The Laplacian Paradigm in the Broadcast Congested Clique

Sebastian Forster
sebastian.forster@plus.ac.at
University of Salzburg
Salzburg, Austria

Tijn de Vos
tijn.devos@plus.ac.at
University of Salzburg
Salzburg, Austria

ABSTRACT
In this paper, we bring the main tools of the Laplacian paradigm to the Broadcast Congested Clique. We introduce an algorithm to compute spectral sparsifiers in a polylogarithmic number of rounds, which directly leads to an efficient Laplacian solver. Based on this primitive, we consider the linear program solver of Lee and Sidford [30]. We show how to solve certain linear programs up to additive error $\epsilon$ with $n$ constraints on an $n$-vertex Broadcast Congested Clique network in $\tilde{O}(\sqrt{n} \log(1/\epsilon))$ rounds. Using this, we show how to find an exact solution to the minimum cost flow problem in $\tilde{O}(\sqrt{n})$ rounds.

CCS CONCEPTS
• Theory of computation → Distributed algorithms; Sparsification and spanners; Network flows.

KEYWORDS
Broadcast Congested Clique, Laplacian solver, minimum cost flow, linear programs, spectral sparsification

ACM Reference Format:
Sebastian Forster and Tijn de Vos. 2022. The Laplacian Paradigm in the Broadcast Congested Clique. In Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC ’22), July 25–29, 2022, Salerno, Italy. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3519270.3538436

1 INTRODUCTION
In this paper, we study algorithms for the Broadcast Congested Clique (BCC) model [17]. In this model, the (problem-specific) input is distributed among several processors and the goal is that at the end of the computation each processor knows the input or at least the share of the output relevant to it. The computation proceeds in rounds and in each round each processor can send one message to all other processors. We can also view the communication as happening via a shared blackboard to which each processor may write (in the sense of appending) at most one message per round. The main metric in designing and analyzing algorithms for the Broadcast Congested Clique is the number of rounds performed by the algorithm.

A typical way of for example distributing an $n \times n$ input matrix among $n$ processors would be that initially processor $i$ only knows row $i$ of the matrix. In many graph problems, this input matrix is the adjacency matrix of the graph. If communication with other processors is only possible along the edges of this graph, then the resulting model is often called the Broadcast CONGEST model [34]. Note that the unicast versions of these models, in which each processor may send a different message to each (neighboring) processor, are known as the Congested Clique [33] and the CONGEST model [37], respectively.

In this paper, we bring the main tools of the so-called Laplacian paradigm to the BCC model. In a seminal paper, Spielman and Teng developed an algorithm for approximately solving linear systems of equations with a Laplacian coefficient matrix in a near-linear number of operations [42]. The Laplacian paradigm [43] refers to exploring the applications of this fast primitive in algorithm design. In a broader sense, this paradigm is also understood as the more general idea of employing linear algebra methods from continuous optimization outside of their traditional domains. Using such methods is very natural in distributed models because a matrix-vector multiplication can be carried out in a single round if each processor stores one coordinate of the vector. In recent years, this methodology has been successfully employed in the CONGEST model [8, 22] and in particular, solvers for Laplacian systems with near-optimal round complexity have been developed for the CONGEST model – in networks with arbitrary topology [19] and in bounded-treewidth graphs [2] – and for the HYBRID model [2]. In this paper, we switch the focus to the BCC model and show that it allows a faster implementation of the basic Laplacian primitive.

What further makes the BCC model intriguing is that – in contrast to the Congested Clique – for several problems no tailored BCC algorithms are known that are significantly faster than low-diameter versions of (Broadcast) CONGEST model algorithms. Consider, for example, the single-source shortest path problem. In the (Broadcast) CONGEST model, the fastest known algorithm takes $\tilde{O}(\sqrt{n}D^{3/4} + D)$ rounds [11], where $D$ is the diameter of the underlying (unweighted) communication network. In the BCC model, the state of the art for this problems is $\tilde{O}(\sqrt{n})$ rounds [36], which essentially is not more efficient than the special case $D = 1$ of the Broadcast CONGEST model. In the Congested Clique model however, $\sqrt{n}$ is not a barrier for this problem as it can be solved in $\tilde{O}(n^{1/6})$ rounds [9] on undirected graphs. A similar classification can be made for directed graphs [10, 20]. This naturally leads to the question whether BCC algorithms can be developed that are faster than their CONGEST model counterparts, since it is not clear which one dominates the other in strength.

It has recently been shown that in the CONGEST model, the maximum flow problem as well as the unit-capacity minimum
cost flow problem can be solved in \(O(m^{3/7+\epsilon}(\sqrt{n}D^{1/4}+D))\) rounds [19], where \(m\) denotes the number of edges of the input graph; note that this round complexity can only be sublinear in \(n\) for sparse graphs.

Our contributions. Our main result is an algorithm that solves the minimum cost flow problem\(^2\) (which generalizes both the single-source shortest path problem and the maximum flow problem) in \(O(\sqrt{n})\) rounds in the BCC model, which in particular is sublinear for any graph density and matches the currently known upper bounds for the single-source shortest path problem.

Theorem 1.1. There exists a Broadcast Congested Clique algorithm that, given a directed graph \(G = (V, E)\) with integral costs \(q \in \mathbb{Z}^m\) and capacities \(c \in \mathbb{Z}_{>0}^m\) with \(||q||_\infty \leq M\) and \(||c||_\infty \leq M\) computes a minimum cost maximum \(s\)-\(t\) flow with high probability in \(O(\sqrt{n}\log^3 M)\) rounds.

In obtaining this result, we develop machinery of the Laplacian paradigm that might be of independent interest. The first such tool is an algorithm for computing a spectral sparsifier in the Broadcast CONGEST model.

Theorem 1.2. There exists an algorithm that, given a graph \(G = (V, E, w)\) with positive real weights satisfying \(||w||_\infty \leq U\) and an error parameter \(\epsilon > 0\), with high probability outputs a \((1 \pm \epsilon)\)-spectral sparsifier \(H\) of \(G\), where \(|H| = O\left(ne^{-2}\log^2 n\right)\). Moreover, we obtain an orientation on \(H\) such that with high probability each edge has out-degree \(O(\log^2(n)/\epsilon^2)\). The algorithm runs in \(O\left(\log^2(n)e^{-2}\log(nU/\epsilon)\right)\) rounds in the Broadcast CONGEST model.

At a high level, our sparsifier algorithm is a modification of the CONGEST-model algorithm of Koutis and Xu [27]; essentially, uniform edge sampling is trivial in the CONGEST model, but challenging in the Broadcast CONGEST model. Note that the sparsifier algorithm of Koutis and Xu being restricted to the CONGEST model is a major obstacle for implementing the CONGEST-model Laplacian solver of Forster et al. [19] also in the Broadcast CONGEST model.

Making the sparsifier known to every processor leads to a simple residual-correction algorithm for solving systems of linear equations with a Laplacian coefficient matrix up to high precision in the BCC model. Note that there is reduction [23] from solving linear equations with symmetric diagonally dominant (SSD) coefficient matrices to solving linear equations with Laplacian coefficient matrices, which also applies in the Broadcast Congested Clique.

