Computing Top-k Closeness Centrality Faster in Unweighted Graphs[*]

Elisabetta Bergamini[†] Michele Borassi[‡] Pierluigi Crescenzi[§] Andrea Marino[¶]
Henning Meyerhenke[†]

Abstract

Centrality indices are widely used analytic measures for the importance of nodes in a network. Closeness centrality is very popular among these measures. For a single node v, it takes the sum of the distances of v to all other nodes into account. The currently best algorithms in practical applications for computing the closeness for all nodes exactly in unweighted graphs are based on breadth-first search (BFS) from every node. Thus, even for sparse graphs, these algorithms require quadratic running time in the worst case, which is prohibitive for large networks.

In many relevant applications, however, it is unnecessary to compute closeness values for all nodes. Instead, one requires only the k nodes with the highest closeness values in descending order. Thus, we present a new algorithm for computing this top-k ranking in unweighted graphs. Following the rationale of previous work, our algorithm significantly reduces the number of traversed edges. It does so by computing upper bounds on the closeness and stopping the current BFS search when k nodes already have higher closeness than the bounds computed for the other nodes.

In our experiments with real-world and synthetic instances of various types, one of these new bounds is good for small-world graphs with low diameter (such as social networks), while the other one excels for graphs with high diameter (such as road networks). Combining them yields an algorithm that is faster than the state of the art for top-k computations for all test instances, by a wide margin for high-diameter graphs.

Finally, we prove that the quadratic worst-case complexity cannot be improved on directed, disconnected graphs, under reasonable complexity assumptions.

Keywords: Closeness centrality, algorithmic network analysis, graph algorithms, algorithm engineering

1 Introduction

Finding a graph's most central nodes is a fundamental problem in network analysis. Intuitively, the centrality of nodes represents their structural importance within the domain under consideration. Depending on the definition, the most central nodes can be for example those that are traversed by a large fraction of shortest paths, those that can quickly reach the rest of the graph or the ones that recur more often in random walks [16, Chap. 7]. In this work, we focus on *closeness centrality*, a widely-used centrality measure which is defined as the inverse of the average shortest-path distance. In other words, a node with high closeness represents an individual or an entity that is close, on average, to all the other entities of the domain. The identification of the nodes with highest closeness finds its application in a plethora of research areas. Examples include facility location [12], marketing strategies [11] and identification of key infrastructure nodes as well as disease propagation control and crime prevention [3]. Unfortunately, computing the closeness of all nodes in a graph can be very expensive. The currently best algorithm solves an all-pairs shortest paths (APSP) problem to compute the distance between each pair of nodes. For an unweighted graph $G = (V, E)$ with n nodes and m edges, this can be done in $O(n^{2.373})$ using fast matrix multiplication [25] or in $O(nm)$ running a BFS from each node. Since real-world networks are often sparse and since the first approach contains large hidden constants, BFS-based approaches are predominant in practice (also see Section 2.2). Nevertheless, this running time becomes prohibitive already for networks with a few million nodes, restricting the exact computation of closeness

[*] E. B.'s and H. M.'s work is partially supported by German Research Foundation (DFG) grant ME 3619/3-1 within the Priority Programme 1736 *Algorithms for Big Data*. P. C.'s and A. M.'s work is partially supported by the Italian Ministry of Education, University, and Research (MIUR) under PRIN 2012C4E3KT national research project AMANDA — Algorithmics for MAssive and Networked DAta.

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

[‡] IMT Institute for Advanced Studies Lucca, Italy.

[§] Dipartimento di Ingegneria dell'Informazione, Università di Firenze, Italy.

[¶] Dipartimento di Informatica, Università di Pisa, Italy.

to a small fraction of real applications. Moreover, since closeness values tend to be close together [16, p. 182], resorting to approximations [9, 7] for accelerating the computation may lead to undesirable errors in the nodes' ranking w. r. t. closeness.

The problem we therefore target in this paper is the identification of the k nodes with highest closeness, and doing so faster than computing it for all nodes. As mentioned above, many research areas are indeed interested in the most central nodes of the network, rather than in the closeness value of each single node. For example, somebody willing to open a store might be interested in knowing one or a few locations that are close, on average, to many potential customers and not in the closeness of each possible location. To the best of our knowledge, only two methods that improve on exhaustive computation (that is, computing closeness for all nodes) have been proposed so far [18, 4]. Both are quite recent; only the slightly older one [18] can handle (nonnegative) edge weights.

Contribution. In this paper, we propose an algorithm for unweighted graphs that, compared to [18, 4], further reduces the number of traversed edges to find the top-k nodes w. r. t. closeness centrality. The basic idea is to find, for each node, an upper bound on its closeness and stop the computation when k nodes are found whose closeness is higher than the upper bounds for the other nodes. We present two techniques for computing upper bounds, each effective on a different class of graphs.

In the worst case, our algorithms are not guaranteed to be faster than computing closeness for all nodes. This is not surprising, since it was proven that the complexity of finding the node with highest closeness is equivalent (under subcubic reductions) to the APSP problem [1]. Nonetheless, by combining our two bounds, we achieve significant speedups (between one and four orders of magnitude) on exhaustive computation, both on street networks and complex networks. Compared to the best previous algorithm for unweighted networks [4], we improve the running time in all experiments. Intriguingly, there is a clear distinction between networks with high diameter (such as street networks) and low diameter (complex networks): while the acceleration by our algorithm is only 1.7 on average (but also up to a factor of 18) for complex networks, our algorithm outperforms the previous algorithm [4] by two orders of magnitude on networks with high diameter. Our last result proves that, on directed graphs, in the worst case, an algorithm computing the most closeness central vertex in time $O(m^{2-\epsilon})$ for some $\epsilon > 0$ would falsify the well-known Strong Exponential Time Hypothesis (SETH, [10]), which, informally, says that the SATISFIABILITY problem is not solvable in time $O((2-\epsilon)^n)$ for any $\epsilon > 0$, where n is the number of variables.

2 Preliminaries

2.1 Notation Let $G = (V, E)$ be a (strongly) connected unweighted graph with $n = |V|$ nodes and $m = |E|$ edges. We say node u and v are at distance k if the length of the shortest path between u and v is k. We refer to the distance from u to v as $d(u, v)$. Then, we define the total distance $S(v)$ of node v as the sum of the distances from v to all the other nodes, i. e. $S(v) = \sum_{w \in V} d(v, w)$. The closeness centrality for node v, $c(v)$, is defined as

$$c(v) = \frac{n-1}{S(v)} \quad . \tag{2.1}$$

We define the diameter of G, $\mathrm{diam}(G)$, as the maximum distance between any two nodes in G. Also, we define the neighborhood $N(v)$ of a node v as the set of nodes w such that $(v, w) \in E$ (or $\{v, w\} \in E$, for undirected graphs). The degree of v, $\deg(v)$, is the size of v's neighborhood.

Some extensions of closeness centrality have been proposed also for disconnected graphs [19, 8]. However, we restrict ourselves to connected undirected graphs and strongly-connected directed graphs, unless mentioned explicitly otherwise. We believe this is not a major limitation, since one could just apply our algorithm to the largest (strongly) connected component or to each component separately.

