ($O(1)$) as the Informing Representatives step, since again it involves transferring the same amount of information (in the opposite direction).

\[ \square \]

### B.5 Axiom necessity for the CP-MST algorithm

**Theorem 4** (restated). For each $i \in \{1, 2, 3\}$ there exist a family $\mathcal{F}_i = \{G_i(V, E, C, P)(n)\}$ of partitioned networks that do not satisfy Axiom $A_i$ but satisfy the other two axioms, and the time complexity of any distributed MST algorithm on $\mathcal{F}_i$ is $\Omega(n^{\alpha_i})$ for some constant $\alpha_i > 0$.

**Proof.** 1. Consider a graph $G_2$ on Figure 3 (Left) in which Core is a clique of size $k$ and each node in the Core is connected to $k^3$ Periphery nodes (one node in Core is also connected to $s$, so it has $k^3 + 1$ Periphery neighbors). The number of nodes in $G_2$ is thus: $n = k + k \cdot k^3 + 1 = \Theta(k^4)$.

In [16], it was shown that any distributed algorithm will take at least $\Omega(\frac{\sqrt{n}}{\sqrt{\log n}})$ time on $G_1$. In the graph $G_1$, Core is a clique, thus $G_1$ satisfies Axiom $A_E$. Since every node in Periphery has a direct edge to the Core, we can say that $G_1$ satisfies Axiom $A_C$, i.e., it is possible to perform a convergcast in $O(1)$. But notice that $d_{in} = \sqrt{n}$ while $d_{out} = \sqrt{n^3}$ and thus $G_1$ does not satisfy Axiom $A_B$. 
2. Consider a graph $G_3$ on Figure 5 (Left) in which Core is a collection of $k$ cliques, each of size $k$, where a single node in each clique is connected to a special Core node $u$, and there are no edges between cliques. Each node in the Core (except of $u$) is connected to $k$ Periphery nodes, such that all nodes in specific clique are connected to the Periphery nodes that reside in a specific column. One Core node (from the leftmost clique) is additionally connected to $s$ and another Core node (from the rightmost clique) is connected to $r$. The number of nodes in $G_3$ is thus: $n = k \cdot k + k \cdot k^2 + 2 = \Theta(k^3)$.

Let’s take a look at the nodes $s$ and $r$ and assume the following weights assignment. All the edges between Core and Periphery have weights 10, except of the two edges that come from $s$ and $r$. Weights of all the edges incident to $s$ are 2 and weights of all the edges incident to $r$ are 3. Assume also that all the rest weights in Periphery are 1. Easy to see that such weights will yield MST illustrated in Figure 5 (Right). Notice that if the weight of some edge incident to $s$ will be increased (let’s say to 5) this will cause this edge to be removed from the MST and the appropriate edge incident to $r$ to be included. Thus, in order to allow $r$ to know which of its edges belong to the MST, it needs to receive information regarding all the edges incident to $s$, i.e., at least $k^2$ messages should be delivered from $s$ to $r$. Next we will show that delivering $k^2$ messages from $s$ to $r$ will require at least $k/2$ time.

First, note that any path $s \rightarrow r$ that is not passing via the node $u$ has length of at least $k$, thus we can assume that all the messages will take paths via $u$. Second, we can see that the edge cut of the node $u$ (i.e., its degree) is $k$ and thus, after $k/2$ time it will forward at most $k^2/2$ messages, which is not sufficient for completing the MST task. Thus, any MST algorithm on the graph $G$ will take at least $\Omega(k) = \Omega(\sqrt{n})$ time.

It is left to show that $G_3$ satisfies Axioms $A_B$ and $A_C$, but not $A_E$. For every node in the Core, $d_{in} = k$ and $d_{out} = k$ except the node $u$ for which $d_{out} = 0$. So, for each node in the core $\frac{d_{out}}{d_{in} + 2} = O(1)$ which means that $A_B$ is satisfied. Since every node in Periphery has a direct edge to the Core, we can say that $G_3$ satisfies Axiom $A_C$, i.e., it is possible to perform a convergcast in $O(1)$. It is also easy to see that in the Core there is no $O(1)$-clique emulation ($A_E$), since in order to send $k$ messages out of any clique in Core to any other clique in Core we need $k$ time since there is only one edge from any clique towards the node $u$ that interconnects cliques.

3. Consider a graph $G_4$ on Figure 4 (Right) in which Core is a clique of size $k$ and each node in the Core is connected to $k/2$ Periphery nodes. One Core node is additionally connected to a cycle of size $k^2/2$ that resides in Periphery. The number of nodes in $G_4$ is thus: $n = k + k \cdot k/2 + k^2/2 = \Theta(k^2)$.

It is easy to see that Axioms $A_B$ and $A_E$ are satisfied, but Axiom $A_C$ is not. For some weights assignment, the decision regarding which edge of $r$ to include in the MST depends on the weights of the edges incident to $s$. The last implies that at least one message has to be delivered from $s$ to $r$ which will take $\Omega(k^2) = \Omega(n)$ time.

□
Fig. 4: Left: Graph $G_2$ – each node in the Core is connected to $k^3$ Periphery nodes. Right: Graph $G_4$ – each node in the Core is connected to $k/2$ Periphery nodes, and one Core node is connected to cycle of length $k^2/2$ in Periphery.

Fig. 5: Left: Graph $G_3$ – Core consists of $k$ cliques each of size $k$. Each node in the Core (except $u$) is connected to $k$ nodes in Periphery. Right: Possible MST of $G_3$.

C Proofs for Section 4

Claim 3 (restated). The lower bound for any algorithm for multiplication of a vector by a sparse matrix on any network is $\Omega(D)$, and on a CP-network is $\Omega(k/\log n)$.

