Sparsifying Congested Cliques and Core-Periphery Networks

Alkida Balliu[1,2], Pierre Fraigniaud[1⋆], Zvi Lotker[3⋆⋆], and Dennis Olivetti[1,2]

[1] CNRS and University Paris Diderot, France.
[2] Gran Sasso Science Institute, L'Aquila, Italy.
[3] Ben Gurion University, Israel

Abstract. The *core-periphery* network architecture proposed by Avin et al. [ICALP 2014] was shown to support fast computation for many distributed algorithms, while being much sparser than the *congested clique*. For being efficient, the core-periphery architecture is however bounded to satisfy three axioms, among which is the capability of the core to emulate the clique, i.e., to implement the all-to-all communication pattern, in $O(1)$ rounds in the CONGEST model. In this paper, we show that implementing all-to-all communication in k rounds can be done in n-node networks with roughly n^2/k edges, and this bound is tight. Hence, sparsifying the core beyond just saving a fraction of the edges requires to relax the constraint on the time to simulate the congested clique. We show that, for $p \gg \sqrt{\log n/n}$, a random graph in $\mathcal{G}_{n,p}$ can, w.h.p., perform the all-to-all communication pattern in $O(\min\{\frac{1}{p^2}, np\})$ rounds. Finally, we show that if the core can emulate the congested clique in t rounds, then there exists a distributed MST construction algorithm performing in $O(t \log n)$ rounds. Hence, for $t = O(1)$, our (deterministic) algorithm improves the best known (randomized) algorithm for constructing MST in core-periphery networks by a factor $\Theta(\log n)$.

1 Introduction

1.1 Context and Objectives

Inspired by social networks and complex systems, Avin, Borokhovicha, Lotker, and Peleg [1] proposed a novel network architecture for parallel and distributed computing, called *core-periphery*. Interestingly, the core-periphery architecture is not described explicitly (e.g., via the description of a specific graph family), but rather implicitly via three so-called *axioms*. Specifically, a core-periphery network $G = (V, E)$ has its node set partitioned into a *core* C and a *periphery* P, and the three properties to be satisfied are then the following:

1. **Core boundary:** For every node $v \in C$, $\deg_C(v) \simeq \deg_P(v)$, where, for $S \subseteq V$ and $v \in V$, $\deg_S(v)$ denotes the number of neighbors of v in S.

⋆ Additional supports from ANR project DISPLEXITY, and Inria project GANG.
⋆⋆ Additional supports from Fondation des Sciences Mathématiques de Paris (FSMP).

2. **Clique emulation:** the core can emulate the clique in a constant number of rounds in the CONGEST model. That is, there is a communication protocol running in a constant number of rounds in the CONGEST model such that, assuming that each node $v \in C$ has a message $M_{v,w}$ on $O(\log n)$ bits for every $w \in C$, then, after $O(1)$ rounds, every $w \in C$ has received all messages $M_{v,w}$, for all $v \in C$. In other words, the *all-to-all* communication pattern can be implemented in a constant number of rounds.
3. **Periphery-core convergecast:** there is a communication protocol running in a constant number of rounds in the CONGEST model such that, assuming that each node $v \in P$ has a message M_v on $O(\log n)$ bits, then, after $O(1)$ rounds, for every $v \in P$, at least one node in the core has received M_v.

Figure 1 provides an example of a core-periphery network, i.e., a graph satisfying the three axioms. It was proved in [1] that these three axioms alone enable to design efficient distributed algorithms in the CONGEST model for classical problems such as matrix multiplication and MST construction. Most of the proposed algorithms are optimal in a sense that there is an asymptotically matching lower bound on the number of rounds under the three axiomatic constraints. Moreover, it is shown that if only two out of three axioms were satisfied, then the round complexity of all the considered problems would increase quite significantly — typically, from $O(1)$ to $O(\text{poly}(n))$ in n-node networks. There was an exception though: while the best known lower bound in [1] for MST construction is $\Omega(1)$, the proposed (randomized) MST construction algorithm runs in $O(\log^2 n)$ rounds. (If only two out of three axioms were satisfied, then MST construction would require at least $\tilde{\Omega}(n^{\frac{1}{4}})$ rounds).

The core-periphery model provides an attractive alternative to the *congested clique* model [19]. Indeed, the latter assumes a complete network interconnecting the nodes, i.e., for every two (distinct) nodes u and v, there is an edge $\{u, v\}$ connecting these nodes. The n-node congested clique has therefore $\binom{n}{2}$ edges, and every node has degree $n - 1$. Instead, assuming a core with, e.g., $O(\sqrt{n})$ nodes, even connecting all nodes in the core as a clique would only result in $O(n)$ edges in the core, a number that is much more manageable in practice. On the other hand, it was proved in [1] that $\Omega(\sqrt{n})$ nodes is the limit of how small can be the core, and that the core C must be dense, with $\Theta(|C|^2)$ edges.

In this paper, our objective is twofold. First, we are aiming at establishing tradeoffs between the number of edges, and the capability of emulating the clique. More precisely, we consider the all-to-all communication pattern:

- **Input:** every node v has a message $M_{v,w}$, for every node $w \neq v$;
- **Output:** every node w has received the message $M_{v,w}$, for every node $v \neq w$.

In the CONGEST model, assuming all messages are on $O(\log n)$ bits, all-to-all can be performed in just a single round in the clique. Our first objective is to study the tradeoff between number of edges, and number of rounds for performing all-to-all in the CONGEST model.

Our second objective is to revisit one of the main problems left open in [1], namely the complexity of MST construction in the core-periphery model.

