Estimating Current-Flow Closeness Centrality with a Multigrid Laplacian Solver *

Elisabetta Bergamini† Michael Wegner* Dimitar Lukarski‡ Henning Meyerhenke*

Abstract

Matrices associated with graphs, such as the Laplacian, lead to numerous interesting graph problems expressed as linear systems. One field where Laplacian linear systems play a role is network analysis, e. g. for certain centrality measures that indicate if a node (or an edge) is important in the network. One such centrality measure is current-flow closeness.

To allow network analysis workflows to profit from a fast Laplacian solver, we provide an implementation of the LAMG multigrid solver in the NetworKit package, facilitating the computation of current-flow closeness values or related quantities. Our main contribution consists of two algorithms that accelerate the current-flow computation for one node or a reasonably small node subset significantly. One algorithm is an unbiased estimator using sampling, the other one is based on the Johnson-Lindenstrauss transform. Our inexact algorithms lead to very accurate results in practice. Thanks to them one is now able to compute an estimation of current-flow closeness of one node on networks with tens of millions of nodes and edges within seconds or a few minutes. From a network analytical point of view, our experiments indicate that current-flow closeness can discriminate among different nodes significantly better than traditional shortest-path closeness and is also considerably more resistant to noise – we thus show that two known drawbacks of shortest-path closeness are alleviated by the current-flow variant.

*This work is partially supported by DFG grant ME-3619/3-1 (FINCA) within the SPP 1736 *Algorithms for Big Data*

†Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

‡Paralution Labs UG & Co. KG, Gaggenau, Germany

1 Introduction

Laplacian linear systems $Lp = b$ play a central role in various tasks in algorithmic graph theory and network analysis. Particularly the connection between the graph Laplacian L and electrical networks allows analytical and algorithmic insights [20, 5]. Based on this connection, several centrality measures, which aim at identifying the most important nodes or edges in a network, have been introduced in the literature. For example, spanning edge centrality [20] measures the importance of a node for the connectedness of the graph. Current-flow betweenness [5] indicates the participation of a node in paths between other nodes and current-flow closeness [5] measures the average distance from a node to the other nodes of the network. Differently from shortest-path closeness, here distance is a quantity which takes *all* paths between two nodes into account. In addition to centrality measures, Laplacian linear systems can be used for several other tasks in algorithmic graph theory and algorithmic network analysis, including sparsification [24], graph partitioning [21], approximate maximum network flow [6] and graph drawing [11].

Although algorithms for solving Laplacian linear systems quickly in practice have been proposed, such as the multigrid-based solvers LAMG [19] and CMG [17], an implementation of these solvers is not available in popular network-analysis frameworks, such as NetworkX [14], igraph [7], graph-tool [22] and NetworKit [26]. In this paper, we implement LAMG in NetworKit, bridging the gap between performance-focused frameworks for network analysis and state-of-the-art Laplacian solvers. Also, we formulate the current-flow closeness centrality presented in [5] in terms of linear systems and propose two algorithms for approximating it. According

to our experimental study, our approximations are extremely accurate and are reasonably fast in scenarios where the centrality of a single node or a subset of nodes has to be computed. To the best of our knowledge, only the NetworkX [14] package contains an implementation of current-flow closeness. The approach implemented in NetworkX requires to invert the Laplacian of the graph, which takes $\Omega(n^2)$ time, since the inverse of the Laplacian is in general a dense matrix. This does not allow to scale to large networks with millions of nodes and edges, even when we want to compute the closeness of a single node or of a subset of nodes. With this approach, in fact, computing the closeness of one node is just as expensive as computing it for all nodes. On the contrary, our approach can estimate the closeness of a single node very quickly: For example, it requires less than 2 minutes on a network with 50 millions edges.

Thanks to our approach, we can study for the first time the properties of current-flow closeness in large networks with tens of millions of nodes. We compare current-flow closeness with traditional shortest-path closeness and show that the former succeeds in differentiating nodes significantly better than the latter, and is also more resilient to noise. In addition, we study the correlation between centrality measures and degrees in real-world networks, in relation to a recent theoretical result for random geometric graphs [27]. Our experiments show that there is a strong correlation between degrees and current-flow closeness in complex networks, whereas there is basically no correlation in street networks.

2 Preliminaries

Graph basics. Throughout the paper we consider connected undirected graphs $G = (V, E, w)$ having $n = |V|$ nodes and $m = |E|$ edges (disconnected graphs can be handled by treating each connected component separately). The function $w : E \to \mathbb{R}$ assigns a weight $w(e)$ to each edge $e = \{u, v\}$ where u and v are the nodes connected by the edge. We use ω_{uv} to denote the weight $\omega(e)$ of an edge $e = \{u, v\}$. We denote by $\vec{E}$ the set of bidirected edges of G, i.e. $\vec{E} := \{(u, v) : u, v \in V, \{u, v\} \in E\}$. We say $\pi = (v_1, ..., v_k)$ is a *path* in G if $v_i \in V$ for $i = 1, ..., k$ and $\{v_i, v_{i+1}\} \in E$ for $i = 1, ..., k-1$. We denote the quantity $\sum_{i=1}^{k-1} \omega_{v_i, v_{i+1}}$ as the *length* of path π. The path(s) of minimum length among all paths between two nodes u and v is (are) called the *shortest path(s)* between u and v. A Laplacian matrix L is defined for an undirected graph G as $L := D - A$, where $A = A(G)$ is the (weighted) adjacency matrix of G and $D = D(G)$ the diagonal matrix storing the (weighted) node degrees: $D_{ii} = \sum_{j=1}^{n} \omega_{ij}$.