2.2 Related Work The closeness of all nodes in unweighted graphs can be computed by solving the APSP problem. For this problem, there is no solution that is always better than running a BFS from each node. This requires $O(n(n+m))$ time (for sparse graphs this is faster than approaches based on fast matrix multiplication). Since this running time is prohibitive for large networks, attention has been devoted to approximation algorithms. For graphs with bounded diameter, Eppstein and Wang [9] proposed an algorithm that computes the closeness of all nodes within an additive error ϵ with high probability. The method basically samples a set of source nodes, runs a BFS from them and uses the computed distances to extrapolate the closeness of the other nodes. Subsequently, Brandes and Pich [6] conducted an experimental evaluation of this approximation algorithm, also considering different ways of sampling the source nodes. A more refined approximation algorithm with better practical performance has recently been published [7]. Although the approximation algorithms can often provide scores that are close to the real ones, they may fail at preserving the ranking, in particular for nodes with similar closeness. For this reason, the problem of accurately computing the ranking of the top-k nodes with highest closeness has been considered, both exactly [18], with high probability [17] and with heuristics [15, 14]. Although the algorithms can actually save time compared to the exhaustive computation of closeness for all nodes, it was shown [4] that on many instances their running time is very close to that of APSP. Only very recently a new method that efficiently computes the top-k closeness values in unweighted graphs has been proposed [4] and shown to outperform the existing approaches on several real-world networks. We will refer to this method as CutClos, from the name of the procedure BFSCut, on which it is based. This procedure is performed for each node and it is basically a BFS that keeps track of the sum of the distances of the visited nodes and of a lower bound on the distances of the unvisited nodes. In particular, when all the nodes up to distance j have been visited in the BFS, the remaining nodes must be at distance at least $j+2$, with the exception of the outgoing neighbors of the nodes at distance j. The lower bound on the sum of the unvisited nodes's distances is therefore computed as number of unvisited nodes times $j+2$ minus the sum of the out-degrees of nodes at distance j. When the sum of the visited nodes plus the lower bound becomes larger than that of the k-th node with maximum closeness discovered so far, the BFS is interrupted, avoiding to visit the remaining edges. Clearly, for the approach to work well, the nodes with maximum closeness must be considered as early as possible. In the worst case, if nodes are considered in order of increasing closeness, the approach would be as bad as running a complete BFS for each node. Since in real-world networks there is often a correlation between degree and closeness, the authors propose to consider the nodes in order of decreasing degree.

3 Computing top-k closeness centrality

In this section we describe our new approach for computing the top-k nodes with maximum closeness. The basic idea of the algorithm is to keep track of a lower bound on the total distance of each node (and therefore an upper bound on the closeness). Let $S(v)$ be the total distance of node v and let $\tilde{S}(v)$ be the lower bound. Then, if $S(v) \leq \tilde{S}(w)\ \forall w \in V$, it is also true that $S(v) \leq S(w)\ \forall w \in V$. Thus, v is (one of) the node(s) with maximum closeness. This simple observation allows us to skip the computation of the exact value of $S(w)$ for all the remaining nodes.

The idea sketched above is implemented as Algorithm 1: First, we compute the lower bounds $\tilde{S}$ (Line 1) and insert all the nodes into a priority queue Q, ordered by increasing $\tilde{S}(v)$. Then, we extract the nodes from Q one after another (Line 10). If the priority $S^\star$ of the extracted node $v^\star$ is exactly the total distance of $v^\star$ ($\mathsf{exact}[v] = \text{true}$), then the closeness centrality of $v^\star$ is smaller than the closeness centrality of all the other nodes in Q. Therefore we can append $v^\star$ to the list of the most central nodes. Otherwise (i. e. $S^\star$ is only a lower bound on $S(v^\star)$), we compute the exact total distance of $v^\star$ (Line 16) and we re-enqueue $v^\star$ with priority $S(v^\star)$, possibly with some optimizations (see Section 3.3). Clearly, the tightness of the bounds $\tilde{S}$ can influence the algorithm's performance dramatically. If the bounds are close to the exact values, only a few iterations will be enough to find the k most central nodes, allowing us to skip the computation of $S(v)$ for a large portion of nodes. In Sections 3.1 and 3.2, we propose two different techniques for computing and updating the lower bounds $\tilde{S}$. The first one finds a tighter bound on networks with relatively large diameter and degree distribution with small variance (e. g., street networks), whereas the second one is more performant on complex networks.

Unlike CutClos [4], our algorithm computes lower bounds in the initialization phase, i. e. *before* computing the closeness of any node. This allows us to skip completely all the remaining nodes, once we find k nodes whose exact $S(v)$ is smaller than the lower bounds of the other vertices. On the contrary, CutClos does not compute any initial bound and starts a BFS from *each* node $v \in V$, interrupting it if the lower bound on $S(v)$ becomes larger than the current k-th smallest value of S (see Section 2.2 for more details). Since the lower bound is computed and updated during the BFS, this often requires to visit several edges before the bound becomes large enough to interrupt the BFS, in particular if the diameter is relatively large.

3.1 Level-based lower bound Let G be an undirected graph and let us consider a BFS traversal from a source node s. We refer to the distances $d(s, v)$ between s and all the nodes $v \in V$ as *levels*: node v is at level i if and only if the distance between s and v is i, and we write $l_s(v) = i$ (or simply as $l(v) = i$ if s is clear from the context or if the particular s is irrelevant). Let i and j be two levels, $i \leq j$. Then, the distance between any two nodes v at level i and w at level j must be at least $j-i$. Indeed, if $d(v, w)$ was smaller than $j-i$, w would be at level $i+d(v, w) < j$, which contradicts our assumption. It follows directly that $\sum_{w \in V} |l_s(w) - l_s(v)|$ is a lower bound on $S(v)$, for all $v, s \in V$:

```
Algorithm 1: Top-$k$ closeness
Input  : A graph $G = (V, E)$
Output: top-$k$ nodes with highest closeness and their
        closeness values $c(v)$
$\tilde{S} \leftarrow$ computeLowerBounds($G$);
$Q \leftarrow \emptyset$;
foreach $v \in V$ do
    $Q \leftarrow$ enqueue($v, p_v = \tilde{S}(v)$);
    exact[$v$] $\leftarrow$ false;
end
$i \leftarrow 0$;
TopK $\leftarrow$ [ ];
while $i < k$ do
    $(v^\star, S^\star) \leftarrow$ extractMin($Q$);
    if exact[$v^\star$] then
        TopK[$i$] $\leftarrow (v^\star, S^\star)$;
        $i \leftarrow i + 1$;
    end
    else
        $S(v) \leftarrow \sum_{w \in V} d(v, w)$;
        $Q \leftarrow$ enqueue($v, p_v = S(v)$);
        exact[$v$] $\leftarrow$ true;
    end
end
return TopK
```

LEMMA 3.1. $\tilde{S}(v) := \sum_{w \in V} |l_s(w) - l_s(v)| \leq S(v) \quad \forall v, s \in V$.

To improve the approximation, we notice that the number of nodes at distance 1 from v is exactly the degree of v. Therefore, all the other nodes w such that $|l(v) - l(w)| \leq 1$ must be at least at distance 2 (with the only exception of v itself, whose distance is of course 0). We can now define the level-based lower bound $\tilde{S}(v)^{(\mathrm{un})}_{\mathsf{L}}$ for undirected graphs as:

$$2(\#\{w \in V : |l(w) - l(v)| \leq 1\} - \deg(v) - 1) + \\ + \deg(v) + \sum_{\substack{w \in V \\ |l(w) - l(v)| > 1}} |l(w) - l(v)|,$$

that is:

$$2 \cdot \#\{w \in V : |l(w) - l(v)| \leq 1\} - \deg(v) - 2 + \\ + \sum_{\substack{w \in V \\ |l(w) - l(v)| > 1}} |l(w) - l(v)|. \tag{3.2}$$