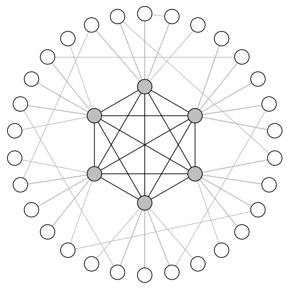

Fig. 1. *Example of a core-periphery network, where the core (gray nodes) is a clique, and the periphery (white nodes) is a sparse graph.*

1.2 Our results

We show that, in the CONGEST model, implementing all-to-all communication in k rounds can be done in n-node networks with roughly n^2/k edges, and this bound is essentially tight because every node must have degree at least $(n-1)/k$ to receive $n-1$ messages in at most k rounds. Hence, sparsifying the clique beyond just saving a fraction of the edges requires to relax the constraint on the time to simulate that clique.

Our first main result is about the ability of random graphs to emulate the clique. Let $\alpha = \sqrt{3e/(e-2)}$ where e is the basis of the natural logarithm. We show that, for $p \geq \alpha\sqrt{\ln n/n}$, a random graph in $\mathcal{G}_{n,p}$ can, w.h.p., perform all-to-all in $O(\min\{\frac{1}{p^2}, np\})$ rounds.

Our second main result is the design of a fast deterministic MST construction algorithm for core-periphery networks under the CONGEST model. Specifically, we show that if the core can emulate the clique in t rounds, then there exists a distributed MST construction algorithm performing in $O(t \log n)$ rounds. Hence, for $t = O(1)$, our deterministic algorithm performs in $O(\log n)$ rounds, improving the randomized algorithm in [1] by a factor $\Theta(\log n)$.

1.3 Related work

The congested clique model has been widely studied in the literature. Lenzen [18] investigated the routing and sorting problems in the context of congested clique. He showed a deterministic algorithm that, if each node is the sender and receiver of at most n messages, allows to route all the messages in $O(1)$ rounds in a clique of size n using messages of size $O(\log n)$. He also showed an algorithm that allows to sort n^2 keys in constant time. Drucker et al. [5] proved that the congested clique is powerful enough to emulate certain classes of bounded depth circuits, which shows how difficult finding lower bounds for the congested clique is. In the case where each node can only broadcast, [5] gives upper and lower bounds for the problem of detecting some types of subgraphs. Hegeman et al. [15] investigated the metric facility location problem providing a $O(1)$

approximation algorithm that runs in expected $O(\log \log \log n)$ rounds. They also showed how to compute a 3-ruling set in the congested clique. In [14] it is shown that, under some restrictions, fast algorithms for the congested clique model can be translated into fast algorithms in the MapReduce framework. Censor-Hillel et al. [3] showed that matrix multiplication on congested clique can be computed in $O(n^{1-2/\omega})$ rounds, where $\omega < 2.3728639$ is the exponent of matrix multiplication. Also, they showed how to use matrix multiplication to solve a variety of graph related problems. In [19], Lotker et al. provided a deterministic MST construction algorithm that runs in $O(\log \log n)$ rounds in the congested clique. Then, Hegeman et al. [13] showed that in this context randomization can help, giving a randomized algorithm that requires $O(\log \log \log n)$ rounds. Recently, this complexity was even reduced further to $O(\log^* n)$ in [12].

In general, the MST construction problem has been widely studied. In the distributed asynchronous context, the most famous algorithm is the one of Gallager, Humblet and Spira [10] that runs in $O(n \log n)$. In the synchronous setting, the first sublinear algorithm was given by Garay et al. in [11] that runs in $O(D + n^{\frac{\ln 3}{\ln 6}} \log^* n)$, where D is the diameter of the graph. This complexity was later improved to $O(D + \sqrt{n} \log^* n)$ in [16]. Then, Peleg et al. [23] showed that this complexity is near optimal, giving a $\Omega(\frac{\sqrt{n}}{\log n})$ lower bound. This bound was later improved by Sarma et al. [24] to $\Omega(\sqrt{\frac{n}{\log n}})$ and then by Ookawa et al. [22] to $\Omega(\sqrt{n})$. All these lower bounds hold for graphs with diameter $\Omega(\log n)$. For constant diameter graphs, there is a bound $\widetilde{\Omega}(n^{1/3})$ rounds for diameter 4, a bound $\widetilde{\Omega}(n^{1/4})$ rounds for diameter 3, and a bound $O(\log n)$ rounds for diameter 2 (see [20]). Finally, Elkin [6] showed that if termination detection is not required, the diameter of the graph is not a lower bound, and that there exists an algorithm that requires $\widetilde{O}(\mu + \sqrt{n})$ rounds, where μ is the so-called MST-radius of the graph.

Feige et al. [7] studied the broadcast problem in random graphs, where a single node has a message that has to be received by all the nodes of the graph. They show that rumor spreading (which propagates the message to a randomly chosen neighbor at each step) is an efficient way to solve the broadcast problem for random graphs. Censor-Hillel et al. [4] studied the broadcast problem in the context where every node is the source of a message and it is limited to send the *same* message to each neighbor at each round. They give an efficient algorithm that solves the problem, also in case of failures.

Finally, it is worth mentioning that a problem related to our results, that is finding disjoint paths between pairs of nodes, has been largely investigated in expander graphs, which are sparse graphs that guarantee strong connectivity properties [2, 8, 17, 9].

2 Deterministic Construction of Sparse Clique Emulators

In this section we provide a deterministic construction yielding a perfect tradeoff between number of edges and number of rounds in clique emulation.

