Karlsruhe Reports in Informatics 2017,8
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Scaling up Group Closeness Maximization

Elisabetta Bergamini, Tanya Gonser, , Henning Meyerhenke

2017

Fakultät für Informatik

Scaling up Group Closeness Maximization*

Elisabetta Bergamini[1], Tanya Gonser[1], and Henning Meyerhenke[1]

1 **Karlsruhe Institute of Technology (KIT), Germany**
{elisabetta.bergamini, tanya.gonser, meyerhenke}@kit.edu

Abstract

Closeness is a widely-used centrality measure in social network analysis. For a node it indicates the inverse average shortest-path distance to the other nodes of the network. While the identification of the k nodes with highest closeness received significant attention, many applications are actually interested in finding a *group* of nodes that is central as a whole. For this problem, only recently a greedy algorithm with approximation ratio $(1 - 1/e)$ has been proposed [Chen et al., ADC 2016]. Since this algorithm's running time is still expensive for large networks, a heuristic without approximation guarantee has also been proposed in the same paper.

In the present paper we develop new techniques to speed up the greedy algorithm without losing its theoretical guarantee. Compared to a straightforward implementation, our approach is orders of magnitude faster and, compared to the heuristic proposed by Chen et al., we always find a solution with better quality in a comparable running time in our experiments.

Our method Greedy++ allows us to approximate the group with maximum closeness on networks with up to hundreds of millions of edges in minutes or at most a few hours. To have the same theoretical guarantee, the greedy approach by [Chen et al., ADC 2016] would take several days already on networks with hundreds of thousands of edges.

In a comparison with the optimum, our experiments show that the solution found by Greedy++ is actually much better than the theoretical guarantee. Over all tested networks, the empirical approximation ratio is never lower than 0.97.

Finally, we study for the first time the correlation between the top-k nodes with highest closeness and an approximation of the most central group in large complex networks and show that the overlap between the two is relatively small.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph algorithms, network analysis, closeness centrality, group centrality

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

One of the main tasks in social network analysis is the identification of important nodes. For this reason, several centrality measures have been introduced over the years and much work has been put into their efficient computation. Closeness centrality is a widely-used measure that ranks the nodes according to their inverse average shortest-path distance to the other nodes. Intuitively, a node with high closeness is a node that is close, on average, to the other nodes of the network and can therefore reach them quickly. In their seminal work, Borgatti and Everett [14] extended the concept of centrality to *groups* of nodes. For a node v and a group S of other nodes, the distance between v and S is defined as the minimum distance between v and the elements of S. Then, a group of nodes has high closeness when its average

* This work was partially supported by DFG SPP 1736 *Algorithms for Big Data*.

distance to the other nodes is small. Finding central groups of nodes is an important task for many applications. For example, in social networks, retailers might want to select a group of nodes as promoters of their product, in order to maximize the spread among users [17]. In this context, picking the k most central nodes might lead to a large overlap in the set of influenced nodes, whereas there might be k nodes that are not among the most central when considered individually, but that influence different areas of the graph. Closely related to finding the group with highest closeness is p-median, a fundamental facility location problem in operations research [15]. One can see Group Closeness Maximization (GCM) as a special case of p-median: the standard GCM formulation applies only to graphs without vertex weights, whereas p-median also applies to geometric inputs and weighted objects (to name only few of the possible generalizations [12]). For p-median, several heuristics, metaheuristics and approximation algorithms have been proposed over the years (see [24] for an annotated bibliograohy). However, these methods are mostly applicable to relatively small networks only. In [23], the authors compare state-of-the-art methods on a street network of Sweden ($\approx$ 190K nodes) and show that existing methods either fail because of their memory requirements (>32 GB) or take more than 14 hours to find an approximation. Other recent methods have been shown to scale to inputs with up to 90 000 points/nodes [2, 16].

Specifically for GCM, an $(1-1/e)$-approximation algorithm has been proposed recently by Chen et al. [9]. Unfortunately, the algorithm is not scalable to graphs with more than about 10^4 vertices, since it requires to compute pairwise distances. Thus, Chen et al. proposed in the same paper also a more scalable heuristic without guarantees on the solution quality.