A straightforward way to compute $\tilde{S}^{(\mathrm{un})}_{\mathsf{L}}$ would be to run a BFS from a node s and then, for each node v, to consider the level difference between v and all the other nodes. However, this would require $O(n^2)$ operations. Algorithm 2 describes a more efficient computation of $\tilde{S}^{(\mathrm{un})}_{\mathsf{L}}$. Since the bound $\tilde{S}$ for nodes at the same level differs only in the degrees of the nodes (see Eq. (3.2)), we can compute the approximation only once for each level and then subtract, for each node, its degree. Naming maxL the maximum level in the BFS search, Algorithm 2 computes, for each level i, the value $2 \cdot \#\{w \in V : |l(w) - i| \leq 1\} + \sum_{\substack{w \in V \\ |l(w) - i| > 1}} |l(w) - i|$ by summing over all the levels j and then uses this sum to compute $\tilde{S}^{(\mathrm{un})}_{\mathsf{L}}(v)$ for all nodes v at level i. This requires exactly $\sum_{i=1}^{\mathsf{maxL}} \mathsf{maxL} + \#\{v \in V : l(v) = i\} = \mathsf{maxL}^2 + n$ operations. Since $\mathsf{maxL} \leq \mathrm{diam}(G)$ and adding the complexity of the initial BFS, the following holds:

```
Algorithm 2: Level-based lower bound for
undirected graphs
Input  : A graph $G = (V, E)$
Output: Lower bounds $\tilde{S}^{(\mathrm{un})}_{\mathsf{L}}(v)$ of each node $v \in V$
$s \leftarrow$ seedNode();
$d \leftarrow$ BFSfrom($s$);
maxL $\leftarrow \max_{v \in V} d(s, v)$;
for $i = 1, 2, ..., $ maxL do
    L[$i$] = $\{w \in V : d(s, w) = i\}$;
    nL[$i$] = #L[$i$];
end
for $i = 1, 2, ..., $ maxL do
    sum $\leftarrow 0$;
    for $j = 1, 2, ..., $ maxL do
        if $|j - i| \leq 1$ then
            sum $\leftarrow$ sum + 2nL[$j$];
        end
        else
            sum $\leftarrow$ sum + $|j - i| \cdot$ nL[$j$];
        end
    end
    for $v \in$ L[$i$] do
        $\tilde{S}^{(\mathrm{un})}_{\mathsf{L}}(v) \leftarrow$ sum $- \deg(v) - 2$;
    end
end
return $\tilde{S}^{(\mathrm{un})}_{\mathsf{L}}$
```

PROPOSITION 3.1. *Computing the lower bound* $\tilde{S}^{(\mathrm{un})}_{\mathsf{L}}$ *takes* $O(\mathrm{diam}^2(G) + n + m)$ *time.*

Notice that many real-world networks (e.g. social networks) exhibit the *small world phenomenon* [24, 21], i.e. their diameter is $O(\log n)$. Therefore, for these networks and large enough n, the computation of the lower bound is linear in the number of edges.

For directed graphs, the result does not hold for nodes w whose level is smaller than $l(v)$, since there might be a directed edge or a shortcut from v to w. Yet, for nodes w such that $l(w) > l(v)$, it is still true that $d(v, w) \geq l(w) - l(v)$. For the remaining nodes (apart from the outgoing neighbors of v), we can only say that the distance must be at least 2. The upper bound for directed graphs $\tilde{S}^{(\mathrm{dir})}_{\mathsf{L}}$ can therefore be defined as:

$$2 \cdot \#\{w \in V : l(w) - l(v) \leq 1\} + \sum_{\substack{w \in V \\ l(w)-l(v)>1}} (l(w) - l(v)) - \deg(v) - 2 \qquad (3.3)$$

The computation of $\tilde{S}_{\mathsf{L}}^{(\mathrm{dir})}$ for directed graphs is analogous to the one described in Algorithm 2. The particular choice of the source node s does not influence the correctness of the bounds; however, it can determine how close they are to S. A node s with large eccentricity (i. e., maximum distance) is preferable, since the presence of more level allows for a major differentiation in the lower bounds of the different nodes. For this reason, in `seedNode()` in Algorithm 2, first we pick a node v at random and then we choose as s the node with maximum distance from v.

3.2 Neighborhood-based lower bound

In a tree we can compute the closeness centrality of all nodes faster than running a BFS from each node. We exploit this by providing an exact bound for trees which translates into a lower bound in general unweighted graphs later on.

Let us consider a node s for which we want to compute $S(s)$. The number of nodes at level 1 in the BFS tree from s is clearly the degree of s. What about level 2? Since there are no loops, all the neighbors of the nodes in $N(s)$ are nodes at level 2 for s, with the only exception of s itself. Therefore, naming $\mathsf{L}_k[s]$ the set of nodes at level k from s and $\#\mathsf{L}_k[s]$ the number of these nodes, we can write $\#\mathsf{L}_2[s] = \sum_{w \in N(s)} \#\mathsf{L}_1[w] - \deg(s)$. In general, we can always relate the number of nodes in each level k of s to the number of nodes at level $k-1$ in the BFS trees of the neighbors of s. Let us now consider $\#\mathsf{L}_k[s]$, for $k > 2$. Figure 1 shows an example where s has three neighbors w_1, w_2 and w_3. Suppose we want to compute $\#\mathsf{L}_4[s]$ using information from w_1, w_2 and w_3. Clearly, $\mathsf{L}_4[s] \subset \mathsf{L}_3[w_1] \cup \mathsf{L}_3[w_2] \cup \mathsf{L}_3[w_3]$; however, there are also other nodes in the union that are not in $\mathsf{L}_4[s]$. Let us consider w_1: the nodes in $\mathsf{L}_3[w_1]$ (red nodes in the leftmost tree) are of two types: nodes in $\mathsf{L}_4[s]$ (the ones in the subtree of w_1) and nodes in $\mathsf{L}_2[s]$ (the ones in the subtrees of w_2 and w_3). An analogous behavior can be observed for w_2 and w_3 (central and rightmost trees). If we simply sum all the nodes in $\#\mathsf{L}_3[w_1]$, $\#\mathsf{L}_3[w_2]$ and $\#\mathsf{L}_3[w_3]$, we would be counting each node at level 2 twice, i. e. once for each node in $N(s)$ minus one. In general, for $k > 2$, we can write

$$\#\mathsf{L}_k[s] = \sum_{w \in N(s)} \#\mathsf{L}_{k-1}[w] - \#\mathsf{L}_{k-2}[s] \cdot (\deg(s) - 1). \qquad (3.4)$$

Algorithm 3: Closeness centrality in trees

```
   Input  : A tree $T = (V, E)$
   Output: Closeness centralities $c(v)$ of each node $v \in V$
1  foreach $s \in V$ do
2      $\#\mathsf{L}_{k-1}[s] \leftarrow \deg(s)$;
3      $\mathsf{S}[s] \leftarrow \deg(s)$;
4  end
5  $\mathsf{k} \leftarrow 2$;
6  $\mathsf{nFinished} \leftarrow 0$;
7  while $\mathsf{nFinished} < n$ do
8      foreach $s \in V$ do
9          if $\mathsf{k} = 2$ then
10             $\#\mathsf{L}_k[s] \leftarrow \sum_{w \in N(s)} \#\mathsf{L}_{k-1}[w] - \deg(s)$;
11         end
12         else
13             $\#\mathsf{L}_k[s] \leftarrow \sum_{w \in N(s)} \#\mathsf{L}_{k-1}[w] -$
               $\#\mathsf{L}_{k-2}[s](\deg(s) - 1)$;
14         end
15     end
16     foreach $s \in V$ do
17         $\#\mathsf{L}_{k-2}[s] \leftarrow \#\mathsf{L}_{k-1}[s]$;
18         $\#\mathsf{L}_{k-1}[s] \leftarrow \#\mathsf{L}_k[s]$;
19         if $\#\mathsf{L}_{k-1}[s] > 0$ then
20             $\mathsf{S}[s] \leftarrow \mathsf{S}[s] + \mathsf{k} \cdot \#\mathsf{L}_{k-1}[s]$;
21         end
22         else
23             $\mathsf{nFinished} \leftarrow \mathsf{nFinished} + 1$;
24         end
25     end
26     $\mathsf{k} \leftarrow \mathsf{k} + 1$;
27 end
28 foreach $s \in V$ do
29     $c(v) \leftarrow (n-1)/\mathsf{S}[v]$;
30 end
31 return $c$
```

From this observation, we define a new method to compute the total distance of all nodes, described in Algorithm 3. Instead of computing the BFS tree of each node one by one, at each step we compute the number $\#\mathsf{L}_k[v]$ of nodes at level k for *all* nodes v. First (Lines 1 - 4), we compute $\#\mathsf{L}_1[v]$ for each node (and add that to $S(v)$). Then (Lines 7 - 27), we consider all the other levels k one by one. For each k, we use $\#\mathsf{L}_{k-1}[w]$ of the neighbors w of v and $\#\mathsf{L}_{k-2}[v]$ to compute $\#\mathsf{L}_k[v]$ (Line 10 and 13). If, for some k, $\#\mathsf{L}_k[v] = 0$, all the nodes have been added to $S(v)$. Therefore, we can stop the algorithm when $\#\mathsf{L}_k[v] = 0 \ \ \forall v \in V$.