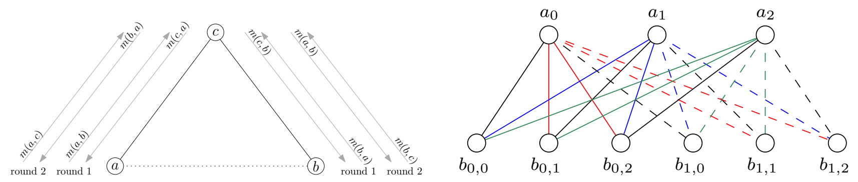

Fig. 2. *(Left) Emulation of removed edge $\{a, b\}$ ($m(x, y)$ denotes the message from x to y). (Right) Emulating K_9 with $K_{3,6}$. The plain red path $(b_{0,1}, a_0, b_{0,2})$ is used at the 1st round for exchanging messages between $b_{0,1}$ and $b_{0,2}$, and, at the 2nd round, it is used for sending messages from $b_{0,1}$ to $b_{1,2}$, and from $b_{0,2}$ to $b_{1,1}$.*

Theorem 1. *Let $n \geq 1$, and $k \geq 3$. There is an n-node graph with at most $\lceil \frac{k-2}{(k-1)^2} n^2 \rceil$ edges that can emulate the n-node clique in k rounds. Also, there is an n-node graph with at most $\frac{1}{3}n^2$ edges that can emulate the n-node clique in 2 rounds.*

Proof. First, we show that there is an n-node graph with at most $\frac{1}{3}n^2$ edges that can emulate the n-node clique in 2 rounds. For this purpose, recall that the so-called Johnson graph $J(n, r)$ has vertex set composed of all the r-element subsets of the set $\{1, \dots, n\}$, and two vertices are adjacent iff they meet in a $(r-1)$-element set.

Fact 1 *There exists an independent set I of size at least $\lceil \frac{1}{n}\binom{n}{3} \rceil$ in the Johnson graph $J(n, 3)$.*

To establish this fact, for every k, $0 \leq k < n$, let us consider the set

$$I_k = \{\{x, y, z\} \in V(J(n, 3)) \mid x + y + z \equiv k \pmod{n}\}$$

Every set I_k is an independent set. Indeed, if two triples $\{x, y, z\}$ and $\{x, y, z'\}$ are both in I_k, then $x+y+z \equiv k \pmod{n}$ and $x+y+z' \equiv k \pmod{n}$. Therefore, $z \equiv z' \pmod{n}$, which implies $z = z'$, because $z, z' \in \{1, \dots, n\}$. Observe that $\{I_0, \dots, I_{n-1}\}$ is a partition of $V(J(n, 3))$. Therefore, one of them has size at least $\lceil \frac{1}{n}\binom{n}{3} \rceil$, which establishes Fact 1.

Let I as in Fact 1. Note that for any $\{a, b, c\} \in I$, none of the edges $\{a, b\}, \{a, c\}, \{b, c\}$ are appearing in any other triples of I. Thus, the edge $\{a, b\}$ of the complete graph can be emulated by the path $\{a, c\}, \{b, c\}$ without congestion resulting from the emulation of another edge $\{a', b'\}$. Moreover, the edge $\{a, b\}$ itself does not belong to any path used to emulate other edges. It follows that one can remove $|I|$ edges from K_n, one from each triple in the independent set I, and all removed edges can be emulated by edge-disjoint paths of length 2. Fig. 2(left) shows how to emulate the six communications $x \to y$ for every ordered pair (x, y), $x \in \{a, b, c\}$, $y \in \{a, b, c\}$, $x \neq y$, in just 2 rounds. It follows that there is an n-node graph with at most $\frac{n^2}{3}$ edges that can emulate the n-node clique in 2 rounds.

We now move on with the general case, that is, we show that there is an n-node graph with at most $\lceil \frac{n^2(k-2)}{(k-1)^2} \rceil$ edges that can emulate the n-node clique in k rounds.

Fact 2 *All-to-all communication between the nodes of the same part of the complete bipartite graph $K_{r,r}$ can be performed in 2 rounds.*

Indeed, let A and B be the two parts of $K_{r,r}$, where the nodes in A and B are labeled $a_0, \ldots, a_{r-1}$ and $b_0, \ldots, b_{r-1}$, respectively. Let us consider $a_i \in A$, and its message for node $a_j \in A$. This message is routed via node $b_k \in B$ where $i + j + k \equiv 0 \pmod{r}$. This guarantees that each edge is used at most once in each direction, at each round. Indeed, sender a_i chooses different intermediate nodes to route messages to the different receivers a_j, $j \neq i$. Similarly, for the same receiver j, different senders a_i, $i \neq j$, choose different intermediate nodes. This proves Fact 2.

By performing the above routing scheme in parallel, we directly get the following:

Fact 3 *Let A and B be the two parts of the complete bipartite graph $K_{r,kr}$, and let us partition the nodes of B into k groups $B_0, \ldots, B_{k-1}$ of r nodes each. The k all-to-all communication patterns between the nodes of B_i can be performed in parallel for all $i \in \{0, \ldots, k-1\}$, in 2 rounds, also in parallel to all-to-all communication between the nodes of A.*

We have now all the ingredients to establish the general case of Theorem 1. Let $k \geq 1$, and let $K_{r,kr}$ be the n-node complete bipartite graph with $r = \frac{n}{k+1}$ nodes in the first part A, and $kr = \frac{nk}{k+1}$ nodes in the other part B. Note that $K_{r,kr}$ has $kr^2 = \frac{n^2k}{(k+1)^2}$ edges. We show how to perform all-to-all in $K_{r,kr}$ in $k+2$ rounds. We divide the kr nodes of B into k groups $B_0, \ldots, B_{k-1}$ of r nodes each. For $i \in \{0, \ldots, k-1\}$, we set $B_i = \{b_{i,j}, 0 \leq j \leq r-1\}$ — cf. Figure 2(right). We describe a routing scheme that allows the kr nodes of B to perform all-to-all, by relaying their messages using the r nodes of A. Routing is achieved by repeating k times the all to all routing protocol in Fact 3, where, at each phase $s = 1, \ldots, k$, nodes of B_i perform the communications with the nodes in $B_{j+s \bmod k}$. Importantly, the above routing scheme does not require $2k$ rounds but only $k+1$ rounds, because the kr nodes in B do not have to wait for receiving relayed messages in order to start sending new messages, and the phases can be pipelined. One more round is used to route the direct communication between every node in A and every node in B. Interestingly, during the $k+1$ rounds needed to perform all-to-all communications between the nodes in B, the edges are always used in both directions, except for the first and last round. We can use these two rounds to let the nodes in A perform their own all-to-all among them using the same routing pattern as in Fact 2. In total, in the $\frac{n^2k}{(k+1)^2}$-edge graph $K_{r,kr}$, all-to-all is performed in $k+2$ rounds. □

We complete the section by showing that the bounds in Theorem 1 provide an essentially optimal tradeoff between the number of rounds k performed in the emulation, and the number of edges m of the emulator. A trivial lower bound $\frac{1}{2}\frac{n(n-1)}{k}$ can be obtained by noticing that every node must have degree at least $\frac{n-1}{k}$ for receiving $n-1$ messages in k rounds. The following theorem improves this trivial bound by a factor 2, and matches with the bound in Theorem 1.