Proof. First, we show the $\Omega(n)$ lower bound for an arbitrary network. In order to obtain $s'(1)$, we need at least to multiply $s(1)$ by $A(1, 1)$ (assuming $s(1) \neq 0$ and $A(1, 1) \neq 0$). But $s(1)$ and $A(1, 1)$ can be located at different nodes $u$ and $v$ with distance $d(u, v) = D$. Thus, it will take at least $\Omega(D)$ rounds to bring $s(1)$ and $A(1, 1)$ together.
Now we are going to show that for any \( CP \)-network, the lower bound is \( \Omega(k/\log n) \) rounds. Consider a \( CP \)-network as in Figure 1 (I). Let \( u \) be a node in \( P \) add its degree is 1. Let \( v \) be any other node in \( V \). Assume that \( u \) initially stores the row \( A_{1,*} \) and the entry \( s(1) \), while \( v \) stores row \( A_{2,*} \) and the entry \( s(2) \).

Next, we show a reduction from the well-known equality problem (EQ), in which two parties are required to determine whether their input vectors \( x, y \in \{0,1\}^k \) are equal. Assuming we are given a procedure \( P \) for our vector by matrix multiplication problem, we use it to solve the EQ problem. Given input vectors \( x, y \) for the EQ problem (at \( u \) and \( v \) respectively), we create an input for the vector by matrix multiplication problem in the following way. Node \( u \) assigns \( A(1, i) = x(i) \) for every \( i \in \{1, \ldots, k\} \) and \( s(1) = 1 \), while node \( v \) assigns \( A(2, i) = y(i) \) for every \( i \in \{1, \ldots, k\} \) and \( s(2) = 1 \). All the other entries of \( A \) and \( s \) are initialized to 0. It follows that \( s'(i) = \sum_{j=1}^{n} s(j)A(j, i) = A(1, i) + A(2, i) = x(i) + y(i) \) for every \( i \in \{1, \ldots, k\} \). Given the value of \( s'(i) \), one can decide whether \( x(i) = y(i) \) for every \( i \in \{1, \ldots, k\} \), since clearly \( x(i) = y(i) \) if \( s'(i) \in \{0,2\} \) (and otherwise \( s'(i) = 1 \)). Notice that the vector \( s' \) is stored distributedly in the network – one entry in each node. But the indication to \( u \) and \( v \) whether all the entries are in \( \{0,2\} \) can be delivered in \( O(1) \) rounds in the following way. Each node in \( P \) sends its entry of \( s' \) to its representative who checks all the received entries and sends an indication bit to all the other nodes in \( C \). So, every node in \( C \) knows now whether all the entries in \( s' \) are in \( \{0,2\} \) (actually, we are interested only in the first \( k \) entries). Representatives can now inform the nodes in \( P \) they represent in \( O(1) \) rounds. It follows that using procedure \( P \) one can solve the EQ problem, which is known to require at least \( k \) bits of communication. Therefore, assuming that each message is \( O(\log n) \) bits, it is required \( \Omega(k/\log n) \) communication rounds.

\[ \square \]

**Theorem 7** (restated). For each \( X \in \{B, E, C\} \) there exist a family \( F_X = \{ G_X(V, E, C, P) \mid n \} \) of partitioned networks that do not satisfy Axiom \( A_X \) but satisfy the other two axioms, and input matrices of size \( n \times n \) and vectors of size \( n \), for every \( n \), such that the time complexity of any algorithm for multiplying a vector by a matrix on the networks of \( F_X \) with the corresponding inputs is \( \Omega(n/\log n) \).

**Proof.** Now we show the necessity of the Axioms \( A_B, A_E \) and \( A_C \) for achieving \( O(k) \) running time. Consider the following cases where in each case one of the axioms is not satisfied while the other two satisfied.

**Necessity of \( A_B \):** Consider the family of dumbbell partitioned networks \( D_2 \). As discussed earlier, Axiom \( A_B \) is violated while the other hold. Let us denote the core’s nodes as \( u \) and \( v \). In \( O(k) \) rounds \( u \) (resp. \( v \)) can collect all the rows of \( A \) and entries of \( s \) stored at the nodes of \( P \) connected to \( u \) (resp. \( v \)). So, assuming \( n/2 \) is integer, after \( O(k) \) rounds, \( u \) and \( v \) have each \( n/2 \) rows of \( A \) and \( n/2 \) entries of \( s \). Assume also an input for which \( u \) has rows \( A_{1,*}, A_{2,*}, \ldots, A_{n/2,*} \) and entries \( s(n/2+1), s(n/2+2), \ldots, s(n) \), and \( v \) has all the remaining rows of \( A \) and entries of \( s \).
We now show a reduction from the well-known equality problem (EQ), in which two parties are required to determine whether their input vectors \(x, y \in \{0,1\}^n\) are equal. Assuming we are given a procedure \(P\) for our vector by matrix multiplication problem, we use it to solve the EQ problem. Given input vectors \(x, y\) for the EQ problem (at \(u\) and \(v\) respectively), we create an input for the vector by matrix multiplication problem in the following way. Node \(u\) assigns \(A(i,i) = x(i) + 1\) for every \(i \in [1, \ldots, n/2]\) and \(s(i) = x(i) + 1\) for every \(i \in [n/2+1, \ldots, n]\), while node \(v\) assigns \(A(i,i) = y(i) + 1\) for every \(i \in [n/2+1, \ldots, n]\) and \(s(i) = y(i) + 1\) for every \(i \in [1, \ldots, n/2]\). All the other entries of \(A\) are initialized to 0, thus \(A\) is a diagonal matrix. It follows that \(s'(i) = \sum_{j=1}^{n} s(j)A(j,i) = s(i)A(i,i) = (x(i) + 1)(y(i) + 1)\) for every \(i\). Given the value of \(s'(i)\), one can decide whether \(x(i) = y(i)\), since clearly \(x(i) = y(i)\) if and only if \(s'(i) \in \{1,4\}\) (and otherwise \(s'(i) = 2\)). It follows that using procedure \(P\) one can solve the EQ problem, which is known to require at least \(n\) bits of communication. Therefore, assuming that each message is \(O(\log n)\) bits, our problem requires \(\Omega(n/\log n)\) communication rounds.