Theorem 1.3. There exists an algorithm in the Broadcast Congested Clique model that, given a graph \(G = (V, E, w)\), with positive real weights satisfying \(||w||_\infty \leq U\) and Laplacian matrix \(L_G\), a parameter \(\epsilon \in (0, 1/2]\), and a vector \(b \in \mathbb{R}^n\), outputs a vector \(y \in \mathbb{R}^n\) such that \(||x - y||_{L_G} \leq \epsilon||x||_{L_G}\) for some \(x \in \mathbb{R}^n\) satisfying \(L_G x = b\). The algorithm needs \(O(\log^2(n)\log(nU))\) preprocessing rounds and takes \(O(\log^2(1/\epsilon)\log(nU/\epsilon))\) rounds for each instance of \((b, \epsilon)\).

\(^2\)Note that in contrast to the algorithm of Forster et al. [19], we do not need to assume unit capacities.

Finally, we show how to implement the algorithm of Lee and Sidford [30]\(^3\) for solving linear programs up to small additive error in \(O(\text{rank})\) iterations in the BCC model. Here, the rank refers to the constraint matrix of the LP and in each iteration a linear system needs to be solved. If the constraint matrix has a special structure – which is the case for the LP formulation of the minimum cost flow problem – then a high-precision Laplacian solver can be employed for this task.

Theorem 1.4. Let \(A \in \mathbb{R}^{m \times n}\) be a constraint matrix with rank\((A)\) = \(n\), let \(b \in \mathbb{R}^n\) be a demand vector, and let \(c \in \mathbb{R}^n\) be a cost vector. Moreover, let \(x_0\) be a given initial point in the feasible region \(\Omega^n := \{x \in \mathbb{R}^m : A^T x = b, \ l \leq x \leq u\}\). Suppose a Broadcast Congested Clique network consists of \(n\) vertices, where each vertex \(i\) knows both every \(j\)-th row of \(A\) for which \(A_{ij} \neq 0\) and knows \((x_0)_j\) if \(A_{ij} \neq 0\). Moreover, suppose that for every \(y \in \mathbb{R}^n\) and positive diagonal \(D \in \mathbb{R}^{m \times m}\) we can compute \((A^T DA)^{-1} y\) up to precision \(\text{poly}(1/m)\) in \(T(n, m)\) rounds. Let \(U := \max(||1/(w-x_0)||_{\infty}, ||1/(w_0-I)||_{\infty}, ||1/(U-I)||_{\infty}, ||1/(U-L)||_{\infty}, ||1/(U-L)||_{\infty})\). Then with high probability the Broadcast Congested Clique algorithm \(LPSolve\) outputs a vector \(x \in \Omega^n\) with \(e^T x \leq OPT + \epsilon\) in \(O(\sqrt{n}\log(U/e)(\log^2(U/e) + T(n, m)))\) rounds.

While this approach of solving LPs is inherently parallelizable (as the PRAM depth analysis of Lee and Sidford indicates), several steps pose a challenge for the BCC model and require more than a mere “translation” between models. In particular we need to use a different version of the Johnson-Lindenstrauss lemma to approximate leverage scores. Further we give a BCC algorithm for projecting vectors on a mixed norm ball.

As in the approach of Lee and Sidford, our main result on minimum cost maximum flow then follows from plugging a suitable linear programming formulation of the problem into the LP solver.

Overview. We provide a visual overview of the results in this paper and how they are interconnected in Figure 1.

```
| Probabilistic Spansners | Spectral Sparsifiers | Laplacian Solve | LP Solver | Min Cost Flow |
|------------------------|---------------------|----------------|----------|--------------|
| BC Section 3.1         | BC Section 3.2      | BC Section 3.3 | BC Section 4 | BCC Section 5 |
```

Figure 1: An overview of the results in this paper. We denote BC for Broadcast CONGEST and BCC for Broadcast Congested Clique.

To compute spectral sparsifiers in the Broadcast CONGEST model, we follow the setup of Koutis and Xu [27]. Roughly said, this consists of repeatedly computing spanners and retaining each edge that is not part of a spanner with probability \(1/4\). While this easily allows for an implementation in the CONGEST model (as pointed out by Koutis and Xu), it is not clear how to do this in a broadcast model – neither the Broadcast CONGEST model, nor the more powerful Broadcast Congested Clique\(^4\). A straightforward

\(^3\)Note that in the more technical parts of our paper we explicitly refer to the arXiv preprints [29] and [31] instead of the conference version [30].

\(^4\)We believe that it would be interesting to explore whether the bounded-independence sampling technique of Doron et al. [15] could also be applied to the algorithmic framework of Koutis and Xu [27]. Such a sampling method based on a random seed of polylogarithmic size would also significantly simplify an argument in the quantum sparsifier algorithm of Apers and de Wolf [5]. Note that in the Broadcast Congested
way to sample an edge would be that one of its endpoints (say the one with the lower ID) decides if it should further exist. The problem with this approach is that a vertex might be responsible for performing the sampling of a polynomial number of incident edges and the broadcast constraint prevents this vertex from sharing the result with each of the corresponding neighbors. We overcome this obstacle as follows. We explicitly maintain the probability that an edge still exists in the current iteration of the sparsifier algorithm of Koutis and Xu. Every time that an edge should be added to the current iteration’s spanner according to the spanner algorithm, one of the endpoints samples whether the edge exists using the maintained probability. Due to the vertex’ subsequent action in the spanner algorithm, the corresponding neighbor can deduce the result of the sampling. We show that this idea of implicitly learning about the result of the sampling can be implemented by modifying the spanner algorithm of Baswana and Sen [6]. We present our modification to compute a spanner on a “probabilistic” graph (in the sense described above) in Section 3.1. In Section 3.2, we prove that this can be plugged into the framework of Koutis and Xu to compute a spectral sparsifier in the Broadcast CONGEST model. Subsequently, we show in Section 3.3 that the spectral sparsifier can be used for Laplacian solving with standard techniques.

In Section 4, we present our LP solver. Given a linear program of the form
\[
\min \sum_{x \in \mathbb{R}^{n}} c^T x,
\]
for some constraint matrix \( A \in \mathbb{R}^{m \times n} \) and some convex region \( R \subseteq \mathbb{R}^m \), Lee and Sidford [30, 31] show how to find an \( \epsilon \)-approximate solution in \( \tilde{O}(\text{rank}(A) \log(1/\epsilon)) \) time. An implementation of this algorithm in the Broadcast Congested Clique is rather technical and needs new subroutines, the main one being our Laplacian solver. The algorithm is an interior point method that uses weighted path finding to make progress. The weights used are the Lewis weights, which can be approximated up to sufficient precision using the computation of leverage scores, which are defined as
\[
\sigma(M) = \text{diag}(M^{T} M)^{-1/2} M^{T}.
\]
where in our case \( M = DA \), for some diagonal matrix \( D \). Com-puting leverage scores exactly is expensive, hence these too are approximated. This can be done by using the observation that \( \sigma(M) = ||M^{T} M||^{-1/2} \) and the Johnson-Lindenstrauss lemma [25], which states that there exists a map \( Q \in \mathbb{R}^{k \times m} \) such that \( (1 - \epsilon)||x||^2 \leq ||Qx||^2 \leq (1 + \epsilon)||x||^2 \), for polylogarithmic \( k \). Nowadays, several different (randomized) constructions for \( Q \) exist. A common choice in the realm of graph algorithms [31, 40] is to use Achlioptas’ method [1], which samples each entry of \( Q \) with a binary coin flip. However, this is in practice not feasible in the Broadcast Congested Clique: we would need a coin flip for every edge, which can be performed by one of the endpoints, but cannot be communicated to the other endpoint due to the broadcast constraint. Instead we use the result of Kane and Nelson [26], that states we need only a polylogarithmic number of random bits in total. These can simply be sampled by one vertex and broadcast to all the other, who then internally construct \( Q \). Now if we can multiply both \( A \) and \( A^{T} \) by a vector, and solve linear systems involving \( A^{T} DA \), for diagonal \( D \), then we can compute these leverage scores efficiently. These demands on \( A \) are not unreasonable when we consider graph problems, because in such cases the constraint matrix will adhere to the structure of the graph Laplacian, and hence our Laplacian solver can be applied.