Graphs as electrical networks. One can regard a graph as an *electrical network* where each edge $\{u, v\}$ corresponds to a resistor with conductance ω_{uv} (the edge weight) or resistance $1/\omega_{uv}$. We can interpret the conductance as the ease with which an electrical current can flow through the edge. We can associate a *supply* $b : V \to \mathbb{R}$ with the electrical network, representing the nodes where current enters and leaves the network. A positive supply $b(v)$ means that current is entering the network from node v and a negative supply means that current is leaving the network. In the following, we will always assume that $\sum_{v \in V} b(v) = 0$ and that $b(s) = +1$ and $b(t) = -1$ for two nodes s and t, and that $b(w) = 0 \ \forall w \neq s, t$. In the following, we will refer to such a supply as vector $b_{st} \in \mathbb{R}^{n \times 1}$. We could interpret this as s and t being the two poles of a battery: this generates a current $e_{st} : \vec{E} \to \mathbb{R}$ flowing through the network. To each node v we can associate a *potential* $p_{st}(v)$ such that the vector $p_{st} \in \mathbb{R}^{n \times 1}$ satisfies the following linear system:

$$Lp_{st} = b_{st} \tag{2.1}$$

Then, the current flowing through edge (u, v) is defined as $(p_{st}(u) - p_{st}(v))/\omega_{st}$. Notice that, since G is connected, the rank of the Laplacian is $n - 1$ and there are infinitely many vectors p_{st} satisfying Eq. (2.1), each of them differing from the other by an additive constant. However, the current is well defined, since it depends on the difference between two potentials.

3 Related Work

3.1 Solving Laplacian linear systems. We focus our description on iterative solvers due to their better time complexity on sparse graphs compared to direct solvers. Most iterative solvers reduce the norm of the residual $r = \|b - Ax\|$ iteratively by

altering the current preliminary solution vector x in every iteration. One usually stops when the (relative) residual is below a certain tolerance τ, which yields a vector x' that is a good enough approximation to the actual solution x. While there are recent advances in theory to solve special linear systems including Laplacians in nearly-linear time [25, 16], those algorithms are not competitive in practice yet [15, 4]. In fact, the Conjugate Gradient (CG) algorithm outperforms the nearly-linear time algorithms in practice even though its asymptotic running time is typically higher.

A popular class of iterative algorithms to solve linear equations quickly in practice is called Algebraic Multigrid (AMG) [23]. The basic idea is to solve the actual linear system by iteratively solving coarser (i.e. smaller) yet similar systems and projecting the solutions of those back to the original system. AMG algorithms can be distinguished by the class of matrices they can handle and the way they construct the coarser systems. Two fast algorithms that are specifically designed for solving Laplacian systems are CMG by Koutis et al. [17] and LAMG by Livne and Brandt [19].

We decided to use LAMG as linear solver due to its particular design for complex networks. To this end, LAMG alternates between two stages called *elimination* and *aggregation* to construct the coarser systems. The former eliminates low degree nodes in the corresponding graph, the latter partitions nodes into *aggregates* based on a special affinity measure [19]. Both stages reduce the number of nodes and thus define a coarsening mechanism. Based on an extensive evaluation, Livne and Brandt state that running times of LAMG and CMG are comparable but LAMG tends to be more robust in the sense that CMG has large outliers on a small set of systems [19].

3.2 Laplacian linear systems for network analysis. The connection between the graph Laplacian and electrical networks (see Section 2) has allowed for the solution of several graph algorithmic problems in terms of Laplacian systems. One of them is a centrality measure called *spanning edge centrality*, which indicates whether an edge is vital for the connectedness of a network [20]. The notion of importance for connectedness is also helpful for graph sparsification. A sparsification algorithm takes a dense graph and wants to find a sparser representation (= with fewer edges) with the same vertex set and similar properties [24], e. g. approximately the same cut sizes or eigenvalues. Since processes described by Laplacian linear systems can distinguish sparse from dense graph regions, edges in dense areas are, intuitively speaking, redundant and can be "sparsified" without doing much harm to the cut sizes when the weights of retained edges are properly scaled. This idiosyncrasy allows the use of processes described by Laplacian linear systems also for graph partitioning [21], approximate maximum network flow [6], and graph drawing [11]. Moreover, the connection to electrical flow makes the use in dynamic load balancing of divisible tokens by diffusion [9] possible.

The interpretation of a graph as an electrical network has also led to the definition of two centrality measures based on current flow, current-flow closeness and current-flow betweenness [5]. Compared to traditional closeness and betweenness centrality, these two measures take *all* paths between two nodes into account and not only *shortest* paths.

4 Current-flow closeness centrality

Closeness centrality measures the efficiency of a node in spreading information to the other nodes of the network. Formally, let $d_{\mathrm{SP}}(u, v)$ the shortest-path distance between u and v (i.e. the length of the shortest path(s) between u and v). Then, closeness of node v is defined as the inverse of the expected shortest-path distance between v and and a random node w:

$$c_{\mathrm{SP}}(v) := \frac{n-1}{\sum_{w \neq v} d_{\mathrm{SP}}(v, w)}. \tag{4.2}$$

The smaller the average distance between v and the other nodes, the higher is the closeness of v. To better understand the meaning of closeness, let us consider the two graphs in Figure 1. Since closeness takes only shortest-path distances into account, the closeness of node x_1 in the graph on the left and the score of node x_2 in the graph on the right will be exactly the same. However, there is only one path connecting x_1 to each of the other nodes. This

means that if just a single edge is removed from the graph, x_1 will become disconnected from part of the other nodes. For example, let us assume the edges represent streets and x_1 is the location of an ambulance. If a congestion occurs, the ambulance in x_1 will not be able to reach part of the nodes (or it will take a very long time to reach them). On the other hand, if the ambulance was in x_2, a congestion would limit only partially (or not at all) the ability of the ambulance to reach the nodes of the network.