Necessity of \(A_E\): Consider the family of sun partitioned networks \(S_n\). As discussed earlier, Axiom \(A_E\) is violated while the other hold. The diameter of \(S_n\) is \(\Omega(n)\), hence any algorithm for this task will require \(\Omega(n)\) communication rounds (due to the lower bound discussed earlier).

Necessity of \(A_C\): Consider the family of lollipop partitioned networks \(L_n\). As discussed earlier, Axiom \(A_C\) is violated while the other hold. Again, the diameter of \(L_n\) is \(\Omega(n)\), hence any distributed matrix transpose algorithm requires \(\Omega(n)\) rounds.

\[\square\]

D Algorithms for additional problems

D.1 Matrix transposition

Initially, each node in \(V\) holds one row of an \(O(k)\)-sparse matrix \(A\) (along with its index). The task is to distributively calculate the matrix \(A^T\) and store its rows in such a way that the node that stores row \(A_{i,*}\) will eventually store row \(A_{i,*}^T\). We start with a claim on the lower bound.

Claim 9. The lower bound for any algorithm for transposing an \(O(k)\)-sparse matrix on any network is \(\Omega(D)\), and on a \(CP\)-network is \(\Omega(k)\).

Proof. Consider a nonzero entry \(A(i,j)\) where \(j \neq i\). Consider the nodes \(u\) and \(v\) that initially store \(A_{i,*}\) and \(A_{j,*}\) respectively. Clearly, in any algorithm, \(A(i,j)\) should be delivered to the node \(v\) (which is required to eventually obtain \(A_{*,j} = A_{j,*}^T\)). Since the distance \(d(u,v)\) may be as large as the diameter, the lower bound is \(\Omega(D)\) rounds.

For a \(CP\)-network, the lower bound is \(\Omega(k)\) since there are inputs for which row \(A_{j,*}^T\) has \(k\) nonzero values which must be delivered to the node that initially has row \(A_{i,*}\). There are \(CP\)-networks in which minimum degree is 1 (see Figure
(I) for an illustration) and hence, delivering \( \Omega(k) \) messages will require \( \Omega(k) \) communication rounds.

\[ \Omega(k) \]

Algorithm 2. The following algorithm generates \( A^T \) in \( O(k) \) rounds on a \( \mathcal{CP} \)-network \( G(V,E,C,P) \).

1. Each \( u \in V \) sends its row (all the nonzero values with their indices in \( A \)) to its representative \( r(u) \in C \). Now each representative has \( O(n_c) \) rows of \( A \) (or \( O(kn_c) \) entries of \( A \)).
2. Each representative, for each entry \( A(i,j) \) it has, sends it to the node in \( C \) that is responsible for the row \( A^T_{j,*} \). By agreement, every node in \( C \) is responsible for the rows of \( A^T \) with indices \( 1 + (n/n_c)(i-1), \ldots, (n/n_c)i \) (assuming \( n/n_c \) is integer).
3. Now each node in \( C \) stores \( O(n/n_c) \) rows of \( A^T \). So, each node \( u \in V \) that initially stored the row \( i \) of \( A \), requests \( A^T_{i,*} \) from its representative. The representative gets the row from the corresponding node in \( C \) and sends it back to \( u \).

Theorem 8. On a \( \mathcal{CP} \)-network \( G(V,E,C,P) \), the transpose of a \( O(k) \)-sparse matrix can be completed in \( O(k) \) rounds w.h.p.

Proof. Consider Algorithm 2 and the \( \mathcal{CP} \)-network \( G(V,E,C,P) \). Step 1 of the algorithm will take \( O(k) \) rounds since each row has up to \( k \) nonzero entries and sending one entry takes \( O(1) \) due to Axiom \( A_C \). Now each representative has \( O(kn_c) \) values since it represents up to \( O(n_c) \) nodes in \( P \) (due to Axiom \( A_B \)).

In the beginning of Step 2, each representative knows the destination for each of the \( A(i,j) \) entries it has (since, by agreement, each node in \( C \) is responsible for collecting entries for specific rows of \( A^T \)). So, it will send \( O(kn_c) \) messages, each one to a specific single destination. Since each node in \( C \) is responsible for \( O(n/n_c) \) rows of \( A^T \), it will receive \( O(kn/n_c) \) messages. Thus, using Axiom \( A_E \) and Theorem 6 we get \( O(k) \) running time.

At Step 3, a single row (with \( O(k) \) nonzero entries) is sent by each node to its representative (takes \( O(k) \) due to the Axiom \( A_C \)), then the requests are delivered to the appropriate nodes in \( C \) and the replies with the appropriate rows of \( A^T \) are received back by the representatives. All this takes \( O(k) \) rounds due to the Axiom \( A_E \) and Theorem 6. Then the rows of \( A^T \) are delivered to the nodes that have requested them. Due to the Axiom \( A_C \) this will also take \( O(k) \) rounds.

We now show the necessity of the Axioms \( A_B \), \( A_E \) and \( A_C \) for achieving \( O(k) \) running time.

Theorem 9. For each \( X \in \{ B, E, C \} \) there exist a family \( \mathcal{F}_X = \{ G_X(V,E,C,P)(n) \} \) of partitioned networks that do not satisfy Axiom \( A_X \) but satisfy the other two
axioms, and input matrices of size $n \times n$ for every $n$, such that the time complexity of any matrix transposition algorithm on the networks of $F_i$ with the corresponding inputs is $\Omega(n)$.

Proof. Consider the following cases where in each case one of the axioms is not satisfied while the other two satisfied.