1.1 Outline and contribution

We present techniques that can reduce considerably the memory and the number of operations required by the greedy algorithm presented in [9], without losing its theoretical guarantee on the quality of the approximation. First, instead of computing and storing all pairwise distances, we use the algorithm presented in [3] to find the node with maximum closeness (Section 3.2). Then, we reduce the subsequent computations using pruned single-source shortest paths (Section 4.1) and exploiting the submodularity of the objective function (Section 4.2). In our experiments in Section 6, we compare our algorithm (Greedy++) with the greedy approach presented in [9] and show that Greedy++ is orders of magnitude faster. Also, we compare Greedy++ with the heuristic proposed in [9] and show that Greedy++ is often faster (or has a comparable running time) and that it always finds a better solution in all our experiments. We also provide an Integer Linear Programming (ILP) formulation of the GCM problem in Section 5 and compare the quality of our solution with the optimum. Our results show that the solution found by Greedy++ is actually much better than the theoretical guarantee and the empirical approximation ratio is never lower than 0.97. Finally, we study the overlap between the group with maximum closeness and the k nodes with highest closeness and highest degree in real-world networks, showing that in most cases this is relatively small (between 30% and 60% of the group size). This confirms the intuition that a central group of nodes is not necessarily composed of nodes that are individually central.

2 Preliminaries

We model a network as a graph $G = (V, E)$ with $|V| =: n$ nodes and $|E| =: m$ edges. Unless stated explicitly, we assume the graph to be connected (or, if directed, strongly connected) and unweighted. Let $d(u, v)$ represent the shortest-path distance between node u and node v. We define the distance between $u \in V$ and a set $S \subseteq V$ of nodes as $d(u, S) := \min_{s \in S} d(u, s)$.

Then, the closeness centrality of node u is defined as $c(u) := \frac{n-1}{\sum_{v \neq u} d(u,v)}$. Similarly, we can define the closeness of a set S as $c(S) := \frac{n-|S|}{\sum_{v \notin S} d(S,v)}$. The Group Closeness Maximization (GCM) problem is defined as finding a set $S^\star \subseteq V$ of a given size k, with maximum group closeness: $S^\star = \arg\max_{S \subseteq V}\{c(S) : |S| = k\}$. Note that an adaptation of our approximation algorithms to the case of $|S| \leq k$ is rather straightforward. In the paper we use SSSP to denote a single-source shortest path computation, i. e., breadth-first search (BFS) for unweighted graphs. We use APSP to denote an all-pairs shortest path distance computation.

3 Related work

Computing closeness centrality requires the distances between all pairs of nodes. For this problem one typically solves a SSSP from each node or uses fast matrix multiplication. In both cases the time required is at least quadratic in the number of nodes. For this reason, several approximation algorithms for closeness centrality have been proposed [13, 7, 10, 8]. The basic idea is to sample a set of nodes (pivots), compute the distance between the pivots and the other nodes and then estimate the closeness scores of all nodes using the computed distances. Although these algorithms can often approximate the scores well, they may fail at preserving the ranking of nodes, in particular for those with similar closeness values. In [4], it has been shown that the algorithm by Chechik et al. [8] would require n^2 SSSP computations to guarantee an exact ranking in complex networks, which is clearly impractical.

For this reason, recently *exact* algorithms for finding the k nodes with maximum closeness have been proposed [3, 4, 6, 22]. The authors of [6] propose an algorithm whose worst-case complexity is the same as that of running a SSSP from each node. Nevertheless, they show that their approach is very scalable in practice and that it outperforms all existing approximation and exact methods. Subsequently, the algorithm presented in [6] has been further improved in [3] and extended in [4]. Since we use this algorithm to solve a subtask of our greedy approach for group closeness maximization, we describe it in Section 3.1.

GCM has been very recently considered in [9], where the authors show that finding the group with maximum closeness is an NP-hard problem. Also, they propose a greedy algorithm and prove that the solution found by the algorithm is at most a factor $(1 - 1/e)$ away from the optimum. Since the greedy algorithm is still expensive (its complexity is $\Theta(kn^2)$ plus the cost of an APSP, for a group of size k), the authors propose an alternative heuristic based on sampling. In particular, they first propose a baseline heuristic (BSA), which basically samples a set of nodes and then selects iteratively the node that minimizes the distance of the current solution to the samples. Then, they show that the running time of BSA can be improved by dividing the set of samples in partitions (and they call this second heuristic Order-based Sampling Algorithm, OSA). However, the two heuristics do not have the theoretical guarantee of the greedy algorithm, so we cannot know how well they approximates the optimum. Since the algorithm proposed in this paper builds on the greedy algorithm of [9], we describe it in more detail in Section 3.2.