PROPOSITION 3.2. *Algorithm 3 requires $O(\mathrm{diam}(T) \cdot m)$ operations to compute the closeness centrality of all nodes in a tree T.*

Proof. The for loop in Lines 1 - 4 of Algorithm 3 clearly takes $O(n)$ time. For each level of the while loop of Lines 7 - 27, each node scans its neighbors in Line 10 or Line 13. In total, this leads to $O(m)$

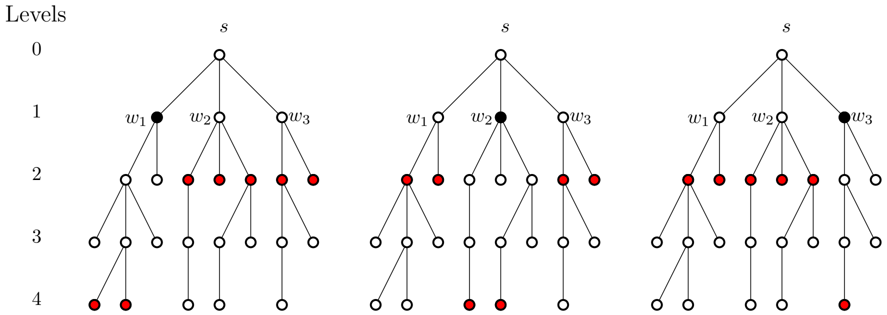

Figure 1: Relation between nodes at level 4 for s and the neighbors of s. The red nodes represent the nodes at level 3 for w_1 (left), for w_2 (center) and for w_3 (right).

operations per level. Since the maximum number of levels that a node can have is equal to the diameter, the algorithm requires $O(\text{diam}(T) \cdot m)$ operations. □

For cyclic undirected graphs, Eq. (3.4) is not true anymore – but a related lower bound on $\#\mathsf{L}_k[\cdot]$ will still be useful. Indeed, there could be nodes x for which there are multiple paths between s and x and that are therefore contained in the subtrees of more than one neighbor of s. This means that we would count x multiple times when considering its level k, overestimating the number of nodes at level k. However, what we know for sure is that at level k there cannot be *more nodes* than in Eq. (3.4). If, for each node v, we assume that the number $\#\mathsf{L}_k[v]$ of nodes at level k is that of Eq. (3.4) and we stop the computation when the sum of the numbers of nodes at levels $i \leq k$ is equal to n, we get a lower bound on $S(v)$, which we call $\tilde{S}_{\mathsf{N}}^{(\text{un})}(v)$. In fact, since $\#\mathsf{L}_k[v] \geq \#\{w \in V | d(v, w) = k\}$, the sum of the levels of the first n nodes found by our algorithm is always smaller than or equal to the sum of the actual distances of the n nodes. More formally, we can define $\tilde{S}_{\mathsf{N}}^{(\text{un})}(v)$ as

$$\tilde{S}_{\mathsf{N}}^{(\text{un})}(v) := \sum_{k=1}^{\text{diam}(G)} k \cdot \min\left\{\#\mathsf{L}_k[v],\ n - \sum_{i=0}^{k-1} \#\mathsf{L}_i[v]\right\} \tag{3.5}$$

The procedure is described in Algorithm 4. The computation of $\tilde{S}_{\mathsf{N}}^{(\text{un})}$ works basically like Algorithm 3, with the difference that here we keep track of the number of the nodes found in all the levels up to k ($\mathsf{nVisited}$) and stop the computation when $\mathsf{nVisited}$ becomes equal to n (if it becomes larger, in the last level we consider only $n - \mathsf{nVisited}$ nodes, see Lines 28 - 33).

PROPOSITION 3.3. *For an unweighted graph G, computing the lower bound $\tilde{S}_{\mathsf{N}}^{(\text{un})}$ described in Algorithm 4 takes $O(\text{diam}(G) \cdot m)$ time.*

Proof. Like in Algorithm 3, the number of operations performed by Algorithm 4 at each level of the while loop is $O(m)$. At each level i, all the nodes at distance i are accounted for (possibly multiple times) in Lines 12 and 15. Therefore, at each level, the variable $\mathsf{nVisited}$ is always greater than or equal to the the number of nodes v at distance $d(v) \leq i$. Since $d(v) \leq \text{diam}(G)$ for all nodes v, the maximum number of levels scanned in the while loop cannot be larger than $\text{diam}(G)$, therefore the total complexity is $O(\text{diam}(G) \cdot m)$. □

In directed graphs, we can simply consider the out-neighbors, without subtracting the number of nodes discovered in the subtrees of the other neighbors in Eq. (3.4). The lower bound (which we refer to as $\tilde{S}_{\mathsf{N}}^{(\text{dir})}$) is obtained by replacing Eq. (3.4) with the following in Lines 12 and 15 of Algorithm 4 and in Eq. (3.5):

$$\#\mathsf{L}_k[s] = \sum_{w \in N(s)} \#\mathsf{L}_{k-1}[w] \tag{3.6}$$

In the following, we will use $\tilde{S}_{\mathsf{N}}$ and the term *neighborhood-based lower bound* to indicate either $\tilde{S}_{\mathsf{N}}^{(\text{un})}$ or $\tilde{S}_{\mathsf{N}}^{(\text{dir})}$, depending on whether the graph under consideration is undirected or directed. Analogously, we will use $\tilde{S}_{\mathsf{L}}$ and *level-based lower bound* for both $\tilde{S}_{\mathsf{L}}^{(\text{un})}$ and $\tilde{S}_{\mathsf{L}}^{(\text{dir})}$.

3.3 Additional engineering The two lower bounds that we described in the previous sections cover the first line of Algorithm 1 (i. e. $\mathsf{computeLowerBounds}(G)$). One could either compute only one of them or both of them, taking for each node the maximum among the two. Since in Algorithm 1 we stop the computation only when we find k nodes whose exact value of S is smaller than the lower bounds of the other nodes, having lower bounds that are close to the exact values is crucial. For example, if all the lower bounds were smaller than $\min_{v \in V} S(v)$, our algorithm would be as bad as computing closeness for each node. On the other hand, if $S(v_1), \ldots, S(v_k)$ of the top-k nodes $(v_1, \ldots, v_k)$ were smaller than the lower bounds of the remaining nodes, we could find the top-k nodes with a constant number of BFSs. For this reason, after the initialization, we try to further improve the bounds of the nodes also while computing the exact closeness of a node v in Line 16. In the case of the level-based lower bound, we can keep track of the nodes in each level of the BFS from v and recompute $\tilde{S}_L$ using v as source. If, for some node, the new bound is larger than its current one, we can update it. Also, to additionally reduce the number of visited edges, we can stop the BFS beforehand when we discover that the closeness of v cannot be larger than that of the current k-th top node, similarly to CutClos [4] (see Section 2.2 and [4] for more details).

4 Experiments

Implementation and Settings. For the experimental evaluation, we implemented three versions of our algorithm: one based on the level-based lower bound (FastClos$_L$), one that uses the neighborhood-based lower bound (FastClos$_N$) and one that combines both (FastClos), i. e. takes the maximum among the two bounds. For a comparison with the state of the art, we also implemented the algorithm presented by Borassi et al. [4] (CutClos) as baseline. This algorithm was shown to outperform the other existing algorithms for exact and approximate top-k closeness centrality [4]. We implemented all algorithms in C++, building on the open-source *NetworKit* framework [20]. The machine used has 2 x 8 Intel(R) Xeon(R) E5-2680 cores at 2.7 GHz, of which we use only one, and 256 GB RAM. All computations are sequential to make the comparison to previous work more meaningful.