Necessity of $A_B$: Consider the family of dumbbell partitioned networks $D_n$. As discussed earlier, Axiom $A_B$ is violated while the other hold. Let $A$ be a matrix where at least the following $n/2$ entries are nonzero (assume $n/2$ is even):

$$A(n/2+1,1), A(n/2+2,2), \ldots, A(n,n/2).$$

Then we input the rows $A_{1,*} - A_{n/2,*}$ to the nodes in the first star of $D_n$, and rows $A_{n/2+1,*} - A_{n,*}$ to the nodes in the second star of $D_n$. Clearly, the entries we specified before are initially located in the second star but they all must be delivered to the first star (by problem definition, an entry $A(i,j)$ should be eventually stored in a node that initially has row $A_{j,*}$). Since there is only one edge connecting the stars, any algorithm for the specified task will take $\Omega(n)$ rounds.

Necessity of $A_E$: Consider the family of sun partitioned networks $S_n$. As discussed earlier, Axiom $A_E$ is violated while the other hold. The diameter of $S_n$ is $\Omega(n)$, hence any algorithm for this task will require $\Omega(n)$ communication rounds (due to the lower bound discussed earlier).

Necessity of $A_C$: Consider the family of lollipop partitioned networks $L_n$. As discussed earlier, Axiom $A_C$ is violated while the other hold. Again, the diameter of $L_n$ is $\Omega(n)$, hence any distributed matrix transpose algorithm requires $\Omega(n)$ rounds.

□

D.2 Matrix multiplication

Let $A$ and $B$ be square $n \times n$ matrices with $O(k)$ nonzero entries in each row and each column. Initially, each node in $V$ holds one row of $A$ (along with the index of the row) and one row of $B$ (along with the index of the row). The task is to distributively calculate $C = AB$ and store its rows at the corresponding nodes in $V$, such that the node that initially stored row $i$ of $B$ will store row $i$ of $C$. We start with a claim on the lower bound.

Claim 10. The lower bound for any algorithm for $O(k)$-sparse matrix multiplication on any network is $\Omega(D)$, and on a $CP$-network is $\Omega(k^2)$.

Proof. Let us start with a lower bound for any network. In order to obtain $C(1,1)$, we need at least to multiply $A(1,1)$ by $B(1,1)$ (assuming input in which $A(1,1) \neq 0$ and $B(1,1) \neq 0$). But $A(1,1)$ and $B(1,1)$ can be located at different nodes $u$ and $v$ with distance $d(u,v) = D$. Thus, it will take at least $\Omega(D)$ rounds to bring $A(1,1)$ and $B(1,1)$ together.

For a $CP$-network, consider a network illustrated on Figure 1 (I), where degree of a node $u \in P$ is 1. Assume that initially $u$ has row $A_{1,*}$ and $B_{1,*}$ and thus, by problem definition, it has to eventually receive the row $C_{1,*}$. We show
now that there are $\Omega(k^2)$ messages are required to allow $u$ to obtain $C_{1,s}$. Assume that \{\text{b}\}_i^j for $i, j \in [1, \ldots, k]$ are $k^2$ distinct values. Consider an input in which $A(1, i) = 1$ for every $i \in [2, \ldots, k + 1]$, $B(2, i) = b_1^1$ for every $i \in [1, \ldots, k]$, $B(3, i) = b_2^2$ for every $i \in [k+1, \ldots, 2k]$, and so on until $B(k+1, i) = b_k^k$ for every $i \in [k(k-1) + 1, \ldots, k^2]$. All other entries of $A$ and $B$ are set to 0. Easy to see that $C(1, i) = b_1^i$ for every $i \in [1, \ldots, k]$, $C(1, i) = b_2^i$ for every $i \in [k+1, \ldots, 2k]$, and so on until $C(k, i) = b_k^i$ for every $i \in [k(k-1) + 1, \ldots, k^2]$. So, in order to obtain the row $C_{1,s}$, $u$ must receive all the $k^2$ values $\{b_i^j\}$ which will take $\Omega(k^2)$ communication rounds.

\textbf{Algorithm 3.} The following algorithm solves the $O(k)$-sparse matrix multiplication task in $O(k)$ rounds on a $CP$-network $G(V, E, C, \mathcal{P})$.

1. Each node in $V$ sends its row of $B$ to its representative. Now, each node in $C$ has $O(kn_c)$ entries of $B$.
2. Now, nodes in $C$ redistribute the rows of $B$ among themselves in a way that the node with ID $i$ will store rows $1 + (n/n_c)(i - 1), \ldots, (n/n_c)i$ (assuming $n/n_c$ is integer). Now, each node $u \in C$ has $O(n/n_c)$ rows of $B$.
3. Each node in $V$ sends its row of $A$ to its representative. Notice that the row $i$ of $B$ needed to be multiplied only by values of the column $i$ of $A$.
4. Each node $u \in C$ sends the values of $A$ it has to the nodes in $C$ that hold the corresponding rows of $B$. I.e., the value $A(i, j)$ will be sent to the node in $C$ which holds the row $B_j,i$. Now we have all the summands prepared and distributed across all the nodes in $C$. It is now left to combine corresponding summands.
5. Each node $u \in C$ sends each of its values to the corresponding node in $C$ that is responsible for gathering the summands for the values of specific $O(n/n_c)$ rows of the resulting matrix $C$.
6. Now each node $u \in V$ that initially stored row $i$ of $B$, requests row $i$ of $C$ from its representative. The representative gets the row from the corresponding node in $C$ and sends it back to $u$. Easy to see that this step takes $O(k^2)$ rounds.

\textbf{Theorem 10.} On a $CP$-network $G(V, E, C, \mathcal{P})$, the multiplication of two $O(k)$-sparse matrices can be completed in $O(k)$ rounds w.h.p.

\textbf{Proof.} Consider Algorithm 3 and the network $G(E, V)$ with partition $\langle C, \mathcal{P} \rangle$ that satisfies Axioms $A_B, A_E, A_C$.

Step 1 will take $O(k)$ due to the Axiom $A_C$ and the fact that the number of nonzero entries in each row is bounded by $k$. Each node in $C$ will have $O(kn_c)$ entries of $B$ since it represents $O(n_c)$ nodes in $\mathcal{P}$ due to the Axiom $A_B$. Using Theorem 7 with the parameters: $M_s = O(kn_c)$ and $M_r = O(kn/n_c)$, we show that the redistribution performed at Step 2 takes $O((kn_c + k/n_c)n_c)/O(k)$ rounds.