A second challenge in implementing Lee and Sidford’s LP solver is a subroutine that computes projections on a mixed norm ball. To be precise: for \( a, I \in \mathbb{R}^m \) distributed over the network, the goal is to find
\[
\arg \max_{a} \langle a, x \rangle, \quad ||x||_2 + ||a - x||_\infty \leq 1.
\]
We show that we can solve this maximization problem when we know the sums \( \sum_{k \in [i]} a_k \), \( \sum_{k \in [i]} x_k \), and \( \sum_{k \in [i]} ||a_k||_k \) for all \( i \in [m] \). Computing such a sum for fixed \( i \) is feasible in a polylogarithmic number of rounds. Moreover, we show that we do not need to inspect these sums for all \( i \in [m] \), but that we can do a binary search, which reduces the total time complexity to polylogarithmic.

Following Lee and Sidford [30], we apply the LP solver to an LP formulation of the minimum cost maximum flow problem in Section 5. The corresponding constraint matrix \( A \) has \( O(n) \) rows and thus rank \( O(\sqrt{n}) \). Furthermore, \( (A^{T} DA) \) (for any diagonal matrix \( D \)) is symmetric diagonally dominant and thus \( (A^{T} DA)^{\frac{-1}{2}} \) can be approximated to high precision in a polylogarithmic number of rounds with our Laplacian solver. We only need to solve the LP up to precision \( \Theta(1/m^{O(1)}) \), since we can round the approximate solution to an exact solution. Hence, the minimum cost flow LP can be solved in \( \tilde{O}(\sqrt{n}) \) rounds.

2 PRELIMINARIES

First we detail the models we will be working with. Next, we review spanners and sparsifiers, and how to construct the latter from the former. Then we show how spectral sparsifiers can be used for solving Laplacian systems. Finally, we introduce flow problems on weighted graphs.

2.1 Models

In this paper, we consider multiple variants of message passing models with bandwidth constraints on the communication. Let us start by defining the CONGEST model. The CONGEST model [37] consists of a network of processors, which communicate in synchronous rounds. In each round, a processor can send information to its neighbors over a non-faulty link with limited bandwidth. We model the network of processors by a graph \( G = (V, E) \), where we identify the processors with the vertices and the communication links with the edges. We write \( n = |V| \) and \( m = |E| \). Each vertex has a unique identifier of size \( O(\log n) \), initially only known by the vertex itself and its neighbors. Computation in this model is done in rounds. At the start of each round, each vertex can send one message to each of its neighbors, and receives messages from them. The messages are of size at most \( B = \Theta(\log n) \). Before the next
round, each vertex can perform (unlimited) internal computation. We measure the efficiency of an algorithm by the number of rounds.

In the CONGEST model, each vertex can send distinct messages to each of its neighbors. A more strict assumption on message passing, is that each vertex sends the same message to each of its neighbors, essentially broadcasting it to its neighbors. The CONGEST model together with this assumption is called the Broadcast CONGEST model [34].

Alternatively, we can define the Laplacian matrix in terms of the Laplacian of a weighted graph \( G \), or the graph Laplacian, is a matrix \( L \in \mathbb{R}^{n \times n} \) defined by

\[
L_{uv} = \begin{cases} 
-w(u,v) & \text{if } u \text{ is adjacent to } v; \\
\sum_{x \in V} w(u,x) & \text{if } u = v; \\
0 & \text{else.}
\end{cases}
\]

Alternatively, we can define the Laplacian matrix in terms of the edge-vertex incidence matrix \( B \), defined by

\[
B(ε, v) := \begin{cases} 
1 & \text{if } ε = e_{\text{head}}; \\
-1 & \text{if } ε = e_{\text{tail}}; \\
0 & \text{otherwise.}
\end{cases}
\]

The Laplacian then becomes \( L = B^TWB \), where \( W \in \mathbb{R}^{m \times m} \) is the diagonal matrix defined by the weights: \( W_{ee} := w(ε) \).

Spectral sparsifiers were first introduced by Spielman and Teng [41]. A spectral sparsifier is a (rewighted) subgraph that has approximately the same Laplacian matrix as the original graph.

**Definition 2.1.** Let \( G = (V, E) \) be a graph with weights \( w_G : E \to \mathbb{R} \) and \( n = |V| \). We say that a subgraph \( H \subseteq G \) with weights \( w_H : E(H) \to \mathbb{R} \) is a \((1 \pm \epsilon)\)-spectral sparsifier for \( G \) if we have for all \( x \in \mathbb{R}^n \):

\[
(1 - \epsilon) x^T L_H x \leq x^T L_G x \leq (1 + \epsilon) x^T L_H x, \tag{1}
\]

where \( L_G \) and \( L_H \) are the Laplacians of \( G \) and \( H \) respectively.

We introduce the short-hand notation \( A \preceq B \) when \( B - A \) is positive semi-definite. This reduces equation 1 to \((1 - \epsilon)L_H \preceq L_G \preceq (1 + \epsilon)L_H\).

Koutis and Xu [27] showed how to compute a spectral sparsifier by repeatedly computing spanners. This technique was later slightly improved by Kyng et al. [28]. Spanners are a special type of spanning subgraphs, where we demand that distances are preserved up to a constant factor. Trivially, any graph is a spanner of itself. In practice, the goal will be to find sparse subgraphs that are still spanners for the input graph.

**Definition 2.2.** Let \( G = (V, E) \) be a graph with weights \( w : E \to \mathbb{R} \). We say that a subgraph \( S \subseteq G \) with weights \( w_S = w|_S \) is a spanner of stretch \( \alpha \) for \( G \) if for each \( u, v \in V \) we have

\[
d_S(u, v) \leq \alpha d_G(u, v),
\]

where we write \( d_H(u, v) \) for the distance from \( u \) to \( v \) in \( H \). A \( t \)-bundle spanner of stretch \( \alpha \) is a union \( T = \bigcup_{i=1}^t T_i \), where each \( T_i \) is a spanner of stretch \( \alpha \) in \( G \setminus \bigcup_{j=1}^{i-1} T_j \).

The algorithm of Koutis and Xu for computing spectral sparsifiers is relatively simple: compute a \( t \)-bundle spanner of stretch \( \alpha \), sample the remaining edges with probability \( 1/4 \), repeat for \([\log(m)]\) iterations on the computed bundle spanner and sampled edges. The sparsifier then consists of the last bundle spanner, together with the set of edges left after the \([\log(m)]\) iterations, where edges are reweighted in a certain manner. In the original algorithm, the stretch \( \alpha \) was fixed, but the number \( t \) of spanners in each bundle grew in each iteration. Kyng et al. [28] showed that \( t \) can be kept constant throughout the algorithm, leading to a sparser result.