In [28], an algorithm for computing and maximizing group closeness on disk-resident graphs has been proposed. The basic idea is to estimate the closeness of a group using the nodes at distance at most H from the group (where H can be any integer value greater than 0). Although they show that their approach can scale quite well for small values of H, there is no guarantee on how close their estimation is to the real centrality of the group. The problem of finding a central group of nodes has also been considered for betweenness centrality, for which sampling-based approximation algorithms have been proposed [20, 27].

3.1 Top-k closeness algorithm

The basic idea of the top-k closeness algorithm for complex networks proposed in [3] can be summarized as follows: Let us assume we want to find the k nodes with highest closeness centrality. Also, assume we have an upper bound $\tilde{c}(v)$ on the closeness of a node v. Then, if k nodes exist such that their exact closeness is higher than the upper bound $\tilde{c}(v)$, we know that v is not one of the k nodes with highest closeness and we do not need to compute its exact closeness $c(v)$. The algorithm is summarized in Algorithm 2 in the appendix. At each iteration, x_k contains the k-th highest closeness value found so far. Function `BFScut`(v, x_k) in Line 4 computes iteratively an upper bound on the closeness of v in the following way: A BFS rooted in v is initiated. After all nodes up to a certain distance d from v have been visited, we know that all remaining nodes are *at least* at distance $d+1$. If we assume that all the unvisited nodes are *exactly* at distance $d+1$, this gives us an upper bound $\tilde{c}_d(v)$ on the closeness of v for each possible distance value d. Therefore, at each step of the BFS rooted in v, we can compare $\tilde{c}_d(v)$ with x_k. If $x_k \geq \tilde{c}_d(v)$, then we can interrupt the BFS and return 0, meaning that v is not one of the top-k nodes. Otherwise, a whole BFS is computed for v (and `BFScut` returns the exact closeness of v). Function `Kth`(c) in Line 6 returns the k-th largest element of c and `TopK`(c) in Line 9 returns the k largest elements of c.

Notice that the described algorithm refers to unweighted graphs (we use the same notation as in [3]). However, it can be easily extended to weighted graphs by substituting the BFS with Dijkstra and computing the upper bound based on the current node v we are visiting (we know that the distance of all remaining nodes is at least as large as the distance of v). In [3], some improvements on Algorithm 2 are proposed for unweighted graphs only. In particular, the idea is to compute upper bounds on the closeness of each node in a pre-processing phase and then process the nodes according to these bounds instead of based on their degree. For more details, we refer the reader to [3].

3.2 Greedy approximation algorithm

Chen et al. [9] proposed a greedy approximation algorithm (Greedy) for group closeness. We recall that the objective is to find a set $S^\star$ such that $S^\star = \arg\max_{S \subseteq V}\{c(S) : |S| = k\}$.

Greedy runs k iterations, after which it returns a set S. For each iteration, Greedy adds to the set S the node u with the largest marginal gain $c(S \cup \{u\}) - c(S)$. Since the objective function c is monotone and submodular (as proven in [9]), Greedy provides a $(1-1/e)$-approximation for the GCM problem, i. e. $c(S) \geq (1-1/e)c(S^\star)$. Algorithm 3 in the appendix shows the pseudocode of Greedy. In Line 2 the pairwise distances are computed and stored in the $n \times n$ matrices d and M. At each iteration, d always contains the pairwise distances, whereas M contains, for each node pair (u, w), the distance $d(S \cup \{u\}, w)$. Initially $d = M$, since $S = \emptyset$. Then, every time a node s is added to S, M is updated in Line 10. *Score* contains $c(S \cup \{u\})$ for each node u, which is computed in Line 13 by summing over $M(u, w)$, $\forall w \in V$.