The example above illustrates that traditional closeness is unable to model scenarios where the distance between two nodes does not only depend on the length of the shortest path between them, but also on the number of shortest or relatively short paths between the nodes. For this reason, a variant of closeness named *current-flow closeness centrality* has been introduced [5]. If we see the graph as an electrical network, the *effective resistance* $d_{\mathrm{ER}}(u,v) := p_{uv}(u) - p_{uv}(v)$ can be interpreted as an alternative distance measure between nodes u and v. Indeed, if we multiply $d_{\mathrm{ER}}(u,v)$ by the volume of G (i.e. the sum of the weights of the edges in G), we get the *commute time* between u and v. The commute time between nodes u and v is defined as $H(u,v) + H(v,u)$, where the hitting time $H(x,y)$ is the expected time step in which a random walk in the graph starting in x reaches y for the first time. Thus, the commute time can be seen as the expected time a random walk needs for going from u to v and back again. Since the commute time is based on random walks, it depends on *all* the paths between two nodes. Thus, we can use the effective resistance to define a modified centrality measure [5]:

(4.3)
$$c_{\mathrm{ER}}(v) := \frac{n-1}{\sum_{w \neq v} d_{\mathrm{ER}}(v,w)} = \frac{n-1}{\sum_{w \neq v} p_{vw}(v) - p_{vw}(w)}.$$

By convention, we define $d_{\mathrm{ER}}(v,v) := 0 \ \forall v \in V$. To compute $c_{\mathrm{ER}}(v)$ of a node v, we can solve $n-1$ linear systems. Alternatively, we could invert the Laplacian matrix L of G (after omitting the row and column corresponding to a node, in order to get a regular matrix $\tilde{L}$), using the property that $p_{vw}(v) - p_{vw}(w) = \tilde{L}^{-1}_{vv} - 2\tilde{L}^{-1}_{vw} + \tilde{L}^{-1}_{ww}$ [5].

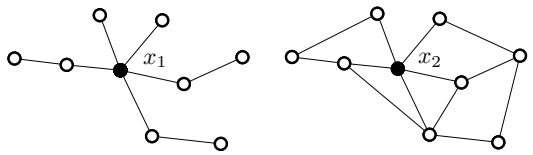

Figure 1: Shortest-path closeness centrality cannot distinguish between node x_1 and node x_2.

5 Approximating current-flow closeness

As outlined in [20], the fastest Laplacian linear solvers with a theoretical time complexity guarantee run in $O(m \log n \log(1/\tau))$ time, where τ is the tolerance of the solver. Multigrid methods such as CMG and LAMG are much faster in practice and have an *empirical* running time of $O(m \log(1/\tau))$.

Computing current-flow closeness for only one node to the desired tolerance would already require the solution of $n-1$ linear systems, yielding $O(n^2 \log(1/\tau))$ time in practice assuming a sparse graph. This is infeasible for large networks with millions of nodes and edges. For this reason, we propose two approximation techniques for computing current-flow closeness for a subset of the nodes in large graphs, and we compare them in our experimental evaluation. The first one is based on a simple sampling approach, which recalls the one used for classical closeness [10]. The second one uses the Johnson-Lindenstrauss transform (JLT), which allows to project the system into a lower-dimensional space by using $\mathcal{O}(\log n)$ random vectors. The two approximations are different in nature and we were able to prove a theoretical guarantee on the quality of the approximation only for the second one. However, our experiments in Section 6 show that both approaches work very well in practice.

5.1 Sampling-based approximation. The idea is to sample uniformly at random a set $S \subseteq V$ of nodes $S = \{s_1, ..., s_k\}$, which we call *pivots*. To approximate the current-flow closeness of a node v, we compute the effective resistance $d_{\mathrm{ER}}(s,v)$ between all nodes $s \in S$ and v. Then, the closeness of v can be approximated as

$$\tilde{c}_{\mathrm{ER}}(v) := \frac{k}{n} \cdot \frac{n-1}{\sum_{i=1}^{k} d_{\mathrm{ER}}(v, s_i)}.$$

PROPOSITION 5.1. *$\tilde{c}_{ER}(v)$ is un unbiased estimator for $c_{ER}(v)$ (i.e. $E[\tilde{c}_{ER}(v)] = c_{ER}(v)$).*

Proof. We show that $Y := \frac{n}{k}\sum_{s_i \in S} d_{\mathrm{ER}}(v, s_i)$ is an unbiased estimator for $s(v) = \sum_{w \in V} d_{\mathrm{ER}}(v, w)$, then the theorem follows directly from the properties of expected value. In the following, we denote the set of k-combinations of V with V_k.

$$\begin{aligned} E(Y) &= \sum_{S=\{s_1,\dots,s_k\} \in V_k} \frac{1}{\binom{n}{k}} \frac{n}{k} \sum_{s_i \in S} d_{\mathrm{ER}}(v, s_i) = \\ &= \frac{1}{\binom{n}{k}} \frac{n}{k} \sum_{w \in V} \binom{n-1}{k-1} d_{\mathrm{ER}}(v, w) \\ &= \sum_{w \in V} d_{\mathrm{ER}}(v, w). \end{aligned}$$ □

With k pivots, the empirical complexity of our approach is $\mathcal{O}\big(km \log(1/\tau)\big)$ with a multigrid solver. Our experiments in Section 6.4 show that a very small k (e.g., $k = 10$) is already enough to get a very good approximation.