Methodology. We test the algorithms on a large collection of real-world networks. Tests on synthetic networks are added in order to examine the scaling behavior of the algorithms. The real networks are taken from SNAP (`snap.stanford.edu`), KONECT (`konect.uni-koblenz.de`) and the 10th DIMACS Implementation Challenge [2]. In case of disconnected networks, we always extract the largest (strongly) connected component first. In all the tests we adopt as a measure of performance the *performance ratio*, introduced by Borassi et al. [4]. Naming $|E_{vis}|$ the number of edges visited by a top-k algorithm, the performance ratio is defined as $|E_{vis}|$ divided by the number of edges that the exhaustive algorithm (running a BFS for all nodes) would use, i. e. $\frac{|E_{vis}|}{n \cdot m}$. This measure does not depend on the particular implementation of the algorithms, but only on the actual number of operations performed. Moreover, it allows an assessment independent of computer architectures. In the measure we do not consider the initialization phases of any of the algorithms, since the times they require are negligible. We refer to the inverse of the performance ratio as *speedup* and evaluate the algorithms by comparing their speedups.

Results. Our results show that our two lower bounds perform differently depending on the nature of the network. In particular, FastClos$_L$ performs very well on street networks (where closeness has a straightforward interpretation) and on networks with similar properties, i. e. a relatively large diameter and small variance in the degree distribution. The presence of several "levels" with relatively few vertices allows for a differentiation in the lower bounds of the different nodes. This leads to a fast identification of the most central nodes. On the other hand, FastClos$_N$ and the baseline CutClos perform relatively poorly on this kind of networks, because their assumption is that closeness centrality is strongly correlated with the degrees. For example, to find the top node with highest closeness on the street network of Luxembourg, FastClos$_L$ (as well as the combined version FastClos) has a speedup of more than 400, whereas FastClos$_N$ and CutClos have speedups of 3.2 and 2.6, respectively. Since CutClos does not scale well to large street networks (for networks with millions of nodes, it does not finish the computation in two days), we tested only FastClos on a set of street networks, also taken from [2]. The results are shown in Table 1. Using FastClos, we are able to find the top 100 nodes with highest closeness in the whole European street network in minutes (where the exhaustive algorithm would take years).

To examine the scaling on graphs with properties similar to those of street networks, we compared the algorithms on synthetic Delaunay graphs of increasing sizes, taken from [2]. Figure 2 (left) shows the speedups of FastClos and those of CutClos, with k (the number of nodes with highest closeness computed) equal to 1 and 10. The results obtained using only FastClos$_N$ are extremely similar and therefore omitted. Since we use a log-log scale, it is quite apparent that the speedup of FastClos increases exponentially with the size of the graph, reaching values of more than 10^3 for graphs with 2^{20} nodes. The speedup of CutClos, in turn, is basically constant and always smaller than 10.

Very different is the behavior of the algorithms on complex networks (e. g. social networks, communication networks), characterized by a strongly varying

Graph	Nodes	Edges	Speedup ($k = 1$)	Speedup ($k = 10$)	Speedup ($k = 100$)
`luxembourg.osm`	114 599	119 666	415.2	375.7	236.7
`belgium.osm`	1 441 295	1 549 970	4 214.3	3 992.5	2 649.4
`netherlands.osm`	2 216 688	2 441 238	5 833.3	5 191.3	3 895.7
`italy.osm`	6 686 493	7 013 978	15 300.8	14 925.2	12 474.8
`great-britain.osm`	7 733 822	8 156 517	20 298.7	19 579.2	12 762.0
`europe.osm`	50 912 018	54 054 660	57 462.7	55 947.2	48 303.6

Table 1: Speedups of FASTCLOS on street networks.

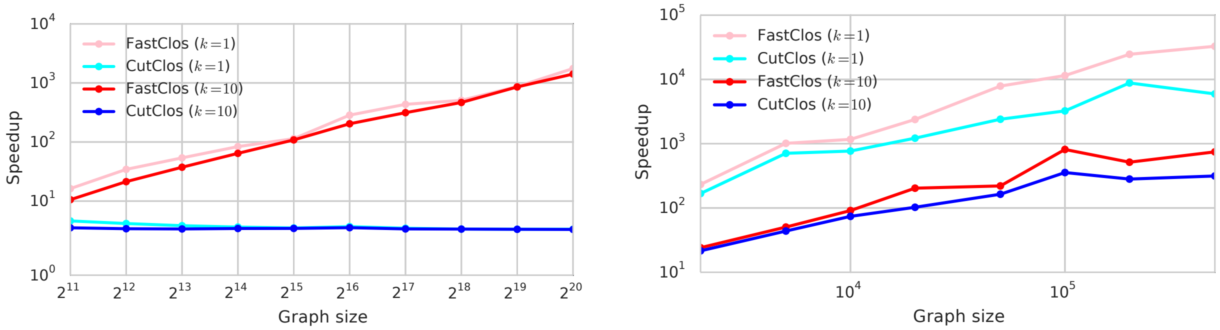

Figure 2: Left: speedups of FASTCLOS and CUTCLOS on Delaunay graphs, with $k = 1$ and $k = 10$, n equal to powers of 2 and $m \approx 3n$. Right: speedups of FASTCLOS and CUTCLOS on random hyperbolic graphs, with n of increasing sizes and $m \approx 10n$.

degree distribution and a very small diameter. Here FASTCLOS$_L$ cannot perform well, because very similar lower bounds are computed for many nodes, due to the small number of levels in the BFS trees. On the other hand, the structure of complex networks allows FASTCLOS$_N$ to compute lower bounds that are close to the the actual closeness values, quickly identifying the top-k nodes. Also CUTCLOS performs very well on complex networks, since here the degree is often related to the closeness of nodes and the small diameter makes it possible to quickly discard nodes that cannot be in the top-k list. Figure 2 (right) shows the speedups of FASTCLOS and CUTCLOS on synthetic complex networks of increasing sizes, created with an efficient generator [22] according to an hyperbolic geometry-based model [13]. This model was shown to reproduce many properties of real complex networks (such as low diameter and power-law degree distribution, see [23] and the references therein). The figure shows that the speedups of both algorithms increase with the size of the graph, with FASTCLOS reaching values larger than 10^4. The speedups of FASTCLOS are always larger than those of CUTCLOS and the gap between the two seems to increase with the size of the graphs. It is also interesting to note that the gap between the number of operations required to find the top node and the top-k nodes is often much larger in complex networks than in street networks. Table 2 summarizes the speedups of FASTCLOS and CUTCLOS on complex undirected networks. We do not report the results of FASTCLOS$_L$ and FASTCLOS$_N$ because the first one does not perform well on complex networks so it would have been too expensive to compute for many of the tested networks and because those of FASTCLOS$_N$ are basically the same as those of the combined algorithm FASTCLOS. The table shows that FASTCLOS is always faster than CUTCLOS. The speedups vary considerably depending on the properties of the networks, but they tend to increase with the size of the network and to be larger in networks with small diameters. Table 3 contains the results on directed networks. Also here FASTCLOS is always faster than CUTCLOS, although the difference between the two is often relatively small.

To summarize, our results show that our combined version FASTCLOS performs very similarly to FASTCLOS$_L$ on street networks and to FASTCLOS$_N$ in complex networks, i. e. it always performs like the fastest of the two methods on the network it is applied to. Also, FASTCLOS is always faster than the currently best existing method CUTCLOS, several orders of magnitude in street networks and up to a factor 18 in complex networks, where CUTCLOS already performs well. In these networks, we are on average 1.7 times faster than CUTCLOS (geometric mean).