For Step 4, we again use Theorem 7 with the parameters: $M_s = O(k)O(n_c)$ ($O(k)$ values per row), $M_r = O(n/n_c)O(k)$ ($O(k)$ values per column of $A$), and
obtain that the running time is $O(k)$. Note that, each node in $C$ has $O(n/n_c)$ rows (of $B$) with $O(k)$ elements in each. Each row was multiplied by $O(k)$ values received in the previous step ($O(k)$ since each column of $A$ has $O(k)$ nonzero values). Thus, each $u \in C$ has $O(k^2n/n_c)$ values that needed to be sent to the appropriate nodes in the next step.

At Step 5, each $u \in C$ sends each of its values to the corresponding node in $C$ that is responsible for gathering the summands for the values of specific $O(n/n_c)$ rows of the resulting matrix $C$. Clearly, $M_s = O(k^2n/n_c)$ since each row of $B$ has $O(k)$ different entries, and was multiplied by $O(k)$ different entries of $A$. Now let’s find $M_r$. Each $u \in C$ is responsible for $O(n/n_c)$ rows of $C$. Thus, e.g., for a row $C_{1,*}$ it needs to receive the following summands: $C(1,1) = \sum A(1,i)B(i,1)$, $C(1,2) = \sum A(1,i)B(i,2)$, ..., $C(1,n) = \sum A(1,i)B(i,n)$. Since the number of nonzero entries in each row of $A$ and $B$ is $O(k)$, the number of nonzero entries in each row of $C$ is bounded by $O(k^2)$. Thus, for each row of $C$, a node in $C$ will receive $O(k^2)$ messages. So, $M_r = O(k^2n/n_c)$ and thus the running time of this step is $O(k^2)$.

At the last step, each node $u \in V$ sends a single message (request for a row) to its representative. This takes $O(1)$ due to the convergecast. Then, the representative gets the row from the corresponding node in $C$ and sends it back to $u$. Using Axiom $A_E$ and Theorem $\square$ with $M_s = O(n_c)$ and $M_r = O(n/n_c)$ we get $O(1)$ running time for delivering the request inside the core. In a similar way, sending the rows inside the core will take $O(k^2)$ rounds. The same amount of time will be required to deliver those rows to the nodes in $P$ that requested them ($O(1)$ per row entry due to convergecast, and $O(k^2)$ nonzero entries per row).

\begin{theorem}
\begin{proof}
For each $X \in \{B,E,C\}$ there exist a family $\mathcal{F}_X = \{G_X(V,E,C,P)(n)\}$ of partitioned networks that do not satisfy Axiom $A_X$ but satisfy the other two axioms, and input matrices of size $n \times n$, for every $n$, such that the time complexity of any algorithm for the multiplication of two $O(k)$-sparse matrices on the networks of $\mathcal{F}_X$ with the corresponding inputs is $\Omega(n/\log n)$.
\end{proof}
\end{theorem}

\begin{proof}
Necessity of $A_B$: Consider the family of dumbbell partitioned networks $D_n$. As discussed earlier, Axiom $A_B$ is violated while the other hold. As we did in the proof of Theorem $\square$ we can show a reduction of from the well equality problem (EQ) to our problem of multiplication of two sparse matrices. Here we initialize $A$ and $B$ to be diagonal matrices and the first core node, $u$, will have first half of $A$’s rows and second half of $B$’s rows. Due to the initialization, each entry of the resulting matrix $C$ will be a multiplication of the corresponding entries of $x$ and $y$. Thus, obtaining $C$ will allow us to determine whether $x$ and $y$ are equal. Thus, the lower bound for our problem is $\Omega(n/\log n)$ communication rounds.

The proof of the necessity of $A_E$ and $A_C$ is the same as in the proof of Theorem $\square$ and is based on the diameter argument.
\end{proof}
Now we show another set of algorithms that deal with aggregate functions. In some of the following algorithms, we use the following result on distributed sorting in clique, presented in [3].

**Theorem 12** ([3]). Consider a clique network $G(V, E)$ where nodes in $V$ have IDs $[1, \ldots, n]$. Each node is given $n$ values. For simplicity, assume that all the $n^2$ values are distinct. The following tasks can be solved deterministically in $\Theta(1)$ rounds.

1. Node $i$ needs to learn the values with indices $i(n - 1) + 1, \ldots$ in the total order of all values.
2. Node $i$ needs to determine the indices of its input (initial) values in the total order of all values.

**Observation 1.** Theorem 12 can be naturally extended to the case where each node initially holds $O(n)$ keys (instead of exactly $n$).

### D.3 Find my rank

Let each node $v \in V$ hold some initial value. Each node needs to know the position of its value in the ordered list of all the values, i.e., the rank of its value. We start with a claim on the lower bound.

**Claim 11.** The lower bound for any algorithm for finding the rank of each value on any network is $\Omega(D)$, and on a CP-network is $\Omega(1)$.

**Proof.** Consider the two nodes $u, v \in V$. In order to decide whose value is larger, at least one bit of information must travel between them. Thus, the lower bound for this task on any graph is $\Omega(D)$. \qed

**Algorithm 4.** The following algorithm finds the rank of each value in $O(1)$ rounds on a CP-network $G(V, E, C, P)$.

1. Each node in $V$ sends its initial value to its representative in $C$.
2. Nodes in $C$ perform sorting of all the values they have, and obtain the ranks of those values. So, each node $u \in C$ will know the ranks of the values it had received from the nodes it represents.
3. The ranks are delivered to the appropriate nodes in $P$.