**Algorithm 1: SpectralSparsifyOutline\((V, E, w, \epsilon)\)**

1. Set \( k := \lceil \log n \rceil \), \( t := 400 \log^2(n) \epsilon^{-2} \), and \( E_0 := E \).
2. For \( i = 1, \ldots, \lceil \log m \rceil \)
   3. Compute a \( t \)-bundle spanner \( B_i \) of stretch \( k \).
   4. \( E_i := B_i \).
   5. For each \( e \in E_{i-1} \setminus B_i \)
      6. \( \text{With probability } 1/4: E_i \leftarrow E_i \cup \{e\} \text{ and } w(e) \leftarrow 4w(e) \).
6. Return \( (V, E_{\lceil \log m \rceil}) \).

**2.3 Laplacian Solving**

We consider the following problem. Let \( L_G \) be the Laplacian matrix for some graph \( G \) on \( n \) vertices. Given \( b \in \mathbb{R}^n \), we want to solve \( L_G x = b \). Solving Laplacian equation exactly can be computationally-demanding. Therefore, we consider an approximation to this problem: we want to find \( y \in \mathbb{R}^n \) such that \( ||x - y||_{L_G} \leq \epsilon ||x||_{L_G} \), where we write \( ||x||_M := \sqrt{x^T M x} \) for any \( M \in \mathbb{R}^{n \times n} \). One way to approach this is by using a spectral sparsifier of \( G \). Hereto we use preconditioned Chebychev iteration, a well known technique from numerical analysis [4, 39]. The statement below most closely resembles the formulation of Peng [38].

**Theorem 2.3.** Suppose we have symmetric positive semi-definite matrices \( A, B \in \mathbb{R}^{n \times n} \), and a parameter \( \kappa \geq 1 \) satisfying

\[
A \preceq B \preceq \kappa A.
\]

Then there exists an algorithm that, given a vector \( b \in \mathbb{R}^n \) and parameter \( \epsilon \in (0, \frac{1}{2}] \), returns a vector \( y \in \mathbb{R}^n \) such that

\[
||x - y||_A \leq \epsilon ||x||_A,
\]

for some \( x \in \mathbb{R}^n \) satisfying \( Ax = b \). The algorithm takes \( O(\sqrt{\kappa} \log(1/\epsilon)) \) iterations, each consisting of multiplying \( A \) by a vector, solving of a linear system involving \( B \), and a constant number of vector operations.

This yields the following corollary for Laplacian solving using spectral sparsifiers.

**Corollary 2.4.** Let \( G \) be a weighted graph on \( n \) vertices, let \( \epsilon \in (0, \frac{1}{2}] \) be a parameter, and let \( b \in \mathbb{R}^n \) a vector. Suppose \( H \) is a \( (1 \pm \frac{1}{2}) \)-spectral sparsifier for \( G \). Then there exists an algorithm that outputs
a vector \( y \in \mathbb{R}^n \) such that \( ||x - y||_{L^2} \leq \epsilon ||x||_{L^2} \), for some \( x \in \mathbb{R}^n \) satisfying \( L \alpha x = b \). The algorithm takes \( O(\log(1/\epsilon)) \) iterations, each consisting of multiplying \( L \) by a vector, solving a Laplacian system involving \( L \gamma \), and a constant number of vector operations.

Proof. As \( H \) is a sparsifier for \( G \), we have: \( \left( 1 - \frac{1}{2} \right) L_H \leq L_G \leq \left( 1 + \frac{1}{2} \right) L_H \), which we can re-write to \( L_G \leq \left( 1 + \frac{1}{2} \right) L_H \leq \frac{1 + 1}{1 - 2} L_G \).

We set \( A := L_G \) and \( B := \left( 1 + \frac{1}{2} \right) L_H \), which are clearly both symmetric positive semi-definite. Furthermore, we set \( k := \frac{1 + 1}{1 - 2} = 3 \). We apply Theorem 2.3 with these settings to obtain the result. ❄️

### 2.4 Flow Problems

In this section we formally define the maximum flow and the minimum cost maximum flow problems. Let \( G = (V, E) \) be a directed graph, with capacities \( c : E \rightarrow \mathbb{Z}_{\geq 0} \), and designated source and target vertices \( s, t \in V \). We say \( f : E \rightarrow \mathbb{R}_{\geq 0} \) is an \( s-t \) flow if

1. For each vertex \( v \in V \setminus \{ s, t \} \) we have \( \sum_{e \in E : v \in e} f_e = 0 \);
2. For each edge \( e \in E \) we have \( f_e \leq c_e \).

The value of the flow \( f \) is defined as \( \sum_{(s, u) \in E} f(s, u) \). The maximum flow problem is to find a flow of maximum value. Additionally, we can have costs on the edges: \( q : E \rightarrow \mathbb{Z}_{\geq 0} \). The cost of the flow \( f \) is defined as \( \sum_{e \in E} q_e f_e \). The minimum cost maximum flow problem is to find a flow of minimum cost among all flows of maximum value.

Both problems allow for a natural linear program formulation. We present one for the minimum cost maximum flow problem, as this is the more general problem. Denote \( B \) for the edge-vertex incidence matrix (see Section 2.2). Then we can write this as:

\[
\min_{0 \leq x \leq e} q^T x \text{ such that } Bx = Fe_t - Fe_s,
\]

for \( F \) the value of the maximum flow, and \( e_s \) and \( e_t \) the vectors defined by \( (e_t)_v := \delta_{vt} \). The answer to the minimum cost maximum flow problem is then found by a binary search over \( F \).

### 3 Spectral Sparsifiers and Laplacian Solving

In this section, we show how to construct spectral sparsifiers in the Broadcast CONGEST model, so in particular also for the Broadcast Congested Clique. We do this following the method of Koutis and Xu [27], which consists of repeatedly computing sparsifiers and sampling the remaining vertices, see Section 2.2. While sampling edges is easy in the CONGEST model, it is highly non-trivial in the Broadcast CONGEST model. The reason for this is that in the CONGEST model the sampling of an edge can be done by one endpoint, and communicated to the other endpoint. In the Broadcast CONGEST model, the sampling can be done by one endpoint, but the result cannot be communicated efficiently to the other endpoint due to the broadcast constraint. To circumvent this, we show that the sampling needed for spectral sparsification can be done on the fly, rather than a priori in each iteration. Moreover, we show the result can be communicated implicitly. In Section 3.1, we show how to compute sparsifiers where we have probabilities on edges existing, whether an edge exists is evaluated on the fly and (implicitly) communicated to the other endpoint. In Section 3.2 we show how to use this spanner construction to compute spectral sparsifiers in the Broadcast CONGEST model.

#### 3.1 Spanners with Probabilistic Edges

Our goal is to compute a \((2k - 1)\)-spanner for a given probabilistic graph. More precisely, let \( G = (V, E, w) \) be an undirected, weighted graph on \( n \) vertices, with \( p : E \rightarrow [0, 1] \), \( e \mapsto p_e \) a probability function on the edges, and \( k \leq n \) the parameter for the stretch of the spanner. We will give an algorithm \( \text{Spanner}(V, E, w, p, k) \) that computes a subset \( F \subseteq E \), and divides this into two sets \( F = F^+ \sqcup F^- \), such that each edge \( e \in F \) is part of \( F^+ \) independently with probability \( p_e \). This results in a \((2k - 1)\)-spanner \( S = (V, F^+) \) for all graphs \((V, F^+, E')\), where \( E' \subseteq E \setminus F \). Since this is a distributed algorithm, the output comes in a local form. At the end, each vertex \( v \) has identified \( F^+_v \) and \( F^-_v \), where \( u \in F^+_v \iff (u, v) \in F^+ \).