Since it needs to store two $n \times n$ matrices, the memory requirement of Greedy is $\Theta(n^2)$. The running time is $\Theta(n(m + n\log n))$ for the initial APSP computation (when running a SSSP from each node in a weighted graph) and then $\Theta(kn^2)$ for the remaining part.

4 A scalable greedy algorithm

First of all, we notice that we can reduce the memory requirement of Greedy by not storing the matrices d and S. In fact, to find the first element s_0 of S (i. e. the node with maximum closeness) we can simply use the TopKCloseness algorithm described in Section 3.1. Then we can use a vector d_S containing, for each node v, the distance between S and v. Since initially S is composed of only one element s_0, d_S simply contains the distances between s_0 and the other nodes, which can be computed with a SSSP rooted in s_0. Then, Lines 8 - 10 can be replaced with a SSSP rooted in u where we sum, over each node w visited in the SSSP, the minimum between $d_S(w)$ and $d(u, w)$. This sum is exactly the same as $\sum_{w \in V \setminus S} M[u, w]$ and can therefore be used in Line 13 to update $Score[u]$. The memory-efficient version of Greedy is described in Algorithm 1. In the pseudocode we report explicitly every time we need to run a SSSP. In Line 3 and Line 13, the SSSP is needed to compute d_S, whereas in Line 7 we need it to compute $Score[u]$.

Since we have to re-run a SSSP for each node u and for each element of S other than s_0, the running time complexity of the while loop of Algorithm 1 is $O(kn(m + n \log n))$ (for weighted graphs). The worst-case complexity of finding s_0 with `TopCloseness` is the same as that of an APSP (i. e. $n(m + n \log n)$), although in practice it was shown to be basically linear in the size of the graph [4]. For unweighted graphs, the complexity of Algorithm 1 is $O(knm)$, since we can use BFS instead of Dijkstra to compute the SSSPs.

Although the memory requirement is now only $\Theta(n)$ (in addition to the memory required to store the graph), the time complexity is too high to target large networks. For this reason, in the following we propose improvements that, as we will see in Section 6, increase the scalability of Greedy considerably.

Algorithm 1: Memory-efficient greedy algorithm.

```
Input  : A graph $G = (V, E)$, a number $k$
Output : A set $S$ of nodes of size $k$ such that $c(S) \geq (1 - 1/e)c(S^\star)$
1  $s_0 \leftarrow$ TopKCloseness(1);
2  $S \leftarrow \{s_0\}$;
3  SSSP($s_0$);
4  $d_S[u] \leftarrow d(s_0, u) \quad \forall u \in V$;
5  while $|S| < k$ do
6      foreach $u \in V \setminus S$ do
7          SSSP($u$);
           /* $Score[u]$ is set to $c(S \cup \{u\})$ */
8          $t \leftarrow \sum_{w \in V \setminus S} \min\{d(u, w),\ d_S[w]\}$;
9          $Score[u] \leftarrow (n - |S| - 1)/t$;
10     end
11     $s \leftarrow \arg\max_{w \in V \setminus S} Score[w]$;
12     $S \leftarrow S \cup \{s\}$;
13     SSSP($s$);
14     foreach $u \in V$ do
15         $d_S[u] \leftarrow \min\{d_S[u],\ d(s, u)\}$;
16     end
17 end
18 return $S$;
```

4.1 Pruned SSSP

In Line 7 of Algorithm 1, we need to run a SSSP rooted in u to recompute $Score[u]$. However, the only nodes w for which we need to compute $d(u, w)$ are those for which $d(u, w) < d_S[w]$. Indeed, for all the other nodes, the distance from u does not contribute to the sum in Line 8 and therefore to $Score[u]$. Thus, if we know that $d(u, w)$ is larger than or equal to $d_S[w]$, we do not need to visit w in the SSSP. It is not hard to see that, if $d(u, w) \geq d_S[w]$, then the same holds for *all the nodes* in the SSSP subtree rooted in w. In fact, let t be a node in the SSSP subtree of w, i. e. $d(u,t) = d(u,w) + d(w,t)$. There is a path between a node in S and t going through w of length $d_S[w] + d(w,t)$. Therefore $d_S[t] \leq d_S[w] + d(w,t) \leq d(u,t)$. Figure 1 illustrates this concept. This allows us to *prune* the SSSP when we find a node whose distance from u is not smaller than its distance from S. When we visit a new node w, we compare $d(u,w)$ with $d_S[w]$. If the first is not strictly smaller than the second, we do not enqueue its neighbors into the SSSP (priority) queue. Notice that, since only nodes u for which $d_S[U] \leq d(s,u)$ are pruned, the value of $d_S[u]$ in Line 15 is not affected, for any $u \in V$. This means that the solution returned by the improved algorithm is exactly the same as the solution returned by Algorithm 1 (which is also the same solution returned by Algorithm 3 in the appendix).