				Speedup ($k = 1$)		Speedup ($k = 10$)		Speedup ($k = 100$)	
Graph	Nodes	Edges	Diam	FastClos	CutClos	FastClos	CutClos	FastClos	CutClos
`CA-HepPh`	11 204	117 619	13	**9.7**	8.5	**9.5**	8.4	**8.9**	7.9
`CA-AstroPh`	17 903	196 972	14	**69.8**	26.4	**29.3**	18.3	**13.9**	11.0
`CA-CondMat`	21 363	91 286	14	**493.4**	369.9	**95.5**	76.6	**35.5**	15.5
`Email-Enron`	33 696	180 811	11	**896.1**	225.6	**318.8**	114.9	**38.9**	29.3
`Gowalla-edges`	196 591	950 327	14	**33 086.1**	12 030.7	**33.5**	32.8	**28.2**	26.9
`com-youtube`	1 134 890	2 987 624	20	**2 241.0**	2 060.7	**168.9**	162.1	**110.6**	104.7
`as-skitter`	1 694 616	11 094 209	25	**187.4**	166.1	**167.0**	148.6	**139.8**	116.7

Table 2: Speedups of FastClos and CutClos on complex undirected networks.

				Speedup ($k = 1$)		Speedup ($k = 10$)		Speedup ($k = 100$)	
Graph	Nodes	Edges	Diam	FastClos	CutClos	FastClos	CutClos	FastClos	CutClos
`p2p-Gnutella25`	5 153	17 695	10	**1 166.7**	172.0	**58.8**	34.7	**13.7**	11.9
`p2p-Gnutella24`	6 352	22 928	10	**3 631.9**	198.8	**48.9**	30.5	**12.3**	10.9
`Cit-HepTh`	7 464	116 252	17	**148.9**	79.5	**25.3**	22.0	**19.3**	11.9
`Cit-HepPh`	12 711	139 965	12	**149.4**	129.5	**56.3**	49.9	**30.4**	22.5
`p2p-Gnutella31`	14 149	50 916	11	**197.7**	96.7	**20.5**	17.8	**8.1**	7.7
`Slashdot081106`	26 996	337 351	11	**363.0**	171.5	**131.9**	93.9	**53.2**	40.6
`Slashdot090216`	27 222	342 747	11	**333.3**	165.3	**119.7**	87.5	**54.5**	41.3
`soc-Epinions1`	32 223	443 506	14	**1792.9**	336.2	**53.2**	39.3	**28.8**	22.7
`Email-EuAll`	34 203	151 132	14	**24 833.2**	6 306.1	**3 419.6**	1 348.7	**294.6**	207.0
`twitter-combined`	68 413	1 685 152	7	**204.7**	173.4	**152.5**	110.6	**70.6**	47.9
`Slashdot0811`	70 355	818 310	10	**11 021.8**	2 940.3	**423.0**	279.6	**47.9**	42.8
`Slashdot0902`	71 307	841 201	11	**12 006.4**	2 867.0	**394.4**	262.6	**48.6**	43.3
`WikiTalk`	111 881	1 477 893	9	**6 746.1**	3 276.8	**2 604.4**	1 004.6	**1 026.6**	512.9
`Amazon0302`	241 761	1 131 217	32	**38.7**	20.2	**31.7**	16.1	**13.9**	12.9

Table 3: Speedups of FastClos and CutClos on complex directed networks.

5 Hardness Result

In this section, we analyze the time complexity of computing the vertex with highest closeness centrality in the general case of disconnected graphs. If G is not (strongly) connected, the most common generalization [18, 4] for the closeness of a vertex v is

$$c(v) = \frac{r(v)-1}{f(v)}\frac{r(v)-1}{n-1} = \frac{(r(v)-1)^2}{(n-1)f(v)},$$

where $f(v) = \sum_{w \in R(v)} d(v,w)$, $R(v)$ is the set of vertices reachable from v, and $r(v)$ is $|R(v)|$ (note that $v \in R(v)$ by definition). If a vertex has (out)degree 0, the previous fraction becomes $\frac{0}{0}$: in this case, the closeness is not defined.

We believe that our algorithms can be generalized as well, using the proof techniques in [4]. The main result of this section proves that, at least in this general case, the $O(mn)$ worst-case complexity of the *textbook* algorithm cannot be significantly improved, unless the well-known Strong Exponential Time Hypothesis (SETH) is false [10] (this hypothesis says that the k-Satisfiablility problem cannot be solved in time $O((2-\epsilon)^n)$, where $\epsilon > 0$ does not depend on k).

Theorem 5.1. *On directed graphs, in the worst case, an algorithm computing the most closeness central vertex in time $O(m^{2-\epsilon})$ for some $\epsilon > 0$ would falsify the SETH.*

It is worth mentioning that this result still works if we restrict our analysis to graphs with small diameter (where the diameter is the maximum distance between any two connected nodes). Indeed, the diameter of the graph obtained is 7.

The reduction starts from the l-TwoDisjointSet problem, that is, finding two disjoint sets in a collection $\mathcal{C}$ of subsets of a given ground set X, where $|X| < \log^l(|\mathcal{C}|)$. It is well known that an algorithm solving this problem in $O(|\mathcal{C}|^{2-\epsilon})$, where $\epsilon > 0$ does not depend on l, falsifies SETH [10, 5].

Given an instance $(X, \mathcal{C})$ of the l-TwoDisjointSet problem, and given a set $C \in \mathcal{C}$, let R_C be $|\{C' \in \mathcal{C} : C \cap C' \neq \emptyset\}|$: we want to check if $R_C = |\mathcal{C}|$ for all $C \in \mathcal{C}$; indeed, $R_C = |\mathcal{C}|$ means that C intersects all the sets in $\mathcal{C}$. We will construct a directed graph $G = (V, E)$, where $|V|, |E| = O(|\mathcal{C}||X|) = O(|\mathcal{C}| \log^l |\mathcal{C}|)$, such that:

1. V contains a set of vertices $\mathcal{C}_0$ representing the sets in $\mathcal{C}$;
2. the centrality of a vertex in $\mathcal{C}_0$ corresponding to a set C is a function $c(R_C)$, depending only on R_C, meaning that if $R_C = R_{C'}$ then the vertices corresponding to C and C' have the same closeness;
3. the function $c(R_C)$ is decreasing with respect to R_C;

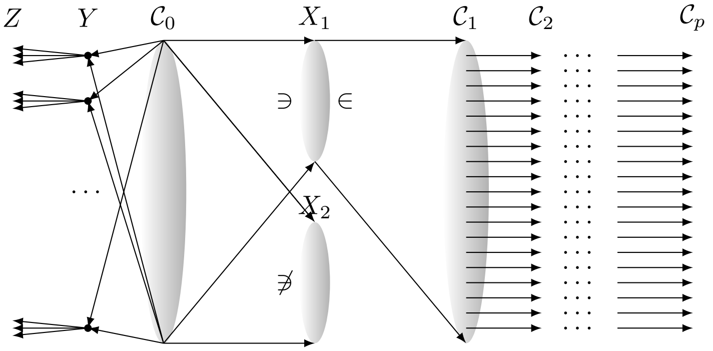

Figure 3: Reducing the TwoDisjointSet problem to the problem of finding the most closeness central vertex.