When \( p \equiv 1 \), our algorithm essentially reduces to the algorithm of Baswana-Sen from [6]. All computational steps coincide, but a difference in communication remains. The reason hereto is that in our algorithm the weights of edges are included in the communication. Depending on the magnitude of the weights, this can result in multiple rounds for each message, and consequently more rounds in total.

We include Baswana and Sen’s algorithm in the full version of this paper. The general idea is that clusters are formed and revised through a number of phases. In each phase, a few of the existing clusters are sampled. These clusters move onto the next phase. Vertices from an unsampled cluster try to connect to a sampled cluster and to some neighboring clusters. As edges only exist with a certain probability, they need to be sampled before they can be used. We will make sure that the two vertices adjacent to an edge, never try to use it at the same time. When a vertex has tried to use an edge, the edge will always be broadcasted if it exists. If not, it turns out that the other vertex adjacent to this edge will be able to deduce this, without it being communicated explicitly.

For the algorithm see the full version, where we also prove the following lemma.

**Lemma 3.1.** The spanner \( S = (V, F^+) \) has stretch at most \( 2k - 1 \) for all graphs \((V, F^+ \cup E')\), where \( E' \subseteq E \setminus F \). For any choice of \( E \), it has at most \( |F^+| = O\left(k^{n+1/k}\right) \) edges in expectation. Moreover, we obtain an orientation on \( F^+ \) such that each edge has out-degree \( O(kn^{1/k}) \) in expectation.

We obtain the following running time of the algorithm, see the full version for a proof.

**Lemma 3.2.** The algorithm \( \text{Spanner}(V, E, w, p, k) \) takes

\[
O\left(kn^{1/k} \left(\log n + \log W\right)\right)
\]

rounds.

This directly gives us a \( t \)-bundle \( B \) of \((2k - 1)\)-spanners, where \( |B| = O(tkn^{1+1/k}) \). See the full version for the algorithm, which takes a total of \( O \left( tkn^{1/k} \left(\log n + \log W\right)\right) \) rounds.
3.2 Sparsification

The algorithm we give for spectral sparsification is based upon Algorithm 1, as given in Section 2.2. In the full version, we give a more concrete version of this algorithm, where we specify how to compute the bundle spanner. This algorithm repeatedly calculates a \( t \)-bundle spanner, and adds the remaining edges with probability \( 1/4 \). We amend this algorithm to be able to apply it in the Broadcast CONGEST model. The key difference is that whenever we need to keep edges with probability \( 1/4 \) we do this ad hoc and ‘locally’, rather than apriori and ‘central’.

Kyng et al. [28] have shown that the number \( t \) of spanners in each bundle can be kept the same throughout the algorithm, as opposed to increasing it in each iteration, which is done in the original algorithm of Koutis and Xu [27]. This results into a reduction of \( \log n \) in the size of the spanner. For our algorithm, Algorithm 2, we use the spanner construction from the previous section, which incorporates the ad hoc sampling with the spanner construction.

\[ \text{Algorithm 2: SpectralSparsify}(V, E, w, \epsilon) \]
\[
1. \text{Set } k := [\log n], \text{ } t := 400 \log^2 (n) e^{-2}, \text{ and } E_0 := E. \\
2. \text{Define } p: E \rightarrow [0, 1], \text{ by } p \equiv 1. \\
3. \text{for } i = 1, \ldots, [\log m] \text{ do} \\
   \qquad \text{Compute a } t \text{-bundle spanner } B_i \text{ of } (V, E_{i-1}), \text{ where edges exist with probability } p_{|E_{i-1}}. \\
   \qquad \text{and let } C_i \text{ denote the edges for which has been determined that they are not sampled.} \\
4. E_i \leftarrow E_{i-1} \setminus C_i. \\
5. \text{foreach } e \in B_i \text{ do} \\
6. \quad p(e) \leftarrow 1. \\
7. \text{foreach } e \in E_i \setminus B_i \text{ do} \\
8. \quad p(e) \leftarrow p(e)/4. \\
9. \quad w(e) \leftarrow 4w(e). \\
10. \text{Set } E' := B_{[\log m]}; \\
11. \text{At each vertex } v \in V: \\
12. \quad \text{foreach } (u, v) \in E_{[\log m]} \setminus E' \text{ do} \\
13. \quad \quad \text{if } \text{ID}(v) < \text{ID}(u) \text{ then} \\
14. \quad \quad \quad \text{With probability } p(u, v): \text{ add } (u, v) \text{ to } E' \text{ and broadcast } (u, v). \\
15. \text{return } (V, E'). \\
\]

Now from Lemma 3.3 and Theorem 3.4, we obtain the following result, the proof of which can be found in the full version.

**Theorem 1.2 (Restated).** There exists an algorithm that, given a graph \( G = (V, E, w) \) with positive real weights satisfying \( ||w||_{\infty} \leq U \) and an error parameter \( \epsilon > 0 \), with high probability outputs a \((1 \pm \epsilon)\)-spectral sparsifier \( H \) of \( G \), where \( |V| = O\left(\frac{n e^{-2} \log^4 n}{\epsilon^2}\right) \).

Moreover, we obtain an orientation on \( H \) such that with high probability each node has out-degree \( O(\log^4 (n)/\epsilon^2) \). The algorithm runs in \( O\left(\log^5 (n) e^{-2} \log(nU/\epsilon)\right) \) rounds in the Broadcast CONGEST model.

3.3 Laplacian Solving in the Broadcast Congested Clique

In this section, we restrict ourselves to the Broadcast Congested Clique: by assuming that communication between any two vertices is possible, we can make sure that in the end every vertex knows the entire sparsifier. Since the sparsification algorithm from Theorem 1.2 in fact gives us a way of orienting the edges of the sparsifier such that every vertex has maximum out-degree \( O(\log^4 (n)/\epsilon^2) \), it can become global knowledge in \( O(\log^4 (n)/\epsilon^2) \) rounds. However, each edge was explicitly added to the sparsifier in the algorithm, so when run in the BCC, at the end of the algorithm each vertex already knows the entire sparsifier. Now, we can use this spectral sparsifier for Laplacian solving, following Section 2.3.

**Theorem 1.3 (Restated).** There exists an algorithm in the Broadcast Congested Clique model that, given a graph \( G = (V, E, w) \), with positive real weights satisfying \( ||w||_{\infty} \leq U \) and Laplacian matrix \( L_G \), a parameter \( \epsilon \in (0, 1/2] \), and a vector \( b \in \mathbb{R}^n \), outputs a vector \( y \in \mathbb{R}^n \) such that \( ||x - y||_{L_G} \leq \epsilon ||x||_{L_G} \) for some \( x \in \mathbb{R}^n \) satisfying \( L_G x = b \). The algorithm needs \( O(\log^5 (n) \log(nU)) \) preprocessing rounds and takes \( O(\log(1/\epsilon) \log(nU/\epsilon)) \) rounds for each instance of \((b, \epsilon)\).