Property 1. Let $n \geq 1$, $k \in \{1, \ldots, n-1\}$, and let G be an n-node graph that can emulate the n-node clique in k rounds. Then G has at least $\frac{n(n-1)}{k+1}$ edges.

Proof. Let m be the number of edges of G. There are $\binom{n}{2}$ pairs of nodes in K_n, communicating $n(n-1)$ messages in total. In G, only m pairs of nodes are directly connected. All the other $\binom{n}{2} - m$ pairs of nodes are not directly connected, and they are at least at distance 2 in G. Thus, the number of mesages generated to route the messages corresponding to these pairs of nodes is at least $4(\binom{n}{2} - m)$. The total number of messages to be transferred is thus at least $2m + 4(\binom{n}{2} - m)$. Since one communication round in G can route at most $2m$ messages, it follows that any routing protocol requires at least $\frac{2m+4\binom{n}{2}-4m}{2m} = \frac{n(n-1)}{m} - 1$ rounds of communication. Thus, $k \geq \frac{n(n-1)}{m} - 1$, which implies $m \geq \frac{n(n-1)}{k+1}$. $\square$

3 Randomized Construction of Sparse Clique Emulators

In this section, we consider clique emulation by Erdős-Rényi random graphs $\mathcal{G}_{n,p}$. Our main result is the following.

Theorem 2. *Let $c \geq 0$, $n \geq 1$, $\alpha = \sqrt{(3+c)e/(e-2)}$ where e is the base of the natural logarithm, and $p \geq \alpha\sqrt{\ln n/n}$. For $G \in \mathcal{G}_{n,p}$,*

$$\Pr[G \textit{ can emulate } K_n \textit{ in } O(\min\{\tfrac{1}{p^2}, np\}) \textit{ rounds}] \geq 1 - O(\tfrac{1}{n^{1+c}})$$

where the big-O notations hide the dependency in c.

Proof. Let $G \in \mathcal{G}_{n,p}$. The proof works as follows. For each missing edge in G between two nodes u and v, we route the messages between these nodes via an intermediate node w, i.e., along a path (u, w, v) of length 2. The intermediate node is picked at random among all nodes w such that $\{u, w\} \in E(G)$, and $\{w, v\} \in E(G)$. To analyze the load of the edges, we have to overcome two problems. First, the load of an edge is not necessarily independent from the load of another edge. Second, we are interested in the maximum, taken over all edges, of the load of the edges. As a consequence, an analysis based only on the expectation of the load of each edge may not yield accurate results. Instead, we base our analysis on a double application of a balls-into-bins protocol.

We aim at constructing a path for routing the messages between every pair of nodes that are not directly connected in G. As said before, the alternative paths used to replace missing edges are of length 2, and the probability expressed in the statement of the theorem reflects the probability that such paths exist, without too much congestion. More specifically, let us consider a missing edge

$\{i,j\}$ in G. Let $S_{i,j}$ be the set of common neighbors to i and j in G. The message from i to j is aimed at being routed via some intermediate node $k \in S_{i,j}$. The first question to address is thus: how large is $S_{i,j}$? To answer this question, let $\mathcal{E}_{i,j}$ be the event "there are at least $\frac{np^2}{e}$ different paths of length 2 between i and j", and let $\mathcal{E} = \bigcap_{\{i,j\}\notin E(G)} \mathcal{E}_{i,j}$.

Fact 4 *Let $\alpha_c = \sqrt{(c+3)e/(e-2)}$, and $p \geq \alpha_c \sqrt{\ln n/n}$. Then*

$$\Pr[\mathcal{E}] \geq 1 - \frac{1}{n^{c+1}}.$$

To establish this fact, let $X_{i,j,k}$ be the Bernoulli random variable, for $\{i,j\} \notin E(G)$, such that $X_{i,j,k} = 1$ iff $k \in S_{i,j}$, i.e., $\{i,k\} \in E(G)$ and $\{k,j\} \in E(G)$. Then let $X_{i,j} = \sum_{k=1}^{n} X_{i,j,k}$. We have $\Pr[X_{i,j,k} = 1] = p^2$, and, for a fixed pair i,j, the variables $X_{i,j,k}$, $k = 1,\ldots,n$, are mutually independent. Thus, using Chernoff bounds, we get:

$$\Pr[X_{i,j} \leq \frac{np^2}{e}] \leq e^{(\frac{2}{e}-1)np^2}.$$

By union bound, it follows that

$$\Pr[\bigcup_{\{i,j\}\notin E(G)} \overline{\mathcal{E}_{i,j}}] \leq n^2 e^{(\frac{2}{e}-1)np^2} \leq \frac{1}{n^{c+1}}$$

as desired, where the last inequality holds because $p \geq \alpha_c \sqrt{\ln n/n}$.