Algorithm 4: Neighborhood-based lower bound for undirected graphs

```
Input : A graph $G = (V, E)$
Output: Lower bounds $\tilde{S}^{(\mathrm{un})}_{\mathsf{N}}(v)$ of each node $v \in V$
foreach $s \in V$ do
    $\#\mathsf{L}_{k-1}[s] \leftarrow \deg(s)$;
    $\tilde{S}^{(\mathrm{un})}_{\mathsf{N}}[s] \leftarrow \deg(s)$;
    $\mathsf{nVisited}[s] \leftarrow \deg(s) + 1$;
    $\mathsf{finished}[s] \leftarrow false$;
end
$k \leftarrow 2$;
$\mathsf{nFinished} \leftarrow 0$;
while $\mathsf{nFinished} < n$ do
    foreach $s \in V$ do
        if $k = 2$ then
            $\#\mathsf{L}_k[s] \leftarrow \sum_{w \in N(s)} \#\mathsf{L}_{k-1}[w] - \deg(s)$;
        end
        else
            $\#\mathsf{L}_k[s] \leftarrow \sum_{w \in N(s)} \#\mathsf{L}_{k-1}[w] -$
            $\#\mathsf{L}_{k-2}[s](\deg(s) - 1)$;
        end
    end
    foreach $s \in V$ do
        if $\mathsf{finished}[v]$ then
            continue;
        end
        $\#\mathsf{L}_{k-2}[s] \leftarrow \#\mathsf{L}_{k-1}[s]$;
        $\#\mathsf{L}_{k-1}[s] \leftarrow \#\mathsf{L}_k[s]$;
        if $n - \mathsf{nVisited}[s] > \#\mathsf{L}_{k-1}[s]$ then
            $\tilde{S}^{(\mathrm{un})}_{\mathsf{N}}[s] \leftarrow \tilde{S}^{(\mathrm{un})}_{\mathsf{N}}[s] + k \cdot \#\mathsf{L}_{k-1}[s]$;
            $\mathsf{nVisited}[s] \leftarrow \mathsf{nVisited}[s] + \#\mathsf{L}_{k-1}[s]$;
        end
        else
            $\tilde{S}^{(\mathrm{un})}_{\mathsf{N}}[s] \leftarrow \tilde{S}^{(\mathrm{un})}_{\mathsf{N}}[s] + k(n - \mathsf{nVisited}[s])$;
            $\mathsf{nVisited}[s] \leftarrow n$;
            $\mathsf{nFinished} \leftarrow \mathsf{nFinished} + 1$;
            $\mathsf{finished}[s] \leftarrow true$;
        end
    end
    $k \leftarrow k + 1$;
end
return $\tilde{S}^{(\mathrm{un})}_{\mathsf{N}}$
```

4. the centrality of vertices in $\mathcal{C}_0$ is bigger than the centrality of all the other vertices.

In such a graph, the maximum closeness is $c(|\mathcal{C}|)$ if and only if $R_C = |\mathcal{C}|$ for each C, if and only if $\mathcal{C}$ does not contain two disjoint sets. Hence, an algorithm finding the maximum closeness in time $O(m^{2-\epsilon})$ yields an algorithm solving the l-TwoDisjointSet problem in $O((|\mathcal{C}| \log^l |\mathcal{C}|)^{2-\epsilon}) = O(|\mathcal{C}|^{2-\frac{\epsilon}{2}})$, against SETH.

To construct this graph (see Figure 3), we start by adding to V the copy $\mathcal{C}_0$ of $\mathcal{C}$, another copy $\mathcal{C}_1$ of $\mathcal{C}$ and a copy X_1 of X, and by adding to E the edges (C, x) for each $C \in \mathcal{C}_0, x \in X_1$ such that $x \in C$, and (x, C) for each $x \in X_1, C \in \mathcal{C}_1$ such that $x \in \mathcal{C}$. Moreover, we add a copy X_2 of X and we connect all pairs (C, x) with $C \in \mathcal{C}_0$, $x \in X_2$ and $x \notin C$: the closeness centrality of a vertex $C \in \mathcal{C}_0$ is $\frac{(|X|+R_C)^2}{(n-1)(|X|+2R_C)}$ (which only depends on R_C). To enforce Items 3 and 4, we add a path of length p leaving each vertex in $\mathcal{C}_1$, and q vertices linked to each vertex in $\mathcal{C}_0$, each of which has out-degree $|\mathcal{C}|$ (we will show that by setting $p = 7$ and $q = 36$, all required conditions are satisfied).

More formally, given a ground set X and a collection $\mathcal{C}$ of subsets of X, i.e. an instance of the l-TwoDisjointSet problem, we build the graph $G = (V, E)$ as follows.

- $V = Z \cup Y \cup \mathcal{C}_0 \cup X_1 \cup X_2 \cup \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_p$, where Z is a set of cardinality $q|\mathcal{C}|$, Y a set of cardinality q, the $\mathcal{C}_i$s are copies of $\mathcal{C}$ and the X_is are copies of X;

- each vertex in Y have $|\mathcal{C}|$ neighbors in Z, and these neighbors are disjoint;

- for each $x \in C$, there are edges from $C \in \mathcal{C}_0$ to $x \in X_1$, and from $x \in X_1$ to $C \in \mathcal{C}_1$;

- for each $x \notin C$, there is an edge from $C \in \mathcal{C}_0$ to $x \in X_2$;

- each $C \in \mathcal{C}_i$, $1 \leq i \leq p$, is connected to the same set in $\mathcal{C}_{i+1}$;
- no other edge is present in the graph.

Note that, under the assumption $|X| < \log^l(|\mathcal{C}|)$, the number of edges in this graph is $O(|\mathcal{C}|\log^l(|\mathcal{C}|))$.

LEMMA 5.1. *Assuming $|\mathcal{C}| > 1$, all vertices outside $\mathcal{C}_0$ have closeness centrality at most $\frac{2|\mathcal{C}|}{n-1}$, where n is the number of vertices.*

Proof. If a vertex is in Z, X_2, or $\mathcal{C}_p$, its closeness centrality is not defined, because it has out-degree 0.

A vertex $y \in Y$ reaches $|\mathcal{C}|$ vertices in 1 step, and hence its closeness centrality is $\frac{|\mathcal{C}|^2}{|\mathcal{C}|(n-1)} = \frac{|\mathcal{C}|}{n-1}$.

A vertex in $\mathcal{C}_i$ reaches $p-i$ other vertices, and their distance is $1, \ldots, p-i$: consequently, its closeness centrality is $\frac{(p-i)^2}{\frac{(p-i)(p-i+1)}{2}(n-1)} = \frac{2(p-i)}{(n-1)(p-i+1)} \leq \frac{2}{n-1}$.

Finally, for a vertex $x \in X_1$ contained in N_x sets, for each $1 \leq i \leq p$, x reaches N_x vertices in $\mathcal{C}_i$, and these vertices are at distance i. Hence, the closeness of x is $\frac{(pN_x)^2}{\frac{p(p+1)}{2}N_x(n-1)} = \frac{2pN_x}{(n-1)(p+1)} \leq \frac{2N_x}{n-1} \leq \frac{2|\mathcal{C}|}{n-1}$. This concludes the proof. □

Let us now compute the closeness centrality of a vertex $C \in \mathcal{C}_0$. The reachable vertices are:

- all q vertices in Y, at distance 1;
- all $|\mathcal{C}|q$ vertices in Z, at distance 2;
- $|X|$ vertices in X_1 or X_2, at distance 1;
- R_C vertices in $\mathcal{C}_i$ for each i, at distance $i+1$.

Hence, the closeness centrality of C is: $c(R_C) = \frac{(q(1+|\mathcal{C}|)+|X|+pR_C)^2}{\left(q(1+2|\mathcal{C}|)+|X|+\left(\frac{(p+1)(p+2)}{2}-1\right)R_C\right)(n-1)} = \frac{(q(1+|\mathcal{C}|)+|X|+pR_C)^2}{(q(1+2|\mathcal{C}|)+|X|+g(p)R_C)(n-1)}$ if $g(p) = \frac{(p+1)(p+2)}{2} - 1$. We want to choose p and q verifying:

1. the closeness of vertices in $\mathcal{C}_0$ is bigger than $\frac{2|\mathcal{C}|}{n-1}$ (and hence bigger than the closeness of all other vertices);
2. $c(R_C)$ is a decreasing function of R_C for $0 \leq R_C \leq |\mathcal{C}|$.

If we verify these two conditions, the most central vertex has closeness $c(|\mathcal{C}|)$ if and only if, for all sets $C \in \mathcal{C}$, $R_C = |\mathcal{C}|$, if and only if there are not two disjoint sets in $\mathcal{C}$. Otherwise, the closeness centrality of the most central vertex is smaller. This way, by computing the maximum closeness centrality in time $O(m^{2-\epsilon})$, we would be able to compute if there are two disjoint sets in $O((|\mathcal{C}|\log^l(|\mathcal{C}|))^{2-\epsilon}) = O(|C|^{2-\frac{\epsilon}{2}})$, against SETH.

It only remains to find these values of p, q: for Item 2, the derivative $c'(R_C)$ of c is $(q(1+|\mathcal{C}|)+|X|+pR_C)\frac{[pg(p)R_c+2p(q(1+2|\mathcal{C}|)+|X|)-g(p)(q(1+|\mathcal{C}|)+|X|)]}{(q(1+2|\mathcal{C}|)+|X|+g(p)R_C)^2(n-1)}$.