5.2 Projection-based approximation. Spielman and Srivastava [24] show how to compute an approximation of effective resistance based on the JLT. Let B be the $m \times n$ incidence matrix where each row corresponds to an edge of G and each node corresponds to a node such that, for edge $e = \{u, v\}$, $B(e, u) = +1$, $B(e, v) = -1$ and $B(e, w) = 0\ \forall w \neq u, v$ (since G is undirected, the direction of edge e can be chosen arbitrarily). Then, they show that the effective resistance between node u and node v can be re-written as $d_{\mathrm{ER}}(u, v) = ||W^{1/2}BL^{\dagger}(e_u - e_v)||_2^2$, where W is the diagonal $m \times m$ matrix such that $W(e, e) = \omega(e)$, $L^{\dagger}$ is the Moore-Penrose pseudoinverse [13] of L and e_u is the $n \times 1$ vector such that $e(u) = 1$ and equal to 0 everywhere else. The effective resistances can therefore be seen as pairwise distances between vectors in $\{W^{1/2}BL^{\dagger}e_u\}_{u \in V}$, which allows to apply the JLT: If we project the vectors into a lower-dimensional space spanned by $k = \mathcal{O}\big(\log n\big)$ random vectors, the pairwise distances are approximately preserved. In other words, we can consider the pairwise distances between vectors in $\{QW^{1/2}BL^{\dagger}e_u\}_{u \in V}$, where Q is a random projection matrix of size $k \times m$ with elements in $\{0, +\frac{1}{\sqrt{k}}, -\frac{1}{\sqrt{k}}\}$.

Since we do not want to compute $QW^{1/2}BL^{\dagger}$ directly (it would require to invert L), we approximate it by solving k linear systems: for $i = 1, ..., k$, the i-th row z_i^T of $QW^{1/2}BL^{\dagger}$ can be computed by solving the system $Lz_i = \{QW^{1/2}B\}_{\cdot,i}$, see Algorithm 1 (which we reuse from [24]). Note that the multiplication in Line 2 requires only $\mathcal{O}\big(2m \log n\big)$ operations, since B is sparse (with $2m$ non-zero entries) and W is diagonal. When choosing k in

Algorithm 1: Effective resistance approximation [24]

```
   Input  : G = (V, E)
   Output : Approx. d̃_ER(u, v) ∀(u, v) ∈ V × V
1  Construct random matrix Q;
2  Compute Y = QW^{1/2}B;
3  Z ← empty k × n matrix;
4  for i = 1, ..., k do
5  |  solve the system Lz_i = Y_{·,i};
6  |  Z_{i,·} ← z_i^T;
7  end
8  foreach (u, v) ∈ V × V do
9  |  d̃_ER(u, v) ← ||Z_{·,u} − Z_{·,v}||_2^2;
10 end
11 return d̃_ER
```

Algorithm 1 equal to $\mathcal{O}\big(\log n/\epsilon^2\big)$ for any $\epsilon > 0$, it was shown [24] that, with probability $\geq 1 - 1/n$,

$$(1 - \epsilon) d_{\mathrm{ER}}(v, w) \leq \tilde{d}_{\mathrm{ER}}(v, w) \leq (1 + \epsilon) d_{\mathrm{ER}}(v, w)$$

for all $(v, w) \in V \times V$. An approximation of current-flow closeness for node v can therefore be computed as $\tilde{c}_{\mathrm{ER}}(v) := (n-1)/\sum_{w \neq v} \tilde{d}_{\mathrm{ER}}(v, w)$. This requires $\mathcal{O}\big(2m \log n\big)$ for Line 2 and $\mathcal{O}\big(km \log(1/\tau)\big)$ empirical time for Lines 4 - 7. Then we need to compute $\tilde{d}_{\mathrm{ER}}(v, w)$ for all $w \neq w$, which requires $\mathcal{O}\big(nk\big)$ operations. Assuming $n = \mathcal{O}\big(m\big)$, the total (empirical) running time is $\mathcal{O}\big(m(k \log(1/\tau) + log n)\big)$.

If $(1 - \epsilon) d_{\mathrm{ER}}(v, w) \leq \tilde{d}_{\mathrm{ER}}(v, w) \leq (1 + \epsilon) d_{\mathrm{ER}}(v, w)$ for each $w \neq v$, then also $(1 - \epsilon) c_{\mathrm{ER}}(v) \leq \tilde{c}_{\mathrm{ER}}(v) \leq (1 + \epsilon) c_{\mathrm{ER}}(v)$. This is provably true only with probability $(1 - 1/n)^{n-1}$. However, our experimental results show that the approximation works well in practice: on all tested instances, $\tilde{c}_{\mathrm{ER}}$ is *always* within a $(1 + \epsilon)$-factor from c_{ER} (see Section 6.4).

6 Experimental Evaluation

In this section we evaluate the performance of the two approximation algorithms described in Section 5. First, we want to give some more details on the implementation, the benchmarking setup and the graph instances we used, before elaborating on the results of our evaluation.

6.1 Implementation. We implemented both approximation algorithms in NetworKit [26], an open-source tool for fast exploratory analysis of massive networks. As became apparent in Section 3.2, linear systems play a quite important role in network analysis and since both approximation approaches introduced in this paper rely on solving Laplacian systems, we provide the Laplacian solver LAMG by Livne and Brandt [19] in NetworKit with our own new implementation. The original implementation by Livne and Brandt is written in Matlab with some performance-critical parts in C and therefore difficult for us to integrate into large-scale network analysis workflows.

In informal experiments, our C++ implementation of LAMG outperforms their Matlab/C implementation regarding the solve times by a factor of 1.5 on average. In comparison the CMG solver by Koutis et al. [17] is on average 11% faster than our LAMG implementation on our large test instances. When solving linear systems, in all our experiments we set the relative residual error τ to 10^{-5}.

6.2 Benchmarking Setup. All experiments were done on a machine equipped with 256 GB RAM and a 2.7 GHz Intel Xeon CPU E5-2680 having 2 sockets with 8 cores each and hyperthreading enabled. The machine runs 64 bit SUSE Linux and we compiled our code with g++-4.8.1 and OpenMP 3.1.