**Theorem 13.** On a CP-network $G(V, E, C, P)$, the task of obtaining a rank can be completed in $O(1)$ rounds w.h.p.

---

4 This limitation can be eliminated by chaining each value with the node ID and its order at the node. Thus, each input value becomes unique.
Proof. Consider Algorithm 4 and the CP-network \( G(V, E, C, P) \). The first step takes \( O(1) \) due to Axiom \( \mathcal{A}_C \). Now, each representative \( u \in C \) has \( O(n_c) \) values due to the Axiom \( \mathcal{A}_B \). Step 2, is performed in \( O(1) \) rounds due to the Theorem 12 and Axiom \( \mathcal{A}_E \). The last step will take \( O(1) \) rounds due to \( \mathcal{A}_C \). \( \square \)

**Theorem 14.** For each \( X \in \{B, E, C\} \) there exist a family \( \mathcal{F}_X = \{G_X(V, E, C, P)(n)\} \) of partitioned networks that do not satisfy Axiom \( \mathcal{A}_X \) but satisfy the other two axioms, and input matrices of size \( n \times n \) and vectors of size \( n \), for every \( n \), such that the time complexity of any algorithm for finding the rank of each value on the networks of \( \mathcal{F}_X \) with the corresponding inputs is \( \Omega(n) \).

**Proof.** Necessity of \( \mathcal{A}_B \): Consider the family of dumbbell partitioned networks \( D_n \). As discussed earlier, Axiom \( \mathcal{A}_B \) is violated while the other hold. Consider a sorted list of \( n \) values \( (a_i \leq a_{i+1}) : a_1, a_2, a_3, ... , a_n \). Clearly, every two adjacent values must be compared in any sorting procedure. For example, if we assume that some sorting procedure does not compare \( a_2 \) and \( a_3 \), and also \( a_4 = a_3 + 2 \) then we can replace \( a_2 \) with \( a_3 + 1 \) and get the same output \( a_1, a_2, a_3, ... , a_n \) which is now incorrect since \( a_2 > a_3 \).

Let us denote the two core’s nodes as \( u \) and \( v \). Node \( u \) and all the nodes in \( P \) that are connected to it, will be assigned values (one value to one node) with odd indices in the ordered list (i.e., \( a_1, a_3, ... \)). The other \( n/2 \) values will be assigned to the remaining \( n/2 \) nodes (one value to one node).

Now, in order to obtain a sorted list, at least the following comparisons must take place: \( a_1 \) with \( a_2 \), \( a_3 \) with \( a_4 \), and so on. We can see that about \( n/2 \) pairs of values have to be brought together while initially they are located at the different sides of the link \((u, v)\). Thus, \( \Omega(n) \) messages must be sent over the link \((u, v)\) in order to perform the sorting task. The last takes \( \Omega(n) \) communication rounds.

The proof of the necessity of \( \mathcal{A}_E \) and \( \mathcal{A}_C \) is the same as in the proof of Theorem 9 and is based on the diameter argument. \( \square \)

### D.4 Find median

Let each node \( v \in V \) hold some initial value. Each node needs to know the value which is the median in the ordered list of all the values in the network. We start with a claim on the lower bound.

**Claim 12.** The lower bound for any algorithm for finding median on any network is \( \Omega(D) \), and on a CP-network is \( \Omega(1) \).

**Proof.** Consider two nodes \( u, v \in V \). It is obvious that if \( u \) wants to obtain the median of the initial values, at least one bit of information must travel between \( u \) and \( v \) (otherwise \( u \) will never know that \( v \) exists while \( v \) may even be the median). Since the distance \( d(u, v) \) can be as large as the diameter, the lower bound for this task is \( \Omega(D) \) communication rounds. \( \square \)
Algorithm 5. The following algorithm finds the median in $O(1)$ rounds on a $CP$-network $G(V,E,C,P)$.

1. Each node in $V$ sends its initial value to its representative in $C$.
2. Nodes in $C$ perform sorting of all the values they have, and obtain the ranks of those values. So, each node $i \in C$ now knows values with indices $i(n_c - 1) + 1, \ldots, in_c$ according to the total order of all values.
3. The node in $C$ which has the value with index $n^2/2$ (which is the median value) sends it to all the nodes in $C$.
4. The median value is delivered to the nodes in $P$.

Theorem 15. On a $CP$-network $G(V,E,C,P)$, the finding median of all the initial values can be completed in $O(1)$ rounds w.h.p.

Proof. Consider Algorithm 5 and the $CP$-network $G(V,E,C,P)$. The first step takes $O(1)$ due to Axiom $A_C$. Now, each representative $u \in C$ has $O(n_c)$ values due to Axiom $A_B$. Step 2, is performed in $O(1)$ rounds due to Theorem 12 and Axiom $A_E$. The last step will take $O(1)$ rounds due to Axiom $A_C$. $\square$

Theorem 16. For each $X \in \{B,E,C\}$ there exist a family $F_X = \{G_X(V,E,C,P)(n)\}$ of partitioned networks that do not satisfy Axiom $A_X$ but satisfy the other two axioms, and input matrices of size $n \times n$ and vectors of size $n$, for every $n$, such that the time complexity of any algorithm for finding median of all the initial values on the networks of $F_X$ with the corresponding inputs is $\Omega(\log n)$.

Proof. Necessity of $A_B$: Consider the family of dumbbell partitioned networks $D_n$. As discussed earlier, Axiom $A_B$ is violated while the other hold. In constant time, each of the two centers $A$ and $B$ can learn the inputs of its star. Now the problem becomes for $A$ and $B$ to learn the median of the union of their sets. This operation is known to require at least $O(\log n)$ communication rounds. More formally, it is shown in [20] that the Median function does not admit a deterministic protocol using $O(\log^{1-\epsilon} n)$ rounds and a logarithmic amount of communication at each round, for any $\epsilon > 0$ (even though the total communication complexity of the problem is known to be only $O(\log n)$ bits). This allows us to conclude the same lower bound in our case too.

The proof of the necessity of $A_E$ and $A_C$ is the same as in the proof of Theorem 9 and is based on the diameter argument. $\square$

D.5 Find mode

Let each node $v \in V$ hold some initial value. Each node needs to know the value (values) that appears most frequently. We start with a claim on the lower bound.