**Proof.** The algorithm satisfying these properties is as follows. In the preprocessing stage, we find a \((1 \pm 1/2)\)-spectral sparsifier \( H \) for \( G \) in \( O(\log^5 (n) \log(nU)) \) rounds, using SpectralSparsify\((V, E, w, 1/2)\). In this step, see also Theorem 1.2. At the end of this process, \( H \) is known to every vertex. Hence any computation with \( H \) can be done internally. Also note that multiplying \( L_G \) by a vector in the distributed setting only requires each vertex to know the vector values in neighboring vertices and the weights of the edges corresponding to those vertices. Communicating the vector values might need several broadcast rounds, depending on the number of bits necessary to represent the values. Since we aim for error \( \epsilon \), \( O(\log(nU/\epsilon)) \) bits suffice. Hence this takes at most \( O(\log(nU/\epsilon)) \) rounds. Now we apply Corollary 2.4 to find \( y \). This takes \( O(\log(1/\epsilon)) \) iterations of a solve in \( L_H \) (done internally at each vertex), a multiplication of \( L_H \) by a vector, and a constant number of vector operations (both done in \( O(\log(nU/\epsilon)) \)). Hence we have a total of \( O(\log(1/\epsilon) \log(nU/\epsilon)) \) rounds. \( \square \)

4 A LINEAR PROGRAM SOLVER

In this section, we show how to solve linear programs in the Broadcast Congested Clique. The linear programs we consider are distributed over the network in such a way that certain operations
with the constraint matrix $A$ are easy, these operations are matrix-vector multiplication and certain inversions.

Our linear program solver consists of an efficient implementation of Lee and Sidford [30, 31] in the Broadcast Congested Clique, which shows that one can obtain an $\epsilon$-approximation to a linear program using $O(\sqrt{\text{rank}} \log(1/\epsilon))$ linear system solves. Using our spectral sparsifier based Laplacian solver, we can solve certain linear systems up to the required precision in polylogarithmically many rounds, hence we obtain a $O(\sqrt{\text{rank}} \log(1/\epsilon))$ round algorithm for solving linear programs that give rise to the correct kind of linear system solves. One such example is in computing minimum cost flows, for which we show how to do this in Section 5.

To be precise, let $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n, c \in \mathbb{R}^m, l_i \in \mathbb{R} \cup \{-\infty\}$, and $u_i \in \mathbb{R} \cup \{+\infty\}$ for all $i \in [m]$. The goal is to solve linear programs in the following form

$$\text{OPT} := \min_{x \in \mathbb{R}^n : Ax = b, l_i \leq x_i \leq u_i} c^T x.$$  

We assume that the interior polytope $\Omega^0 := \{x \in \mathbb{R}^n : A^T x = b, l_i \leq x_i \leq u_i\}$ is non-empty and $\text{dom}(x_i) := \{x : l_i < x_i < u_i\}$ is never the entire real line, i.e., either $l_i \neq -\infty$ or $u_i \neq +\infty$. We then obtain the following theorem.

**Theorem 1.4 (Restated).** Let $A \in \mathbb{R}^{m \times n}$ be a constraint matrix with $\text{rank}(A) = n$, let $b \in \mathbb{R}^n$ be a demand vector, and let $c \in \mathbb{R}^m$ be a cost vector. Moreover, let $x_0$ be a given initial point in the feasible region $\Omega^0 := \{x \in \mathbb{R}^n : A^T x = b, l_i \leq x_i \leq u_i\}$. Suppose a Broadcast Congested Clique network consists of $n$ vertices, where each vertex $i$ knows both the entire $j$-th row of $A$ for which $A_{ji} \neq 0$ and knows $(x_0)_j$ if $A_{ji} \neq 0$. Moreover, suppose that for every $y \in \mathbb{R}^n$ and positive diagonal $D \in \mathbb{R}^{m \times m}$ we can compute $(A^T DA)^{-1} y$ up to precision $\text{poly}(1/m)$ in $T(n, m)$ rounds. Let $U := \max\{||1/(u-x_0)||_\infty, ||1/(x_0-I)||_\infty, ||u-I||_\infty, ||c||_\infty\}$. Then with high probability the Broadcast Congested Clique algorithm $LPSolve$ outputs a vector $x \in \Omega^0$ with $c^T x \leq \text{OPT} + \epsilon$ in $O(\sqrt{n}\log(U/(\epsilon))(\log^2(U/\epsilon) + T(n, m)))$ rounds.

As mentioned, the algorithm fulfilling this theorem is an efficient implementation of the LP solver of Lee and Sidford [30, 31] in the Broadcast Congested Clique. Therefore, we refer to [31] for a proof of correctness. We will show how to implement each step in the Broadcast Congested Clique, and bound the running time. Using our Laplacian solver, we can show that we can run most of the Lee-Sidford algorithm directly in the Broadcast Congested Clique. However, there are two subroutines that need adjustment. We need to compute the approximate leverage scores differently (see Section 4.1), and we need to adjust the routine for projections on a mixed norm ball (See Section 4.2). For the main algorithm that uses both of these as subroutines, we refer to the full version.

The idea of the algorithm is to use weighted path finding, which is a weighted variant of the standard logarithmic barrier function. In particular we follow a central path reweighted by the $\ell_p$ Lewis weights. This means that the barrier function is multiplied by the Lewis weight of the current point. Now each step of the interior point method consists of taking a Newton step and recomputing the weights.

Throughout this section, we will simplify to the situation where $(A^T DA)^{-1}y$ is solved exactly, rather than to precision $\text{poly}(1/m)$. The fact that $\text{poly}(1/m)$ precision suffices is proved by Lee and Sidford in [29]. Throughout the algorithm, vectors will be stored in the natural manner: for $y \in \mathbb{R}^n$ vertex $i$ stores $y_i$, and for $y \in \mathbb{R}^m$ vertex $i$ knows $y_j$ if $A_{ji} \neq 0$. Together with our assumptions on which vertex knows which part of $A$, this means we can perform matrix-vector efficiently. Since we can assume weights and vector values to be at most $O(\text{poly}(n, m) U/\epsilon)$, we obtain this in $O(\log(U/\epsilon))$ rounds.

### 4.1 Weight Functions and Leverage Scores

For details on the weighted path finding procedure and the used barrier functions, see the full version. In this paper, we use the regularized Lewis weights.

**Definition 4.1.** For $M \in \mathbb{R}^{m \times n}$ with rank$(M) = n$, we let $\sigma(M) := \text{diag}(M^T M)^{-1} M^T$ denote the leverage scores of $M$. For all $p > 0$, we define the $\ell_p$-Lewis weights $w_p(M)$ as the unique vector $w \in \mathbb{R}^m$ such that $w = \sigma(W^{-\frac{1}{2}} M)$, where $w = \text{diag}(W)$. We define the regularized Lewis weights as $g(x) := w_p(Mx) + c_0$, for $p = 1 - \frac{1}{\log(4m)}$ and $c_0 := \frac{1}{2m}$.

In the full version, we show how to compute these weights efficiently, given an algorithm that computes the leverage score. So let us start with the leverage scores. Computing $M(M^T M)^{-1} M^T$ to determine its diagonal is expensive, as are any other known techniques of computing $\sigma(M)$ exactly. However, approximating them is significantly less computationally heavy. As shown in [16, 40], we can reduce the computation to solving a polylogarithmic number of regression problems (see also [13, 32, 35, 45]). Namely by noting that $\sigma(M) = ||M(M^T M)^{-1} M^T||_2^2$, and using the Johnson-Lindenstrauss lemma, which states that this norm is preserved approximately under projections onto certain low dimensional subspaces. In previous work [31, 40], this was done by randomly sampling subspaces. A common approach is to sample polylogarithmically many vectors in $\mathbb{R}^m$ according to some (simple) distribution. However, this is problematic for the Broadcast Congested Clique: it is unclear how one vertex can sample the value of an edge and efficiently communicate this to its corresponding neighbor. Therefore, we use a different variant of the Johnson-Lindenstrauss lemma, by Kane and Nelson [26], that requires significantly fewer random bits.