6.3 Instances. Tables 1 and 2 in the appendix show the set of instances we use for our experiments. While Table 1 includes rather small complex networks with up to about 150 000 edges, Table 2 includes larger networks with up to 56 million edges. If a network has more than one connected component, we used the largest connected component (LCC). We ignore self-loops and the direction of edges in case a graph is directed. All the graphs are unweighted.

6.4 Approximation algorithms. In this section we compare the two approximation algorithms described in Section 5.1 and Section 5.2, respectively. We refer to the first one as SAMPLING and to the second one as PROJECTION. SAMPLING depends on the number $|S|$ of samples, whereas PROJECTION depends on the dimension k of the $k \times n$ random projection matrix. For simplicity, we call *exact* the approach computing c_{ER} as in equation 4.3, solving $n-1$ linear systems to the desired tolerance τ.

For our experiments, we select 100 nodes for each of the networks shown in Table 1 in the appendix. For each of these nodes, we compute current-flow exactly (to the desired tolerance τ) and the two approximations with different parameters. In particular, we set the number $|S|$ of samples of SAMPLING to 10, 20, 50, 100, 200, 500, and 1000. When running PROJECTION, we fix k to $\lceil \log n/\epsilon^2 \rceil$ and set ϵ equal to 0.5, 0.2, 0.1, and 0.05. To measure the accuracy of the algorithms, we use the well-known Spearmann rank correlation coefficient, which measures how close the ranking of nodes determined by the approximation algorithm is close to that of the exact algorithm. We recall that the closer the Spearmann coefficient is to 1, the more correlated are the two rankings, with 0 meaning no correlation and 1 meaning the two ranks are identical.

Figure 2 reports the accuracy (Spearmann coefficient) and the running times in seconds for each approximation algorithm and for each parameter. We do not report explicitly to which parameter each point in the plot corresponds to, but this can be easily deduced from the running times: a smaller sample size corresponds to a smaller running time for SAMPLING and a larger ϵ corresponds to a smaller running time for PROJECTION. Figure 2 reports, for each approximation algorithm and for each parameter, the average over all networks of Table 1 of time and Spearmann coefficient.

The results are quite self-explanatory: the SAMPLING approach clearly outperforms PROJECTION and its accuracy is extremely high already with only 10 samples. We also compute for each algorithm and parameter the number of rank inversions,

i.e. the number of node pairs $\{u, v\}$ for which the approximated closeness of u is smaller than the approximated closeness of v, but the exact closeness of u is larger than or equal to the exact one of v (or vice versa). With ten pivots, the average number of rank inversions of SAMPLING is 12.5; it is always below 10 for higher number of samples. This means that, out of $\binom{100}{2} = 4950$ pairs, less than 10 are inverted, corresponding to 0.2%.

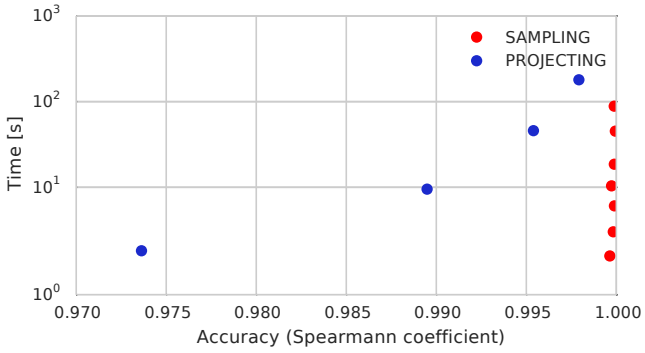

Figure 2: Time vs. Spearmann coefficient for the two approximation algorithms, using different parameters. The points represent the average among the networks of Table 1 in the Appendix.

In addition to accuracy in terms of ranks, we also evaluate the maximum relative error. We define the relative error for a node v as $e(v) = \max\{r(v), 1/r(v)\}$, where $r(v)$ is the ratio between the exact current-flow closeness of v and its approximation. The maximum relative error is then defined as $\max_{v \in V} e(v)$. Figure 3 reports the results. It is interesting to notice that, with respect to this measure, the two algorithms behave quite similarly. Also, notice that the maximum relative error for PROJECTION is always smaller than ϵ (we recall the values of ϵ used are 0.05, 0.1, 0.2 and 0.5), although we can only prove that this is true with probability at least $(1 - 1/n)^{n-1}$.

To summarize, our results show that both algorithms lead to very good accuracy in terms of maximum relative error, whereas the sampling approach better preserves the ranking of nodes, even when the number of samples is very small. For this reason, in our experiments on large graphs, we make use of the sampling approach. On average (over the instances of Table 1), computing c_{ER} on 100

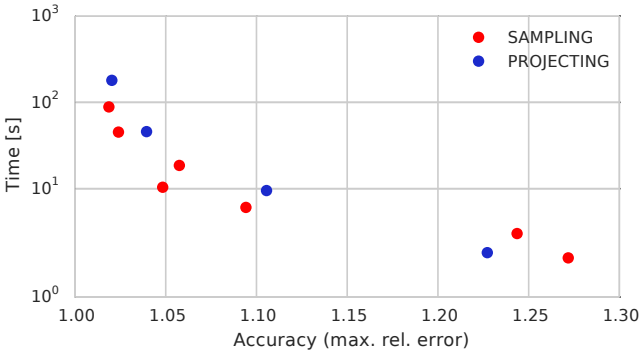

Figure 3: Time vs. maximum relative error for the two approximation algorithms, using different parameters. The points represent the average among the networks of Table 1 in the Appendix.

nodes takes more than 20 minutes, whereas using SAMPLING with 20 pivots takes only 2.87 seconds. Table 3 in the Appedix shows the detailed running times.