4.2 Submodularity improvement

A function f is submodular whenever $f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T)$, for $S \subseteq T$. It is not hard to see that the closeness c of a set is a submodular function [9]. We can use this property to reduce the number of evaluations of *Score* (and SSSP computations) in Lines 7 - 9. Let us name S_i the set S computed by Algorithm 1 in the i-th iteration of the while loop (S_i is the set composed of i elements). Since $S_i \subseteq S_{i+1}$, because of submodularity $c(S_i \cup \{u\}) - c(S_i) \geq c(S_{i+1} \cup \{u\}) - c(S_{i+1})$. The difference $c(S_i \cup \{u\}) - c(S_i)$ is then the *marginal gain* $\Delta_i(u)$ of u with respect to S_i.

In other words, we can say that at each iteration of the while loop in Algorithm 1, the marginal gain of each node can only decrease. Now, let us assume that there is a node s whose marginal gain $\Delta_i(s)$ with respect to S_i is larger than the marginal gain $\Delta_{i-1}(u)$ of a node u in the previous iteration. This means that the marginal gain of u at iteration i cannot be larger than $\Delta_i(s)$ (since $\Delta_i(u) \leq \Delta_{i-1}(u) \leq \Delta_i(s)$). This allows us to skip the computation of the score of u in Lines 7 - 9. All we need to do is keep track of the marginal gain of each node in the previous iteration and compare it with the maximum marginal gain found in the current iteration (notice that the node with the maximum marginal gain is also the one with the maximum score, since the marginal gain is simply the score of $S \cup \{u\}$ minus the centrality of S, which does not depend on u).

Intuitively, it is important that a node with high marginal gain is found early in the for loop in Line 6, since this allows us to skip the computation of the score of many other nodes. To do this, we keep the nodes sorted by the last marginal gain that was computed for them and then interrupt the `for` loop as soon as we find a node u whose previous marginal gain is smaller than the best marginal gain of the current iteration (since the nodes are sorted, we can skip all the nodes following u).

Notice that this improvement is compatible with the pruned SSSP improvement proposed in the previous section. For the nodes that cannot be skipped because of what was described in this section, we compute their score with a pruned SSSP.

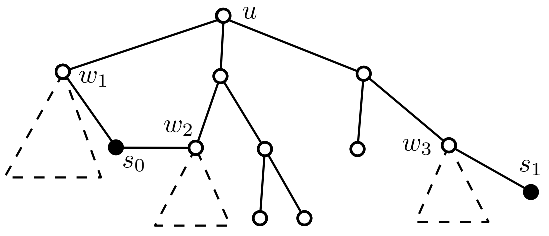


Figure 1 Pruned SSSP. If a node w is such that $d_S[w] \leq d(u,w)$, the same holds for the whole SSSP subtree rooted in w. In the figure, black nodes represent elements of S.

We name our version of Algorithm 1 using pruned SSSPs and the submodularity improvement Greedy++. As explained in Section 4.1, using pruned SSSPs does not affect the solution found by the algorithm. The improvement described in this section might return a different solution only in case there are nodes with the same marginal gain. Indeed, if there are two nodes u and v with the same marginal gain $\Delta^\star$ and such that $\Delta^\star \geq \Delta(w)\ \forall w \in V$, whether we choose u or v depends on which comes first in the ordering of the nodes. Nevertheless, this does not influence the guarantee on the quality of the approximation.

Then, the following theorem holds.