In addition to Fact 4, we will also use the following known result:

Lemma 1 ([21]). *Let $X_1,\ldots,X_n$ be a sequence of random variables in an arbitrary domain, and let $Y_1,\ldots,Y_n$ be a sequence of binary random variables, with the property that Y_i is a function of the variables $X_1,\ldots,X_{i-1}$. If, for every $i = 1,\ldots,n$, we have $\Pr[Y_i = 1|X_1,\ldots,X_{i-1}] \leq q$ then $\Pr[\sum_{i=1}^{n} Y_i \geq k] \leq \Pr[B(n,q) \geq k]$ where $B(n,q)$ denotes the binomial distribution of parameters n and q.*

Our path construction algorithm for every missing edge $\{i,j\} \notin E(G)$ is sequential, and proceeds as follows. For every $\{i,j\} \notin E(G)$, the path from i to j is not necessarily the same as the path from j to i. We process all ordered pairs of nodes (i,j) in n phases, where Phase i, $i = 1,\ldots,n$, constructs all paths (i,j) for $\{i,j\} \notin E(G)$, in increasing order of j. Assume already fixed a set of paths, corresponding to previously considered sender-receiver pairs, and consider now the pair (i,j) (of course corresponding to the missing edge $\{i,j\} \notin E(G)$). The previously constructed paths induce some load on each edge of G, corresponding to the number of paths using that edge. The choice of the path for (i,j) depends on this load, and is inspired from the power of two choices in balls-and-bins protocols. Precisely, for suitable parameters d and r, node i repeats r times the following: pick d incident edges $\{i,k\}$ uniformly at random, and select the least loaded one. Once this is done, node j picks the least loaded edge among the r edges selected by i.

Let $I_{i,j}$ be the node selected to route the message from sender i to receiver j. Messages from i to j will be routed along the path $P_{i,j} = (i, I_{i,j}, j)$. For $h \geq 0$, let $b_{i,h}(j)$ be the number of edges $\{i,k\}$ of load at least h after deciding the intermediate nodes $I_{i,1}, \ldots, I_{i,j}$ of the first j receivers for sender i. We define the following quantities:

$$x = \left\lceil \frac{e^{5+c}}{p^2} \right\rceil \text{ and } \beta = \frac{np^2}{e^{5+c}}.$$

Since $b_{i,x}(n) \leq n/x$, it follows from the above that $b_{i,x}(n) \leq \beta$. Now, let

$$\ell(j) = |\{j' \leq j : I_{i,j'} = I_{i,j}\}|.$$

We define the random variables $Z_{i,j}$ where

$$Z_{i,j} = \begin{cases} 1 \text{ if } \ell(j) \geq x+1 \\ 0 \text{ otherwise.} \end{cases}$$

Hence $Z_{i,j} = 1$ is the bad event that the edge between node i and the intermediate node $I_{i,j}$ used to route from i to j is heavily loaded by i. Conditioned on the fact that $\mathcal{E}$ holds (cf. Fact 4), we get that

$$\Pr[Z_{i,j} = 1] \leq r \left(\frac{\beta}{np^2/e} \right)^d.$$

We let q be the right hand side of the above equation. Let us now consider $Z_i = \sum_{j=1}^{n} Z_{i,j}$. Observe that $Z_{i,j}$ is a function of $I_{i,1}, \ldots, I_{i,j-1}$. Therefore, by Lemma 1 we get that

$$\Pr[Z_i \geq k] \leq \Pr[B(n,q) \geq k].$$

So, in particular, $\Pr[Z_i \geq 1] \leq \Pr[B(n,q) \geq 1]$. We now set $d = \ln n$, and $r \leq n$ (a suitable r will be specified thereafter). Thanks to this choice of d and r, we have $q \leq \frac{1}{n^{3+c}}$, and therefore

$$\Pr[Z_i \geq 1] \leq \Pr[B(n, \frac{1}{n^{3+c}}) \geq 1] \leq \mathbf{E}[B(n, \frac{1}{n^{3+c}})] \leq \frac{1}{n^{2+c}}.$$

Let $Z = \sum_{i=1}^{n} Z_i$. By union bound, we get $\Pr[Z \geq 1] \leq \frac{1}{n^{1+c}}$.

Using a similar analysis, from the perspective of the receiver, and defining the corresponding random variables $Z'_{i,j}$ capturing the load of the edges incident to a receiver j, and $Z'_j = \sum_{i=1}^{n} Z'_{i,j}$, we get

$$\Pr[Z'_j \geq 1] \leq \Pr[B(n,q') \geq 1]$$

where

$$q' = \left(1 - \left(1 - \frac{e\beta}{np^2}\right)^d\right)^r.$$

We get $q' \leq \frac{1}{n^{3+c}}$ by setting $d = \ln n$ and $r = (c+3)\ n^{\epsilon}\ \ln n$ for $\epsilon = -\ln(1-\frac{1}{e^{4+c}})$. By this setting of d and r, we get that

$$\Pr[Z'_j \geq 1] \leq \Pr[B(n, \frac{1}{n^{3+c}}) \geq 1] \leq \mathbf{E}[B(n, \frac{1}{n^{3+c}})] \leq \frac{1}{n^{2+c}}.$$

Let $Z' = \sum_{j=1}^{n} Z'_j$. By union bound, we get $\Pr[Z' \geq 1] \leq \frac{1}{n^{1+c}}$.

Therefore, altogether, we get that

$$\Pr[Z = 0 \text{ and } Z' = 0 \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] \geq (1 - \frac{1}{n^{1+c}})^3 \geq 1 - \frac{3}{n^{1+c}}.$$

In other words, w.h.p., the load of all edges is no more than $x = O(1/p^2)$. On the other hand, with a similar argument as for proving that the degree is large, we have that, w.h.p., the degree of all nodes is at most enp, and therefore the load of an edge does not exceed enp. □

4 MST Construction in Core-Periphery Networks

In [1], a randomized algorithm for Minimum Spanning Tree (MST) construction is presented. It runs in $O(\log^2 n)$ rounds with high probability. We improve this result by describing a deterministic algorithm for MST construction that runs in just $O(\log n)$ rounds. Recall that, for the MST construction task, every node is given as input the weight $w(e)$ of each of its incident edges e. These weights are supposed to be of values polynomial in the size n of the network, and thus each weight can be stored on $O(\log n)$ bits. The output of every node is a set of incident edges, such that the collection of all outputs forms an MST of the network. Without loss of generality, all weights are supposed to be different (since, otherwise, it is sufficient to add to each edge the identities of the extremities of that edge).