**Theorem 4.2 ([26]).** For any integer $m > 0$, and any $\eta > 0, \delta < 1/2$, there exists a family $Q$ of $\kappa \times m$ matrices for $\kappa = \Theta(\eta^{-2} \log(1/\delta))$ such that for any $x \in \mathbb{R}^m$,

$$\mathbb{P}_{Q \in Q}[(1 - \eta) ||x||_2 \leq ||Qx||_2 \leq (1 + \eta) ||x||_2] \geq 1 - \delta,$$

where $Q \in Q$ can be sampled with $O(\log(1/\delta) \log(m))$ uniform random bits.

Algorithm 3 uses this theorem to compute $\sigma(\text{apx})$ such that $(1 - \eta) \sigma(M)_i \leq \sigma(\text{apx})_i \leq (1 + \eta) \sigma(M)_i$, for all $i \in [m]$.

**Lemma 4.3.** For any $\eta > 0$, with probability at least $1 - 1/m^{O(1)}$ the algorithm $\text{ComputeLeverageScores}(M, \eta)$ computes $\sigma(\text{apx})(M)$ such that

$$(1 - \eta) \sigma(M)_i \leq \sigma(\text{apx})(M)_i \leq (1 + \eta) \sigma(M)_i.$$
Algorithm 3: ComputeLeverageScores($M, \eta$)

1. Set $k = \Theta(\log(m)/\eta^2)$.
2. Broadcast vertex IDs to determine the vertex with the highest ID; declare this vertex the leader.
3. The leader samples $\Theta(\log^2(m))$ random bits and broadcasts them.
4. Each vertex constructs $Q \in \mathbb{R}^{k \times m}$ from Theorem 4.2 internally, using the random bits sampled by the leader.
5. Compute $p(j) = M(M^T M)^{-1} M^T Q(j)$.
6. return $\sum_{j=1}^{k} \left( p(j) \right)^2$.

for all $i \in [m]$. If $M = WA$, for some diagonal $W \in \mathbb{R}^{m \times m}$, it runs in $\tilde{O}(\eta^{-2}(\log(u/e) + T(n,m)))$ rounds.

Proof. Note that for any $k \times m$ matrix $Q$ and symmetric $m \times m$ matrix $X$ we have

$$||QXe_i||_2 = \sum_{j=1}^{k} (QX)^2_{ji} = \sum_{j=1}^{k} \left( XQ(j)_i \right)^2.$$ 

Note that $M(M^T M)^{-1} M^T$ is a symmetric $m \times m$ matrix and

$$\sigma_{\text{apx}}(M)_i = \left| |QM(M^T M)^{-1} M^T e_i||_2 \right|.$$ 

hence

$$\sigma_{\text{apx}}(M) = \sum_{i=1}^{k} \left( (M(M^T M)^{-1} M^T Q(j)) \right)^2.$$ 

By Theorem 4.2, we have

$$(1-\eta)||M(M^T M)^{-1} M^T e_i||_2 \leq ||QM(M^T M)^{-1} M^T e_i||_2 \leq (1+\eta)||M(M^T M)^{-1} M^T e_i||_2$$

with probability at least $1 - 1/mO(1)$ for our random $Q \in Q \subseteq \mathbb{R}^{k \times m}$ with $k = \Theta(\tilde{\eta}^{-2}\log(m))$, constructed from the $\Theta(\log(m)^2)$ random bits sampled by the leader. Using that

$$\sigma_{\text{apx}}(M)_i = ||QM(M^T M)^{-1} M^T e_i||_2^2$$ 

and $\sigma(M)_i = ||M(M^T M)^{-1} M^T e_i||_2^2$, we obtain

$$(1-\eta)^2 \sigma(M)_i \leq \sigma_{\text{apx}}(M)_i \leq (1+\eta)^2 \sigma(M)_i.$$ 

Now setting $\tilde{\eta} = \eta/4$ gives $1 - \eta \leq (1-\tilde{\eta})^2$ and $(1+\tilde{\eta})^2 \leq 1 + \eta$, hence we obtain

$$(1-\eta)\sigma(M)_i \leq \sigma_{\text{apx}}(M)_i \leq (1+\eta)\sigma(M)_i.$$ 

This means that we have $k = \Theta(\log(m)/\tilde{\eta}^2) = \Theta(\log(m)/\eta^2)$.

For the running time, note that for $j = 1, \ldots, k$ we need to multiply $M^T$ by a vector, solve a linear system in $M^T M$, and multiply $M$ by a vector. Since $M = WA$, each of these steps can be done in either $\tilde{O}(\eta^{-2}(\log(U/e)))$ or $T(n,m)$ rounds by assumption, giving a total running time of $\tilde{O}(k(\log(U/e) + T(n,m))) = \tilde{O}(\eta^{-2}(\log(U/e) + T(n,m)))$ rounds.

Note that this algorithm is randomized. It is actually the only randomized part of the linear program solver itself and the bottleneck for making it deterministic. However, in the Broadcast Congested Clique our Laplacian solver (Theorem 1.3) is also randomized. On top of that, the algorithm for computing minimum cost maximum flow of Section 1.1 has an auxiliary randomized component.

4.2 Projection on Mixed Norm Ball in Broadcast Congested Clique

In this section, we show how we solve the following problem in the Broadcast Congested Clique. Let $a, l \in \mathbb{R}^m$, the goal is to find

$$\arg \max a^T x, ||x||_2 \leq \eta$$

Here, Lee and Sidford [31] initially sort $m$ values and compute $m$ functions naively to achieve $\tilde{O}(m \log^2(U/e))$ rounds. We overcome this issue by only sorting implicitly, and doing a binary search, such that we only have to compute logarithmically many functions.

Lemma 4.4. Suppose the vectors $a, l \in \mathbb{R}^m$ are distributed over the network such that: 1) for each $i \in [m]$, $a_i$ and $l_i$ are known by exactly one vertex, 2) a vertex knows $a_i$ if and only if it knows $l_i$. Moreover, suppose that $||a||_{\infty}, ||l||_{\infty} \leq O(\text{poly}(m))$. Then there exists an algorithm that finds

$$\arg \max a^T x, \left| x \right|_{\infty} \leq \eta$$

up to precision $O(1/(\text{poly}(m)\epsilon))$ in $\tilde{O}(\log^2(U/e))$ rounds in the Broadcast Congested Clique.

For the proof of this lemma, see the full version. When we apply this lemma in our LP solver, multiple vertices will know the same values $a_i$ and $l_i$, however, each vertex will know which other vertices know $a_i$ and $l_i$, hence we can simply allocate the values $a_i$ and $l_i$ to the vertex with the highest ID.

5 MINIMUM COST MAXIMUM FLOW

In this section, we apply the linear program solver of the previous section to the minimum cost maximum flow problem. This problem is defined as follows. Let $G = (V, E)$ be a directed connected graph, with capacities $c : E \rightarrow \mathbb{Z}_{\geq 0}$ and costs $q : E \rightarrow \mathbb{Z}^+$. The goal is to compute a maximum flow of minimal cost, see Section 2.4. In this section, we prove the following result.