Claim 13. The lower bound for any algorithm for finding mode on any network is $\Omega(D)$, and on a $CP$-network is $\Omega(1)$.
Proof. Consider two nodes \( u, v, w \in V \). Assume an input for which the most frequent value appears with frequency 2 and is initially located at \( v \) and \( w \), while all the other nodes have other distinct values. Obviously, if \( u \) needs to learn the mode, at least one bit of information must travel from \( v \) to \( u \), since otherwise \( u \) will not be aware of the \( v \)'s existence. Since the distance \( d(u,v) \) can be as large as diameter, the lower bound for this task is \( \Omega(D) \) communication rounds.

Algorithm 6. The following algorithm finds the mode in \( O(1) \) rounds on a \( CP \)-network \( G(V,E,C,P) \).

1. Each node in \( V \) sends its initial value to its representative in \( C \).
2. Nodes in \( C \) perform sorting of all the values they have, and obtain the ranks of those values. So, each node \( i \in C \) now knows values with indices \( i(n_C - 1) + 1, \ldots i n_C \) according to the total order of all values.
3. Each node in \( C \) sends to each other node in \( C \):
   (a) its most frequent value (values) and its frequency,
   (b) its minimum value and its frequency,
   (c) its maximum value and its frequency.
   The minimum and maximum are needed in order capture the most frequent values that appear at the boundaries of the ranges.
4. Now each node in \( C \) can find the most frequent value (values) and delivers it to the nodes in \( P \) it represents.

Theorem 17. On a \( CP \)-network \( G(V,E,C,P) \), the task of finding mode (the most frequent value) can be completed in \( O(k) \) rounds w.h.p.

Proof. Consider Algorithm 6 and the \( CP \)-network \( G(V,E,C,P) \). The first step takes \( O(1) \) due to Axiom \( AC \). Now, each representative \( u \in C \) has \( O(n_C) \) values due to Axiom \( AB \). Step 2, is performed in \( O(1) \) rounds due to Theorem 12 and Axiom \( AE \). Step 3 takes \( O(1) \) due to Axiom \( AE \) since each node in \( C \) needs to send \( O(1) \) values to all the other nodes in \( C \). The last step will take \( O(1) \) rounds due to \( AC \) (and assuming there are \( O(1) \) most frequent values).

Theorem 18. For each \( X \in \{B, E, C\} \) there exist a family \( \mathcal{F}_X = \{G_X(V,E,C,P)(n)\} \) of partitioned networks that do not satisfy Axiom \( AX \) but satisfy the other two axioms, and input matrices of size \( n \times n \) and vectors of size \( n \), for every \( n \), such that the time complexity of any algorithm for the finding mode on the networks of \( \mathcal{F}_X \) with the corresponding inputs is \( \Omega(n/\log n) \).

Proof. Necessity of \( AB \): Consider the family of dumbbell partitioned networks \( D_n \). As discussed earlier, Axiom \( AB \) is violated while the other hold. Assume such an input that every element appears exactly once or twice on each side (for simplicity assume there are \( n/4 \) types of elements altogether, and some nodes do not have any element). Hence the most frequent element will appear 2, 3 or 4 times in the graph. The case where the answer is 2 occurs only when
every element appears exactly once on every side. This case is easy to check in a constant amount of communication between the two centers, so we assume we do this check first, and it remains to consider the case where this does not happen. It thus remains to decide whether the answer is 3 or 4. To do that, A (the center of the first star) defines a set $S_A$ of all the elements that appear twice in its star, and B defines a set $S_B$ similarly for its side. Now the answer is 4 if the sets $S_A$ and $S_B$ intersect. Set disjointness has communication complexity $n$, so A and B must exchange at least $n$ bits, or, at least $\Omega(n/\log n)$ messages. Since these messages all cross the single edge connecting the two centers, they require this much time.

The proof of the necessity of $A_{E}$ and $A_{C}$ is the same as in the proof of Theorem 9 and is based on the diameter argument. 

D.6 Find the number of distinct values

Let each node $v \in V$ hold some initial value. Each node needs to know the number of distinct values present in the network. We start with a claim on the lower bound.

**Claim 14.** The lower bound for any algorithm for finding the number of distinct values on any network is $\Omega(D)$, and on a CP-network is $\Omega(1)$.

**Proof.** Consider two nodes $u,v \in V$. Assume an input for which all the values in the network are distinct. Obviously, that if $u$ needs to learn the number of distinct values, at least one bit of information must travel from $v$ to $u$, since otherwise $u$ will not be aware of the $v$’s existence. Since the distance $d(u,v)$ can be as large as diameter, the lower bound for this task is $\Omega(D)$ communication rounds. 

**Algorithm 7.** The following algorithm finds the number of distinct values in $O(1)$ rounds on a CP-network $G(V,E,C,P)$.

1. Each node in $V$ sends its initial value to its representative in $C$.
2. Nodes in $C$ perform sorting of all the values they have, and obtain the ranks of those values. So, each node $i \in C$ now knows values with indices $i(n_c - 1) + 1, \ldots, in_c$ according to the total order of all values.
3. Each node in $C$ sends to every other node in $C$ the number of distinct values and the two border values (min,max). Then, every node in $C$ will be able to find the number of distinct values (for each repeated border value, decrease 1 from the total count).
4. Each representative delivers the number of distinct nodes to the nodes in $P$ it represents.

**Theorem 19.** On a CP-network $G(V,E,C,P)$, the task of finding the number of distinct values can be completed in $O(1)$ rounds w.h.p.
Theorem 12. For each $X \in \{B,E,C\}$ there exist a family $\mathcal{F}_X = \{G_X(V,E,C,P)(n)\}$ of partitioned networks that do not satisfy Axiom $A_X$ but satisfy the other two axioms, and input matrices of size $n \times n$ and vectors of size $n$, for every $n$, such that the time complexity of any algorithm for finding the number of distinct values on the networks of $\mathcal{F}_X$ with the corresponding inputs is $\Omega(n/\log n)$.