Theorem 3. *The MST construction task can be solved in $O(\log n)$ rounds in core-periphery networks under the* CONGEST *model.*

Proof. As usually in the distributed setting, the general idea of the algorithm is based on the sequential Borůvka's algorithm for MST construction, consisting in merging subtrees called *fragments*. Recall that, in Borůvka's algorithm, there are initially n fragments, where each node alone forms a fragment. Each fragment has an ID. Initially, the identity of each fragment is the ID of the single node in the fragment. Then the algorithm proceeds in at most $\lceil \log_2 n \rceil$ phases. At each phase, each fragment F computes the edge e_F of minimum weight incident to fragment F, and adds it to the MST. Fragments connected by such an edge merge, and a new phase begins. This procedure is repeated until there is only one fragment, which is the desired MST.

We first present a (deterministic) distributed algorithm running in $O(\log^2 n)$ rounds in core-periphery networks. This algorithm is composed of at most $\lceil \log_2 n \rceil$ phases, where each phase requires $O(\log n)$ rounds. Then, we show how to actually perform each phase in $O(1)$ rounds, obtaining the desired $O(\log n)$-round algorithm. Recall that a core-periphery network satisfies the three axioms listed in Section 1 where C and P denote the sets of nodes in the core and in the periphery, respectively.

The algorithm starts by an initialization phase, where each node in the periphery looks for a node in the core, which will be its *representative*. By Axiom 3 all nodes in the periphery can concurrently send messages to the core so that each message will be received by at least one node in the core after $O(1)$ rounds. So, each node in the periphery sends a request for a representative by sending its own ID to the core. Every node in the periphery then waits for an acknowledgment from nodes in the core that accepted its request. These acknowledgements follow the same route as the corresponding requests, backward. Hence, all acknowledgments are also received after $O(1)$ rounds. Every node takes as representative the core node whose acknowledgment reaches that node first. If a node receives several acknowledgments simultaneously, then it selects the one with the smallest ID. By Axiom 1, each node in the core can be the representative of at most $O(|C|)$ nodes in the periphery because its degree is at most $O(|C|)$, and thus it can receive at most $O(|C|)$ messages in $O(1)$ rounds. Every node in the core is its own representative.

We assume that the nodes in the core are sorted according to their IDs (this operation can be done in $O(1)$ rounds using all-to-all and Axiom 2). For every node in the core, we denote by $\mathrm{succ}(u)$ and $\mathrm{pred}(u)$ the successor and the predecessor of u in this order, respectively.

We heavily used the protocols in [18]. Note that the *routing* protocol in [18] requires that each node is the the source and destination of at most n messages. However, it can be trivially adapted to be applied with $O(n)$ messages, still requiring $O(1)$ rounds. Similarly, the *sorting* protocol in [18] requires that each node receives at most n keys, but, again, it can be trivially modified for allowing each node to receive $O(n)$ keys, still requiring $O(1)$ rounds.

We now explain how every phase of Borůvka's algorithm is performed.

1. Every node sends the ID of its fragment to all its neighbors.
2. Let $r(v) \in C$ and $\mathrm{id}(F)$ be the representative and the ID of the fragment F of node v, respectively. We denote by $e_F(v)$ the edge of minimum weight incident to v and connecting v to a node not in its fragment F. Each node v in the periphery sends $(e_F(v), w(e_F(v)), \mathrm{id}(F), \mathrm{id}(F'))$ to $r(v)$, where the tail of $e_F(v)$ belongs to F, and its head belongs to fragment $F' \neq F$. Observe that each node in the core receives $O(|C|)$ such messages.
3. Every node in the core, upon reception of 4-tuple $(e_F(v), w(e_F(v)), \mathrm{id}(F), \mathrm{id}(F'))$ from the nodes that it represents (including itself), selects the ones with minimum weight for each fragment F. We denote by S_1 the set of the selected edges by all nodes in the core. Note that $|S_1| = O(|C|^2)$.
4. The algorithm assigns a *leader* to each fragment. The leaders are core nodes chosen in such a way that the fragments are equally distributed among leaders. Let
$$x = \lceil |S_1|/|C| \rceil.$$

Note that $x = O(|C|)$. Given a fragment F, its leader is

$$\ell(F) = 1 + \left\lfloor \frac{|\{(u,v) \in S_1 : \mathrm{id}(F_u) < \mathrm{id}(F)\}|}{x} \right\rfloor$$

where F_u is the fragment of u. Note that $1 \leq \ell(F) \leq |C|$. For each fragment F, all edges incident to F in S_1 are sent to $\ell(F)$ by its representative holding such edges — we shall explain hereafter how this is implemented in core-periphery networks. In this way each leader can select the edge e_F of minimum weight incident to fragment F. Let S_2 be the set of all edges e_F, where F is a fragment.

5. The algorithm then aims at merging the fragments. We call *merge tree* a tree whose nodes are fragments F, and whose edges are the edges e_F connecting these fragments. Note that, in a merge tree, there are two adjacent fragments F and F' connected by two possibly distinct edges e_F and $e_{F'}$. The fragment with smallest ID that is extremity of such an edge is the root of the merge tree. The algorithm proceeds so that each leader $\ell(F)$ of a fragment F in the merge tree becomes aware of the root of the tree. The ID of this root will become the ID of the fragment resulting from merging all the fragments in the merge tree. It is possible to find the root of a tree of height h in $O(\log h)$ steps using *pointer jumping* — we shall explain hereafter how this is precisely implemented in core-periphery networks.
6. By the previous step, for every fragment F, its leader $\ell(F)$ knows the ID of the merge tree it belongs to. Moreover, for each edge (u, v) that was received by a leader from the representative $r(u)$ in step 4, the leader saved $\mathrm{id}(r(u))$. This allows leaders to notify the right representatives of the ID of the root of the merge tree.
7. Finally, the ID of every merged fragment is sent to every node v of the periphery from its representative $r(v)$ in the core.