6.5 Comparison with shortest-path closeness. As explained in Section 4, our intuition is that current-flow closeness should represent the efficiency of a node reaching the other nodes of the network better than shortest-path closeness. To verify this assumption, we first compare the two measures in terms of their capability to discriminate between different nodes. In this experiments, we use the networks of Table 1 and compute (exactly) current-flow and shortest-path closeness on 100 randomly chosen nodes. Figure 4 shows the relative standard deviation for shortest-path and current-flow closeness. The relative standard deviation is defined as the standard deviation divided by the average. It is always significantly higher for current-flow closeness than it is for shortest-path closeness, meaning that there is much more variation in the scores computed by the former.

Also, similarly to what has been done in [20] for spanning edge centrality and edge betweenness centrality, we measure the resilience to noise, in this case for current-flow closeness and shortest-path closeness. The idea is to add edges to the graph and see how well the initial rankings are preserved. Our intuition is that, if we add some edge that creates a shortcut between a node v and some other

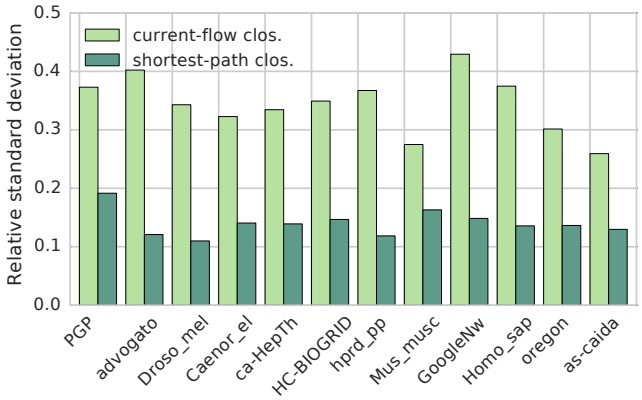

Figure 4: Relative standard deviation for shortest-path and current-flow closeness.

nodes, the shortest-path closeness of v will be more affected than its current-flow closeness, since the former takes only shortest paths into account. This is confirmed by our experiments, summarized in Figure 5. For each network in Table 1, we insert a percentage of the total number of edges varying from 1% to 10%. To have a high number of shortcuts involving the sampled nodes, we always add edges between one of the sampled nodes and other nodes of the graph. Figure 5 shows, for each percentage of inserted edges, the average among all tested networks of the Spearmann correlation coefficient between the initial ranking and the ranking after the insertions. Figure 5 shows that current-flow closeness is more resilient to edge insertions and the difference between the resilience of the two measures increases the more the graph changes.

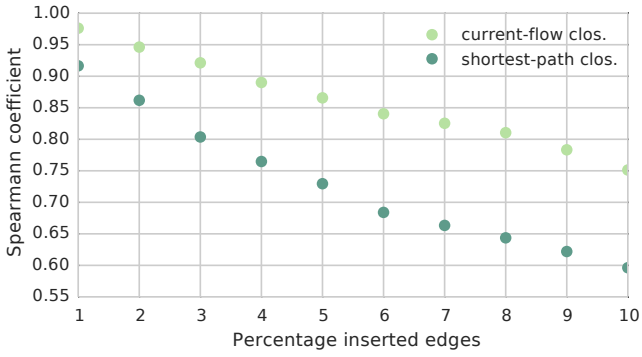

Figure 5: Resilience to noise for different percentages of inserted edges. The points represent the average among the networks of Table 1 in the Appendix.

6.6 Correlation with degree. In certain random geometric graph models, such as ϵ-graphs, kNN graphs, and Gaussian similarity graphs, it was recently shown [27] that the effective resistance between two nodes u and v converges to $1/\deg(u) + 1/\deg(v)$ when the number of nodes goes to infinity. This result also has implications on current-flow closeness: when the number of nodes goes to infinity in such graphs, $c_{\mathrm{ER}}(v)$ goes to $c_{\mathrm{A}}(v) := (n-1)/\sum_{w \neq v}(1/\deg(v) + 1/\deg(w))$. However, the structure of real-world networks (e.g. complex or street networks) is significantly different from random graphs and it is not clear how close current-flow closeness is to this asymptotic value in reality. In this section we therefore study the correlation between current-flow closeness, as well as shortest-path closeness and betweenness, with c_{A}. In our experiments, we consider large networks with up to 56 millions nodes and edges and we use the sampling approach to approximate current-flow closeness (with 20 pivots). Table 4 in the appendix shows the running times of our approximation when computing the closeness of a single node. We approximate betweenness using the approach presented in [12], which is already implemented in NetworKit. Since the results are very different for street and complex networks, we study them separately.

Figure 6 shows the Spearmann coefficient computed between the three centrality measures and c_{A} on the complex networks of Table 2 in the Appendix. The results show that there is in fact a strong positive correlation. This is weaker for shortest-path closeness, with an average Spearmann coefficient of 0.63 and stronger for betweenness and current-flow closeness, with an average of 0.81 and 0.89, respectively. We obtain similar results on the smaller instances of Table 1, where the averages are 0.63, 0.85 and 0.89 for shortest-path closeness, betweenness and current-flow closeness, respectively. The results are very different for street networks (Figure 7). Here the correlation with the degrees is generally very low and sometimes even negative, with an average of -0.02 for closeness, 0.35 for betweenness and 0.07 for current-flow closeness.