This latter value is negative if and only if $pg(p)R_c + 2p\,(q(1+2|\mathcal{C}|)+|X|) - g(p)(q(1+|\mathcal{C}|)+|X|) < 0$. Assuming $g(p) \geq 5p$ and $R_C < |\mathcal{C}|$, this value is:

$$\begin{aligned}
&pg(p)R_C + 2p\,(q(1+2|\mathcal{C}|)+|X|) - g(p)(q(1+|\mathcal{C}|)+|X|) \\
&\leq pg(p)|\mathcal{C}| + 2pq + 4pq|\mathcal{C}| + 2p|X| - g(p)(q-|C|-|X|) \\
&\leq pg(p)|\mathcal{C}| + 4pq|\mathcal{C}| - g(p)q|\mathcal{C}| \\
&\leq pg(p)|\mathcal{C}| - pq|\mathcal{C}|.
\end{aligned}$$

Assuming $q > g(p)$, we conclude that $c'(R_C) < 0$ for $0 \leq R_C \leq |\mathcal{C}|$, and we verify Item 2. In order to verify Item 1, we want $c(R_C) \geq \frac{2|\mathcal{C}|}{n+1}$ (since $c(R_C)$ is decreasing, it is enough $c(|\mathcal{C}|) \geq \frac{2|\mathcal{C}|}{n+1}$). Under the assumptions $q > g(p)$, $0 < |X| \leq |\mathcal{C}|$ (which trivially holds for $|\mathcal{C}|$ big enough, because $|X| \leq \log^p|\mathcal{C}|$),

$$\begin{aligned}
c(|\mathcal{C}|) &= \frac{(q(1+|\mathcal{C}|)+|X|+pR_C)^2}{(q(1+2|\mathcal{C}|)+|X|+g(p)R_C)\,(n-1)} \\
&\geq \frac{q^2|\mathcal{C}|^2}{(q(3|\mathcal{C}|)+|\mathcal{C}|+|\mathcal{C}|)(n-1)} \\
&\geq \frac{q|\mathcal{C}|}{5(n-1)} > \frac{2|\mathcal{C}|}{n-1}
\end{aligned}$$

if $q > 10$.

To fulfill all required conditions, it is enough to choose $p = 7$, $g(p) = 35$, and $q = 36$.

6 Conclusions

In this paper we have presented new methods for finding the k nodes with highest closeness centrality in a network, a problem of high practical relevance in network analysis. By finding lower bounds on the inverse closeness of each node, we limit the exact computation to a small fraction of vertices. More precisely, we propose two lower bounds, one that has shown to work better on networks with relatively large diameter and degree distribution with small variance (e. g., street networks) and one that is more performant on complex networks. The combination of the two approaches into one method is orders of magnitude faster than computing closeness for all nodes in our experiments, on every tested network. Compared with the state of the art, our combined approach is always faster than the currently best algorithm for top-k closeness centrality [4]. Thanks to our new approach, we are able to find the top-10 nodes with highest closeness in the whole European street network (with 54 millions edges) in minutes (where exhaustive computation would take years).

Our results show that our bounds can significantly reduce the number of edges that we have to visit to compute the k nodes with highest closeness. Not surprisingly, the best speedups are visible for small values of k and the improvement decreases as k becomes a larger fraction of n. We also tested our algorithms with larger values of k (e. g. $k = 10^3$ and $k = 10^4$) and in this case our running times become much closer to those of the exhaustive algorithm. In this regard, it is important to notice that the speedup can never be larger than n/k, since at least k BFSs are needed to compute the closeness of the top-k nodes. For this reason, our algorithms are designed for scenarios where the user is only interested in finding the really central nodes of a network.

Future work may include the extension of our technique and of those presented in [4] to weighted and disconnected graphs, for which the computation of a lower bound is the only obstacle. Also, it would be interesting to investigate whether our concepts can be extended to other centrality measures, such as betweenness, and whether they can be used to compute a fast approximation of closeness in very large networks. We also plan to run our code in parallel to further accelerate the computations. Our implementation will be made publicly available as part of a future release of the network analysis tool suite *NetworKit* [20].

Acknowledgements. We would like to thank Moritz von Looz for providing the hyperbolic random graphs used in our experiments and numerous contributors to the *NetworKit* project.

We also would like to thank Leonardo Mezzina, who gave us the access to a server for the experiments.

References

[1] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 1681–1697. SIAM, 2015.

[2] David A. Bader, Henning Meyerhenke, Peter Sanders, Christian Schulz, Andrea Kappes, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, pages 73–82. Springer, 2014.

[3] David C. Bell, John S. Atkinson, and Jerry W. Carlson. Centrality measures for disease transmission networks. *Social Networks*, 21(1):1–21, 1999.

[4] Michele Borassi, Pierluigi Crescenzi, and Andrea Marino. Fast and simple computation of top-k closeness centralities. `http://arxiv.org/abs/1507.01490`, 2015.

[5] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the Square - On the Complexity of Some Quadratic-Time Solvable Problems. `http://arxiv.org/abs/1407.4972`, 2014.

[6] Ulrik Brandes and Christian Pich. Centrality estimation in large networks. *I. J. Bifurcation and Chaos*, 17(7):2303–2318, 2007.

[7] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F. Werneck. Computing classic closeness centrality, at scale. *Proceedings of the 2nd ACM conference on Online social networks, COSN 2014*, pages 37–50. ACM, 2014.

[8] Benjamin Cornwell. A complement-derived centrality index for disconnected graphs. *Connections*, 26(2):70–81, 2005.

[9] David Eppstein and Joseph Wang. Fast approximation of centrality. *J. Graph Algorithms Appl.*, 8:39–45, 2004.

[10] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[11] Christine Kiss and Martin Bichler. Identification of influencers – measuring influence in customer networks. *Decision Support Systems*, 46(1):233 – 253, 2008.

[12] Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, Stefan Richter, Dagmar Tenfelde-Podehl, and Oliver Zlotowski. Centrality indices. *Network Analysis*, volume 3418 of *Lecture Notes in Computer Science*, pages 16–61. Springer Berlin Heidelberg, 2005.

[13] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82:036106, Sep 2010.

[14] Yeon-sup Lim, Daniel S Menasché, Bruno Ribeiro, Don Towsley, and Prithwish Basu. Online estimat-

ing the k central nodes of a network. In *IEEE Network Science Workshop (NSW)*, 2011.

[15] Erwan Le Merrer, Nicolas Le Scouarnec, and Gilles Trédan. Heuristical top-k: fast estimation of centralities in complex networks. *Information Processing Letters*, 114(8):432–436, 2014.

[16] Mark Newman. *Networks: An Introduction.* Oxford University Press, 2010.

[17] Kazuya Okamoto, Wei Chen, and Xiang-Yang Li. Ranking of closeness centrality for large-scale social networks. *Frontiers in Algorithmics, 2nd Annual International Workshop, FAW 2008*, volume 5059 of *Lecture Notes in Computer Science*, pages 186–195. Springer, 2008.

[18] Paul W. Olsen, Alan G. Labouseur, and Jeong-Hyon Hwang. Efficient top-k closeness centrality search. *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014*, pages196–207. IEEE, 2014.

[19] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32(3):245–251, July 2010.

[20] Christian Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: An interactive tool suite for high-performance network analysis. `http://arxiv.org/abs/1403.3005`, 2014.

[21] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32:425–443, 1969.

[22] Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. Generating random hyperbolic graphs in subquadratic time. *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC)*, LNCS. Springer, 2015. To appear.

[23] Moritz von Looz, Christian L. Staudt, Henning Meyerhenke, and Roman Prutkin. Fast generation of dynamic complex networks with underlying hyperbolic geometry. `http://arxiv.org/abs/1501.03545`, 2015.

[24] Duncan J. Watts. *Small worlds : the dynamics of networks between order and randomness.* Princeton University Press, 2003.

[25] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 887–898. ACM, 2012.