It remains to explain how steps 4 and 5 are actually performed.

Step 4 in more details. First, observe that the parameter $x = \lceil |S_1|/|C| \rceil$ can be computed at each node of the core, as performing all-to-all communication in the core allows each core node to compute $|S_1|$. Now, we show how to distribute the fragments among the leaders such that leader $\ell(F)$ becomes aware of the edges $e_F(v) \in S_1$ incident to F.

The edges $(u, v) \in S_1$ are sorted according to the ID of the fragment F_u its tail belongs to, and are then split into groups of x edges. Again, this operation can be done in $O(1)$ rounds using the sorting protocol in [18] because $x = O(|C|)$. The kth group is assigned to the kth node of the core.

Let us consider a core node u, and let $\mathcal{F}(u)$ be the set of fragments F such that $\ell(F) = u$. Let us denote by $\mathrm{id}_{max}(u)$ (resp., $\mathrm{id}_{min}(u)$) the maximum ID (resp., minimum ID) of the fragments $F \in \mathcal{F}(u)$. Having sorted the set S_1 guaranties that the leader u receives all the edges assigned to it, except perhaps some edges starting from fragment $\mathrm{id}_{max}(u)$ that could have been delivered to succ(u). However, there are at most $x - 1$ such edges, since the representatives kept at most one edge per fragment. So, every core node u can send $\mathrm{id}_{max}(u)$ to succ(u), in order to let that node know that the leader of the fragment with ID equal to $\mathrm{id}_{max}(u)$ should be u, and not succ(u). Since each node u has then at most $x-1$ messages to transmit to pred(u), we can transmit these messages using the routing protocol in [18]. Now each leader u has all the outgoing edges of each fragment F with $\ell(F) = u$. Thus, u can compute e_F for each of these fragments. Finally, each node u in the core broadcasts the pair $(\mathrm{id}_{min}(u), \mathrm{id}_{max}(u))$ in the core so that every node in C learns the leader of each fragment.

Note that, while sorting and routing, every node keeps track of the ID of the representative nodes which originally received every edge that is manipulated by that node (this is needed in step 6).

Step 5 in more details. We show how to perform the first step of pointer jumping. Recall that, for every fragment F, the leader $\ell(F)$ knows e_F. This latter edge is the one leading toward the root of the merge tree. Assume that $e_F = (u, v)$, with $u \in F$ and $v \in F'$. The objective for the leader $\ell(F)$ is to learn to which fragment F'' is pointing the edge $e_{F'} = (u', v')$ with $u \in F'$ and $v' \in F''$. In other words, if p denotes the parent relation in a merge tree, the leader $\ell(F)$ of fragment F wants to learn the ID of $p(p(F))$. The bad news is that $\ell(F)$ cannot directly ask $\mathrm{id}(p(p(F)))$ to $\ell(p(F))$ because this could create a bottleneck at $\ell(p(F))$. Nevertheless this issue can be overcame as follows.

First, the edges in S_2 are sorted according to the IDs of the fragment of their heads, and grouped into groups whose heads belong to the same fragment. In this way, only one request is sent for each group (to the leader of the corresponding fragment). Since $x = \lceil |S_1|/|C| \rceil$, we have $x = O(|C|)$, and thus the number of requests that each leader has to make is at most $O(|C|)$.

Second, every leader does not receive more than $O(|C|)$ requests. Indeed, let $q_{u,v}$ be the number of different fragments for which a node u in the core has to send a request to leader v. Let $F_{i_1}, F_{i_2}, \dots, F_{i_{q_{u,v}}}$ be these fragments, with $\ell(F_{i_1}) = \ell(F_{i_2}) = \cdots = \ell(F_{i_{q_{u,v}}}) = v$, and $i_1 < i_2 < \cdots < i_{q_{u,v}}$. Recall that the edges in S_2 are sorted according to the IDs of the fragment of their heads. Thus, if $q_{u,v} > 1$ then the fragments $F_{i_2}, \dots, F_{i_{q_{u,v}}}$ do not appear in any list of fragments assigned to nodes with identity smaller than $\mathrm{id}(u)$. Therefore, leader v receives at least $\sum_{u \in C}(q_{u,v} - 1)$ requests for different fragments. On the other hand, every core node v is the leader of at most x fragments. Therefore $\sum_{u \in C}(q_{u,v} - 1) \leq x$. Hence the number of requests received by v is $\sum_{u \in C} q_{u,v} = O(|C|)$.

These two facts, allow the routing protocol in [18] to be used, for sending the requests to the leaders, and for receiving back their answers. Once this is done, every node u sends $\mathrm{id}(p(p(F)))$ to $\ell(F)$, for every $F \in \mathcal{F}(u)$ in a constant number of rounds, again using [18]. It follows that every leader u can learn the ID of $p(p(F))$ for every $F \in \mathcal{F}(u)$ in a constant number of rounds.

Time analysis. The initialization phase can be performed in $O(1)$ rounds thanks to Axiom 3. Step 1 trivially requires $O(1)$ rounds. Step 2 also requires $O(1)$ rounds thanks to Axiom 3. Step 3 is executed locally by each node, thus it does not require communication. Step 4 can be executed in $O(1)$ rounds using the sorting protocol in [18] because $x = O(|C|)$. Step 6 can also be performed in $O(1)$ rounds using the routing protocol in [18] because each leader handles $O(|C|)$ edges (for which it has to send a fragment ID), and each representative has to receive $O(|C|)$ messages (one for each edge it has to receive a new fragment ID). The last step is the inverse of step 2, and thus can still be executed in $O(1)$ rounds. Step 5 however requires $O(\log n)$ rounds because the merge tree might be of height $\Omega(n^\epsilon)$ for some $\epsilon > 0$. Since the number of phases is also $O(\log n)$, the total number of rounds of this algorithm is $O(\log^2 n)$.