Theorem 1.1 (Restated). There exists a Broadcast Congested Clique algorithm that, given a directed graph $G = (V, E)$ with integral costs $q \in \mathbb{Z}^{|E|}$ and capacities $c \in \mathbb{Z}^{|E|}_{\geq 0}$, with $||q||_{\infty} \leq M$ and $||c||_{\infty} \leq M$, computes a minimum cost maximum $s$-$t$ flow with high probability in $\tilde{O}(\sqrt{|V|\log^3 M})$ rounds.

To prove this, we have to show that the minimum cost maximum flow problem satisfies the conditions of Theorem 1.4. Clearly, the linear program as presented in Section 2.4 satisfies this. However, this would incur two problems. The first is that the LP solver computes an approximate solution. It is not clear how to efficiently transform this into an exact solution. The second problem is that we need an auxiliary binary search to find the maximum flow. Both

In this section, we will write $|V|$ and $|E|$, so that we can reserve $n$ and $m$ for the dimensions of the linear program.
problems are solved simultaneously by considering a closely related LP, see Daitch and Spielman [14] and Lee and Sidford [31].

We let $B \in \mathbb{R}^{|E| \times |V|}$ be the edge-vertex incidence matrix where we omit the row for the source $s$. We let our variables consist of $x \in \mathbb{R}^{|E|}$, $y, z \in \mathbb{R}^{|V|}$ and $F \in \mathbb{R}$. We define the linear program as follows.

$$\min \ 3\tilde{T}^T x + \lambda (1T y + 1T z) - 2n\tilde{M}F$$
subject to $Bx + y - z = Fe_t,$
$$0 \leq x_i \leq c_i,$$
$$0 \leq y_i \leq 4|V|M,$$
$$0 \leq z_i \leq 4|V|M,$$
$$0 \leq F \leq 2|V|M,$$

where $\tilde{M} := 8|E|^2M^3$, $\lambda := 440|E|^4\tilde{M}^2M^3$, and $\tilde{q}$, satisfying $\tilde{q} \leq \tilde{M}$, is defined as follows. For every edge, take a uniformly random number from $[\frac{1}{4|E|^2M^3}, \frac{2}{4|E|^2M^3}, \ldots, \frac{2|E|M}{4|E|^2M^3}]$, and add this to $q_e$. With probability at least $1/2$, the problem with this cost vector has a unique solution, and this solution is also a valid solution for the original problem [14]. We apply this reduction and scale the problem such that the cost vector is integral again.

It is easy to check that the following is an interior point: $F = |V|M, x = \frac{y}{2}, y = 2|V|M - (Bx)^- + F e_t, z = 2|V|M - (Bx)^+$, where we denote $a^+$ and $a^-$ for the vectors defined by

$$(a^+)_i = \begin{cases} a_i & \text{if } a_i \geq 0; \\ 0 & \text{else.} \end{cases} \quad \text{and} \quad (a^-)_i = \begin{cases} a_i & \text{if } a_i \leq 0; \\ 0 & \text{else.} \end{cases}$$

respectively.

A solution to this linear program can be transformed to a solution to the minimum cost maximum flow problem. To be precise, one can find an exact solution to the minimum cost maximum s-t flow problem, if we can find a feasible solution to the above LP with cost value within $\frac{1}{1+\epsilon}$ of the optimum. This solution $x$ is then transformed in two steps: it is made a feasible flow $\tilde{x}$ for the original graph by subtracting the error we may have created by introducing additional variables $y$ and $z$. One can show this is at most $\epsilon := \frac{1}{40|E|M^2}$, since the LP is solved up to precision $\frac{1}{1+\epsilon}$ [31]. We set $\tilde{x} := (1 - \epsilon)x$. This is not yet optimal, but by integrality of costs and the fact that the min-cost solution is unique, we have that the flow $\tilde{x}$ on each edge is at most $1/6$ off from the optimal value [31]. We obtain the optimal value simply by rounding $\tilde{x}$ to the closest integer. In the Broadcast Congested Clique, multiplication by $(1 - \epsilon)$ and rounding can be done internally, so this requires no rounds.

Next, we show how to actually solve the above LP. We set $A := \{B I - I - e_t\}^T$, and use the LP solver of Section 4 with this constraint matrix. To be precise, we run the algorithm on a network of $n = |V| - 1$ vertices, as $s$ does not need to participate. It is clear the knowledge of $A$ is distributed in the required manner, as local knowledge of the edge-vertex incidence matrix $B$ is known by default.

The lemma below states that we can solve linear equations in $A^T DA$ in $\tilde{O}(\log(M))$ rounds (for a proof see the full version). Then Theorem 1.4 solves the LP in $\tilde{O}(\sqrt{n}\log^3 M)$ rounds, where we use that $T(n, m) = \tilde{O}(\log(M))$.

**Lemma 5.1.** Let $D \in \mathbb{R}^{|E|+|V|} \times (|E|+|V|)$ be any positive diagonal matrix, then there is a BCC algorithm that solves linear equations in $A^T DA$ up to precision $1/m^{O(1)}$ in $\tilde{O}(\log(M))$ rounds.

### 6 CONCLUSION

As explained in this paper, the algorithm of Lee and Sidford is based on an interior-point method that (1) performs $\tilde{O}(\sqrt{n})$ iterations and (2) spends $\tilde{O}(m)$ operations per iteration using primitives like matrix-vector multiplication and solving a Laplacian system. Recent advances show how to (slightly) improve upon the $\tilde{O}(m\sqrt{n})$ bound by employing sophisticated dynamic data structures to decrease the (amortized) number of operations spent per iteration [5, 44]. In an even more recent breakthrough [12], the minimum cost flow problem has been solved in $O(m^{2+\epsilon}(1) time in the centralized model. This is done by an interior point method that (1) performs $O(m^{2+\epsilon}(1))$ iterations and (2) spends $O(m^{o(1)})$ time per operation. One might wonder whether alternatively it would also be possible to improve upon the $\tilde{O}(m\sqrt{n})$ bound purely by finding an interior-point method with a reduced number of iterations.

In this paper we have demonstrated how the primitives employed in each iteration can be carried out efficiently in the Broadcast Congested Clique and thus provide evidence that improvements in the iteration count of the interior-point method would likely carry over to the round complexity in the Broadcast Congested Clique – currently this leads to an algorithm with $\tilde{O}(\sqrt{n})$ rounds. This motivates the question of whether a lower bound for the min-cost flow problem can be obtained in this model since such a lower bound would rule out the possibility of an improvement of the iteration count using the same fast primitives. Note that in contrast to the (Unicast) Congested Clique – for which lower bounds would yield a breakthrough in circuit complexity [18] – the Broadcast Congested Clique seems much more amenable to lower bounds; in particular polynomial lower bounds exist for several graph problems in the broadcast model [7, 10, 18, 21, 24].

### ACKNOWLEDGMENTS

This work is supported by the Austrian Science Fund (FWF): P 32863-N. This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 947702).

### REFERENCES

[1] Dimitris Achlioptas. 2003. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of computer and System Sciences* 66, 4 (2003), 671–687. https://doi.org/10.1016/S0022-0000(03)00025-4 Announced at PODS 2001.

[2] Ioannis Anagnostides, Themis Gouleakis, and Christoph Lenzen. 2021. Accelerated Distributed Laplacian Solvers via Shortcuts. *CoRR* abs/2109.05151 (2021). arXiv:2109.05151

[3] Simon Apers and Ronald de Wolf. 2020. Quantum Speedup for Graph Sparsification, Cut Approximation and Laplacian Solving. *In Proc. of the 61st IEEE