While c_{A} and c_{ER} are unrelated on street networks, our results show that there is actually

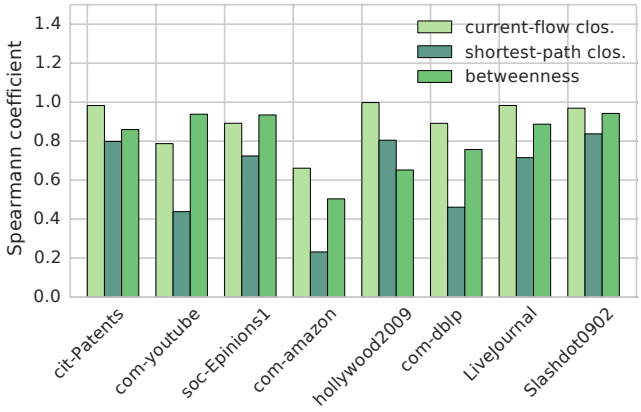

Figure 6: Correlation with c_{A} for complex networks.

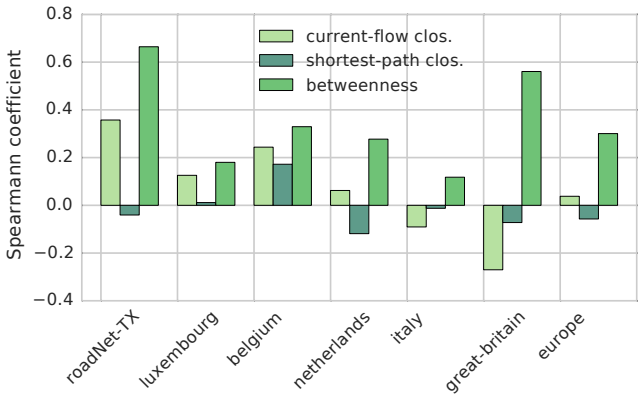

Figure 7: Correlation with c_{A} for street networks.

a strong correlation between them in complex networks. This behavior is likely due to the different type of degree distributions in the two network classes. While complex networks usually feature a skewed degree distribution with many small, but also some high-degree nodes, the degrees in street networks are closely concentrated. Consequently, in some applications where a very good accuracy is not needed, c_{A} might be used as an approximation of c_{ER} in complex networks. The same thing can be said for betweenness, which is only slightly less correlated with c_{A} than c_{ER}. This is very convenient, since c_{A} can be computed in $\mathcal{O}(m)$ time. However, our results in Section 6.4 show that our sampling-based approach can compute an extremely accurate approximation in time $\mathcal{O}(km\log(1/\epsilon))$ even when the number k of samples is very small. For this reason, we believe the sampling approach is probably the best option for most applications.

7 Conclusions

Although many important graph properties can be formulated in terms of Laplacian linear systems, popular network analysis frameworks lack an implementation of state-of-the-art solvers. We bridge this gap by providing a Lean Algebraic Multigrid (LAMG) implementation in NetworKit, our toolkit for large-scale network analysis.

Based on our new LAMG implementation and our algorithms Sampling and Projection, we have computed current-flow closeness centrality and provided the first published results on its behavior on large real-world networks. Our algorithms lead to very accurate results and, thanks to them, we are now able to compute an estimation of current-flow closeness of a subset of nodes on networks with tens of millions of nodes and edges within a few seconds or minutes. In our experiments current-flow closeness alleviates two known problems of shortest-path closeness and can thus be seen as a viable alternative in many scenarios. We have also shown empirically that there is a strong correlation between degrees and both current-flow closeness and betweenness centrality in complex networks, whereas the degree and current-flow closeness are basically unrelated in street networks.

Our approach based on Sampling or Projection is very fast in scenarios where we only need to compute the current-flow closeness for a subset of nodes. However, it might become too expensive if closeness has to be computed for all nodes. In these scenarios an interesting aspect of future work is whether our approximation would still be the best approach or whether inverting the Laplacian of the matrix would be faster in this case.

Another interesting question is whether techniques similar to those used for shortest-path closeness [3] can be used to approximate the top-k nodes with highest current-flow closeness without computing it for all nodes.

References

[1] Lasagne network dataset. `http://lasagne-unifi.sourceforge.net`.

[2] D. A. Bader, H. Meyerhenke, P. Sanders, C. Schulz, A. Kappes, and D. Wagner. Benchmarking for

graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, pages 73–82. 2014.

[3] E. Bergamini, M. Borassi, P. Crescenzi, A. Marino, and H. Meyerhenke. Computing top-k closeness centrality faster in unweighted graphs. In *Proc. of the 18th Workshop on Algorithm Engineering and Experiments, ALENEX 2016*, pages 68–80. SIAM, 2016.

[4] E. G. Boman, K. Deweese, and J. R. Gilbert. Evaluating the dual randomized kaczmarz laplacian linear solver. *Informatica*, 40:95–107, 2016.

[5] U. Brandes and D. Fleischer. Centrality measures based on current flow. In *Proc. of the 22nd Annual Symposium on Theoretical Aspects of Computer Science, STACS 2005*, volume 3404 of *LNCS*, pages 533–544. Springer, 2005.

[6] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S. Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proc. of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 273–282. ACM, 2011.

[7] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[8] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, Dec. 2011.

[9] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25(7):789–812, 1999.

[10] D. Eppstein and J. Wang. Fast approximation of centrality. *Journal of Graph Algorithms and Applications*, 8:39–45, 2004.

[11] E. R. Gansner, Y. Hu, and S. C. North. A maxent-stress model for graph layout. *IEEE Trans. Vis. Comput. Graph.*, 19(6):927–940, 2013.

[12] R. Geisberger, P. Sanders, and D. Schultes. Better approximation of betweenness centrality. In *10th Workshop on Algorithm Engineering and Experiments, ALENEX 2008*, pages 90–100. SIAM, 2008.

[13] G. H. Golub and C. F. V. Loan. *Matrix computations*. Johns Hopkins University Press, 1996.