A faster algorithm. Now, we describe how to modify the above algorithm so that it uses only $O(1)$ rounds for each phase, hence $O(\log n)$ rounds in total. Since the only step that requires a non constant number of rounds is Step 5, we show how to perform that step in $O(1)$ rounds.

The idea is to use a technique introduced first in [20], and also used in Avin et al. [1], called *amortized pointer jumping*. The reduction of long chains of pointers is deferred to later phases of Borůvka's algorithm, and only a constant number of pointer jumps are performed at each phase. This technique exploits the fact that, if a chain is long, it must contain many fragments. As a consequence, when pointer jumping completes, the resulting fragment is quite large, and other nodes involved in small fragments may continue building the MST in parallel, without waiting for large fragments to be constructed.

We show how to do a constant number of pointer jumping steps, then freezing the procedure, and resuming it later in the next phase of Borůvka's algorithm. At each step of pointer jumping, every leader u can know, for every $F \in \mathcal{F}(u)$, if the root of the merge tree has been reached. Suppose that the root has not been reached by u after a constant number of pointer jumping (i.e., the leader does not know yet the new ID of the merged fragment), and that u is currently pointing at fragment F'. In the following, node u adds a flag in its messages, which specifies that the fragment has not been resolved yet, and that it stopped at F'. This flag will be propagated to all nodes that proposed edges that start from unresolved fragments. At the next phase of Borůvka's algorithm, these nodes will propose again the same edges, by specifying also F'. Fragment F' will be used as if it was the destination fragment of the edge. In this way, for every fragment F in a merge tree whose merging has not yet been performed, the same edge e_F as before will be chosen, and other steps of pointer jumping will be performed. This insures that nodes belonging to fragments in such merge trees do not propose new edges, thus emulating a full execution of pointer jumping.

After having reduced the number of rounds for performing step 5 from $O(\log n)$ to $O(1)$, amortized, we get that the resulting algorithm just requires $O(\log n)$ rounds to construct a MST. □

5 Conclusion

We have shown how to emulate the clique by a random graph in $\mathcal{G}_{n,p}$ in time $O(\min\{\frac{1}{p^2}, np\})$ rounds, w.h.p. Hence, on dense random graphs (i.e., $p = \Omega(1)$), our simulation performs in just a multiplicative constant factor away from the optimal, and, on sparse graphs (i.e., $p \simeq \sqrt{\log n/n}$), it performs just a $\log n$ factor away from optimal. However, in general, whenever $p \gg \frac{1}{\sqrt[3]{n}}$, it performs in $O(\frac{1}{p^2})$ rounds, which is a factor $O(\frac{1}{p})$ away from the trivial lower bound $\Omega(\frac{1}{p})$. An intriguing question is whether the n-node clique can be simulated by $\mathcal{G}_{n,p}$ in just $O(\frac{1}{p})$ rounds.

Our deterministic MST algorithm for core-periphery networks performs in $O(\log n)$ rounds, improving the previously known (randomized) algorithm by a factor $\Theta(\log n)$. Recent advances in the congested clique model demonstrate that ultra fast MST algorithms exist for this later model, namely, a recent $O(\log^* n)$-round randomized algorithm [12], and a $O(\log \log n)$-round deterministic algorithm [19]. Another intriguing question is whether such ultra fast algorithms exist for core-periphery networks.

References

1. Chen Avin, Michael Borokhovich, Zvi Lotker, and David Peleg. Distributed computing on core-periphery networks: Axiom-based design. In *41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 399–410, 2014.
2. Andrei Z. Broder, Alan M. Frieze, and Eli Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM J. Comput.*, 23(5):976–989, 1994.
3. Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 143–152, 2015.
4. Keren Censor-Hillel and Tariq Toukan. On fast and robust information spreading in the vertex-congest model. In *22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 270–284, 2015.
5. Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 367–376, 2014.
6. Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 359–368, 2004.
7. Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Struct. Algorithms*, 1(4):447–460, 1990.
8. Alan M. Frieze. Disjoint paths in expander graphs via random walks: A short survey. In *Second International Conference on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 1–14, 1998.
9. Alan M. Frieze. Edge-disjoint paths in expander graphs. In *11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 717–725, 2000.
10. Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.
11. Juan A. Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302–316, 1998.

12. Mohsen Ghaffari and Merav Parter. Mst in log-star rounds of congested clique. In *35th ACM Symposium on Principles of Distributed Computing (PODC)*, 2016.
13. James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 91–100, 2015.
14. James W. Hegeman and Sriram V. Pemmaraju. Lessons from the congested clique applied to MapReduce. *Theor. Comput. Sci.*, 608:268–281, 2015.
15. James W. Hegeman, Sriram V. Pemmaraju, and Vivek Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. In *28th Int. Symposium on Distributed Computing (DISC)*, pages 514–530, 2014.
16. Shay Kutten and David Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.
17. Tom Leighton, Satish Rao, and Aravind Srinivasan. Multicommodity flow and circuit switching. In *31st Hawaii International Conference on System Sciences*, pages 459–465, 1998.
18. Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 42–50, 2013.
19. Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log \text{n})$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005.
20. Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for constant diameter graphs. In *20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 63–71, 2001.
21. Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.
22. Hiroaki Ookawa and Taisuke Izumi. Filling logarithmic gaps in distributed complexity for global problems. In *41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 8939 of *LNCS*, pages 377–388. Springer, 2015.
23. David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.
24. Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *43rd ACM Symposium on Theory of Computing (STOC)*, pages 363–372, 2011.