Proof. Necessity of $A_B$: Consider the family of dumbbell partitioned networks $D_n$. As discussed earlier, Axiom $A_B$ is violated while the other hold. Assume that the inputs are taken out of a range of $m$ distinct possible values. In constant time, the two star centers A and B can collect $m$-bit vectors $x$ and $y$ respectively, representing the values that exist in their respective stars. The goal is for A and B to decide the number of distinct values in the graph, i.e., the number of 1’s in the vector $x \lor y$. We show a reduction from set disjointness to this problem, hence the lower bound for set disjointness holds for it. Assume we are given a procedure $P$ for our problem. We use it to solve set disjointness as follows. (1) A computes $|x|$ and informs B. (2) B computes $|y|$ and informs A. (3) A and B invoke procedure $P$ and compute $|x \lor y|$. (4) The answer is “yes” (the sets are disjoint) iff $|x \lor y| = |x| + |y|$. It is easy to verify that the reduction is correct, hence we get the desired lower bound of $\Omega(m/\log n)$ on the number of round required for finding the number of distinct values.

The proof of the necessity of $A_E$ and $A_C$ is the same as in the proof of Theorem 9 and is based on the diameter argument. \qed

D.7 Get the top $k$ ranked by areas

Let each node $v \in V$ hold some initial value. Assume that each value is assigned to a specific area of a total $\sqrt{n}$ areas. E.g., values may represent news and areas topics, so that each news belongs to a specific topic. Assume also that each node in $V$ is interested in one specific area. The task is to deliver to each node in $V$ the largest $k$ values from the area it is interested in. We start with a claim on the lower bound.

Claim 15. The lower bound for any algorithm for finding the $k$ top ranked values from a specific area on any network is $\Omega(D)$, and on a CP-network is $\Omega(k)$.

Proof. First, let us show the lower bound for any network. Consider two nodes $u, v \in V$ and assume input in which $u$ is interested in the value initially stored
at v. Obviously, delivering the value from v to u will take at least d(u, v). Since the distance d(u, v) may be as large as diameter, the lower bound on the running time is Ω(D).

For a CP-network, the lower bound is Ω(k) since obviously, there are inputs for which k values must be delivered to a node in P. There are CP-networks in which minimum degree is 1 (see Figure 1 (I) for an illustration) and hence, delivering Ω(k) messages will require Ω(k) communication rounds. □

Algorithm 8. The following algorithm finds the k top ranked values from a specific area in O(k) rounds on a CP-network G(V, E, C, P).

Without loss of generality, assume that all the values are from the range [1, ..., n], and each area has its own range for its values (e.g., politics [1, ..., 100], sports [101, ..., 200], etc.).

1. Each node in V sends its initial value to its representative in C.
2. Perform sorting using Theorem 12 and Axiom A_E. It takes O(1). Each node i ∈ C now knows values with indices i(n_c - 1) + 1, ..., n_c according to the total order of all values.
3. Now, each node in C will send the largest k values from each area to the appropriate node in C, so that each node in C will be responsible for at most one area (recall that we have √n areas and n_c = Ω(√n)).
4. Each representative sends requests for areas (up to O(n_c) areas) requested by nodes it represents. Each request is destined to a specific node in C that is responsible for the requested area. Upon request, each node in C returns the k largest value for the area it is responsible for to the nodes that requested them.
5. Each representative u ∈ C delivers the values to the nodes in P it represents.

Theorem 21. On a CP-network G(V, E, C, P), the task of finding the k top ranked values from a specific area can be completed in O(k) rounds w.h.p.

Proof. Consider Algorithm 8 and the CP-network G(V, E, C, P). From Theorem 2, we know that n_c = Ω(√n), thus we can say that the number of areas is O(n_c). The first step takes O(1) due to Axiom A_C. Now, each representative u ∈ C has O(n_c) values due to the Axiom A_E. At Step 2, each node has O(n_c) values (each destined to a specific single node), so M_s = n_c. Each node has to receive M_r = k values (more precisely: M_r = 2k, since it is possible that after the initial sorting, an area is split across two nodes, and each of these two nodes will send up to k values from that area and the receiving node will have to select the correct k). Thus, this step will take (according to Theorem 5) O((n_c + k)/n_c) = O(k/n_c).

Step 3 takes O(k) since sending requests will take O(1) (due to the Axiom A_E and Theorem 5 with M_s = n_c, M_r = n_c), and receiving the desired values will take O(k) (since M_s = kn_c and M_r = kn_c). The last step will take O(k) since each node in P needs k values, and delivering a single value from r(u) ∈ C to u ∈ P takes O(1) due to the Axiom A_C. □
Theorem 22. For each $X \in \{B, E, C\}$ there exist a family $\mathcal{F}_X = \{G_X(V,E,C,P)(n)\}$ of partitioned networks that do not satisfy Axiom $A_X$ but satisfy the other two axioms, and input matrices of size $n \times n$ and vectors of size $n$, for every $n$, such that the time complexity of any algorithm for finding the $k$ top ranked values from a specific area on the networks of $\mathcal{F}_X$ with the corresponding inputs is $\Omega(kn_c)$.

Proof. Necessity of $A_B$: Consider the family of dumbbell partitioned networks $D_n$. As discussed earlier, Axiom $A_B$ is violated while the other hold. Consider $\sqrt{n}/2$ areas and assume that all the $k\sqrt{n}/2$ values belonging to these areas are initially located at the nodes of the first star of $D_n$. Consider a subset of nodes of the second star. Let the subset size be $\sqrt{n}/2$ and each node in this subset is interested in different area of the areas stored in the first star we mentioned. We can see that in order to complete the task, all the $k\sqrt{n}/2$ values have to be sent from the first to the second star, which are interconnected by a single edge. Thus, the running time will be $\Omega(k\sqrt{n})$ rounds.

The proof of the necessity of $A_E$ and $A_C$ is the same as in the proof of Theorem 9 and is based on the diameter argument. □