▶ **Theorem 1.** *Let $S \subseteq V$, $|S| = k$, be the solution returned by Greedy++. Then, it holds that $c(S) \geq (1 - 1/e)c(S^\star)$, where $S^\star = \arg\max_{S \subseteq V}\{c(S) : |S| = k\}$.*

5 ILP formulation of group closeness

To evaluate the quality of the solution found by Greedy++, we want to know how far it is from the optimum. Computing the closeness centrality of all possible subsets of size k would clearly be prohibitive even for tiny networks. Hence, we formulate MGC as an ILP problem. This will be used in the experiments in Section 6.1.

For each node $v_j \in V$, we define a binary variable y_j, which is 1 if node v_j is part of the group with maximum closeness $S^\star$, and it is equal to 0 otherwise. We say a node v_i is *assigned* to a node $v_j \in S^\star$ if $d(v_i, S^\star) = d(v_i, v_j)$. If there are multiple nodes $v_j \in S^\star$ that satisfy this property, v_i can be arbitrarily assigned to one of them. Thus, we also define a variable x_{ij} that, for each node pair (v_i, v_j) is equal to 1 if $v_j \in S^\star$ and v_i is assigned to v_j, and 0 otherwise. We can rewrite our problem in the following form:

$$\max \frac{n-k}{\sum_{i=1}^{n}\sum_{j=1}^{n} d(v_i, v_j)x_{ij}} \tag{1}$$

s.t.: $(i)\ \sum_{j=1}^{n} x_{ij} = 1,\ \forall i \in \{1,...,n\}$; $(ii)\ \sum_{j=1}^{n} y_j = k$; $(iii)\ x_{ij} \leq y_j,\ \forall i,j \in \{1,...,n\}$.

with $x_{ij}, y_j \in \{0,1\}$. Condition (i) indicates that each node in $v_i \in V$ is assigned to exactly one node in $v_j \in S^\star$, (ii) indicates that $|S^\star| = k$ and (iii) indicates that nodes v_i are only assigned to nodes v_j that are in $S^\star$, i. e. nodes for which $y_j = 1$. Since the numerator in Eq. (1) is constant, we can rewrite Eq. (1) as:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} d(v_i, v_j)x_{ij}, \tag{2}$$

which gives us an ILP formulation.

6 Experiments

In the following, we present experimental results concerning several aspects of the algorithm described in Section 4 (which we refer to as Greedy++). In Section 6.1 we study its accuracy in comparison with the optimum. In Section 6.2, we show the speedup of Greedy++ on the greedy algorithm proposed in [9] (which we call Greedy). Then, in Section 6.3, we compare Greedy++ with OSA, the heuristic based on sampling proposed in [9] (we did not implement the other heuristic BSA, since the authors of [9] show that OSA always finds a solution with a similar accuracy as BSA in a smaller running time). In Section 6.4, we study the running time of Greedy++ on additional larger networks, both for a sequential and a parallel implementation (the other algorithms are either too slow or would require too much memory for these networks). Finally, in Section 6.5, we study the correlation between the group with maximum closeness and the top-k nodes with highest closeness in real-world networks.

We implemented all algorithms in C++ building on NetworKit [26], an open-source tool for fast exploratory analysis of large networks. 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. The machine runs 64 bit SUSE Linux and we compiled our code with g++-4.8.1 and OpenMP 3.1. For comparability with previous work, unless stated explicitely running times refer to a sequential implementation.

The graphs used in the experiments are taken from the SNAP [19], KONECT [18] and LASAGNE[1] data sets. The `easyjet` graph in Table 1 was taken from [11]. All graphs are connected, undirected and unweighted.

6.1 Accuracy

We compare the quality of the solution found by Greedy++ with that of the optimum on several small real-world networks, summarized in Table 1. We compute the optimum using the ILP formulation described in Section 5. The ILP model was implemented using the Java optimization modeling library and interface ILOG Concert Technology. The problems for the given data sets were solved with ILOG CPLEX 12.6[2]. The results for $k = 10$ are reported in Table 1. Among all networks, the empirical approximation ratio (ratio between the objective function of the optimum and that of the solution found by Greedy++) is always higher than 0.97. This is much higher than the theoretical guarantee of $(1 - 1/e) \approx 0.63$. Similar results can be observed for $k = 2$ and $k = 20$, reported in Table 3 and Table 4 in the appendix. For $k = 10$, the geometric mean of the approximation ratios is 0.994, for $k = 2$ it is 0.998 and for $k = 20$ it is 0.995. Notice that Greedy++ never takes more than one second on the tested networks, whereas finding the optimum with CPLEX takes hours for the larger instances of Table 1.