[14] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proc. of the 7th Python in Science Conference*, pages 11–15, 2008.

[15] D. Hoske, D. Lukarski, H. Meyerhenke, and M. Wegner. Is nearly-linear the same in theory and practice? A case study with a combinatorial laplacian solver. In *Proc. of 14th International Symposium on Experimental Algorithms, SEA 2015*, volume 9125, page 205. Springer, 2015.

[16] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Proc. of the 45th Annual ACM Symposium on Theory of Computing*, pages 911–920, 2013.

[17] I. Koutis, G. L. Miller, and D. Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 115(12):1638–1646, 2011.

[18] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[19] O. E. Livne and A. Brandt. Lean algebraic multigrid (LAMG): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

[20] C. Mavroforakis, R. Garcia-Lebron, I. Koutis, and E. Terzi. Spanning Edge Centrality: Large-scale Computation and Applications. In *Proc. of the 24th International Conference on World Wide Web, WWW 2015*, pages 732–742, 2015.

[21] H. Meyerhenke, B. Monien, and S. Schamberger. Graph partitioning and disturbed diffusion. *Parallel Computing*, 35(10-11):544–569, 2009.

[22] T. P. Peixoto. The graph-tool python library. *figshare*, 2014.

[23] J. Ruge and K. Stüben. Algebraic multigrid. *Multigrid methods*, 3:73–130, 1987.

[24] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[25] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proc. of the 36th ACM Symposium on Theory of Computing, STOC 2004*, pages 81–90, 2004.

[26] C. L. Staudt, A. Sazonovs, and H. Meyerhenke. NetworKit: A tool suite for high-performance network analysis. *Network Science*, To appear.

[27] U. von Luxburg, A. Radl, and M. Hein. Hitting and commute times in large random neighborhood graphs. *Journal of Machine Learning Research*, 15(1):1751–1798, 2014.

A Instances used for the experiments

Table 1: Properties of smaller benchmark instances used in this paper.

Graph	#Nodes in LCC	#Edges in LCC	Description	Ref.
PGP	10680	24316	PGP trust network	[2]
advogato	5272	42816	Advocato trust network	[1]
Drosophila_melanogaster	10424	40660	Interactome	[1]
Caenorhabditis_elegans	4428	9659	Metabolic network	[1]
CA-HepTh	8638	24806	Collaboration Network	[18]
HC-BIOGRID	4039	10321	Genetic interaction	[1]
hprd_pp	9219	36900	Human proteine interaction	[1]
Mus_musculus	3745	5170	Interactome	[1]
GoogleNw	15763	148585	Hyperlinks between web pages	[1]
Homo_sapiens	13478	61006	Metabolic network	[1]
oregon2_010526	11461	32730	AS peering network	[18]
as-caida20071105	26475	53381	CAIDA AS relationships	[18]

Table 2: Properties of larger benchmark instances used in this paper.

Graph	#Nodes in LCC	#Edges in LCC	Description	Ref.
cit-Patents	3764117	16511740	Citation Network	[18]
com-Amazon	334863	925872	Amazon Product Network	[18]
com-DBLP	317080	1049866	Collaboration Network	[18]
com-Youtube	1134890	2987624	Youtube Social Network	[18]
hollywood-2009	1069126	56306653	Collaboration Network	[8]
com-LiveJournal	3997962	34681189	LiveJournal Social Network	[18]
Slashdot0902	82168	504230	Slashdot Zoo Social Network	[18]
soc-Epinions1	75877	405739	Epinions Social Network	[18]
roadNet-TX	1351137	1879201	Road Network of Texas	[18]
luxembourg.osm	114599	119666	Road Network of Luxembourg	[2]
belgium.osm	1441295	1549970	Road Network of Belgium	[2]
netherlands.osm	2216688	2441238	Road Network of the Netherlands	[2]
italy.osm	6686493	7013978	Road Network of Italy	[2]
great_britain.osm	7733822	8156517	Road Network of Great Britain	[2]
europe.osm	50912018	54054660	Road Network of Europe	[2]

B Additional experimental results

Table 3: Comparison between exact (= within desired tolerance τ) and Sampling approach, with 20 pivots. The first two columns report the running times of the two approaches, when computing current-flow closeness on 100 nodes. The third column reports the Spearmann rank correlation coefficient between the approaches and the fourth the percentage of rank inversions.

Graph	Time exact [s]	Time Sampling 20 [s]	Spearmann coeff.	Rank Inver.
PGP	558.97	1.68	0.99990	0.14%
advogato	383.42	2.39	0.99986	0.16%
Drosophila_melanogaster	1077.78	3.50	0.99986	0.12%
Caenorhabditis_elegans	68.50	0.64	0.99975	0.28%
CA-HepTh	800.65	2.87	0.99989	0.14%
HC-BIOGRID	186.47	1.92	0.99975	0.28%
hprd_pp	988.58	4.01	0.99990	0.10%
Mus_musculus	33.66	0.33	0.99958	0.44%
GoogleNw	4612.19	8.16	0.99987	0.14%
Homo_sapiens	1913.06	4.90	0.99999	0.02%
oregon2_010526	640.97	1.49	0.99988	0.14%
as-caida20071105	3354.62	2.62	0.99990	0.12%

Table 4: Running time of Sampling with 20 pivots when computing c_{ER} of a single node.

Graph	Time approximation [s]
cit-Patents	125.99
com-youtube	5.06
soc-Epinions1	0.28
com-amazon	2.56
hollywood2009	107.52
com-dblp	1.90
LiveJournal	287.27
Slashdot0902	0.41
roadNet-TX	6.28
luxembourg	0.13
belgium	2.39
netherlands	5.33
italy	10.91
great-britain	12.72
europe	103.34