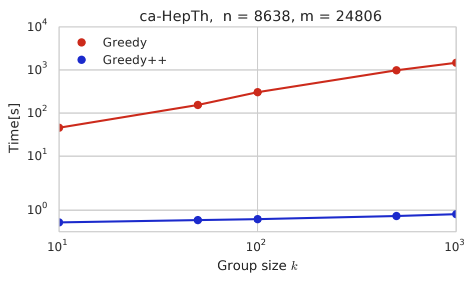
Table 1 Comparison with optimum on small real-world networks, for $k = 10$. The fourth and fifth columns show the objective function of Eq. (2) for the optimum and Greedy++, respectively.

Graph	Nodes	Edges	Category	Optimum	Greedy++	Approx. ratio
`karate`	35	78	friendship	25	25	1.0
`contiguous-usa`	49	107	transport.	40	41	0.976
`easyjet`	136	755	transport.	126	126	1.0
`jazz`	198	2742	collaboration	191	192	0.995
`coli1-1Inter`	328	456	metabolic	475	482	0.985
`pro-pro`	1458	1993	metabolic	4213	4217	0.999
`hamster-friend`	1788	12476	social	2871	2871	1.0
`dnc-temporal`	1833	4366	communicat.	2398	2407	0.996
`caenorhab-eleg`	4428	9659	metabolic	10003	10075	0.993

6.2 Speedup on Greedy

We recall that the solution found by the two algorithms Greedy++ and Greedy is the same, so we only compare running times between the two. Due to the time and space complexity

[1] `piluc.dsi.unifi.it/lasagne`
[2] `www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`



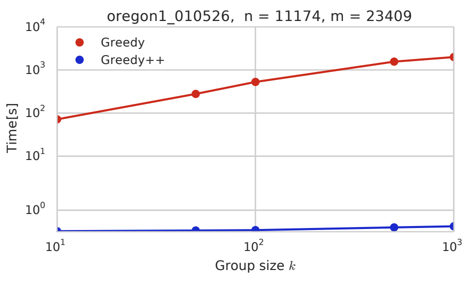


Figure 2 Running times of Greedy and Greedy++ for different group sizes (log-log scale). Top: running times for `ca-HepTh`; bottom: running times for `oregon_1_010526`.

of Greedy, we compare the two approaches on two relatively small networks (`ca-HepTh`: 8638 nodes and 24806 edges and `oregon_1_010526`: 11174 nodes and 23409 edges). Figure 2 shows the running times of the two algorithms for different values of group size k between 10 and 1000. For both graphs, Greedy++ outperforms Greedy by orders of magnitude. For all tested group sizes, Greedy++ finds the solution in less than one second, whereas for $k = 1000$ Greedy requires 25 minutes on the `ca-HepTh` graph and 34 minutes on the `oregon_1_010526` graph. The speedups of Greedy++ on Greedy ranges between 93 ($k = 10$) and 1765 ($k = 1000$) for `ca-HepTh` and between 581 ($k = 10$) and 6125 ($k = 1000$) for `oregon_1_010526`.

6.3 Comparison with OSA

Since OSA is a sampling-based algorithm, the number h of samples influences its performance, both in terms of accuracy and running time. In [9], the authors suggest $h = 1000$ samples as a good tradeoff for group sizes up to 50. Since we are also testing the algorithms on groups with up to 100 nodes, we run OSA both with $h = 1000$ and with a larger sample size of $h = 2000$. We test OSA and Greedy++ on all the networks of Table 2 with $m < 10^7$ (11 networks). We did not run experiments on larger networks because of the high memory requirements of OSA. Since OSA is a sampling-based approach, we repeat each experiment 10 times and report the average running time and accuracy.

Figure 3 shows the group closeness of the solutions found by OSA and Greedy++ on four of the tested graphs (`email-Enron`, `loc-brightkit`, `flickr`, and `gowalla`), for group sizes ranging between 5 and 100. As a baseline, we also report the closeness of the group composed of the k nodes with maximum degree (Degree). In addition to having a theoretical guarantee (whereas OSA has none), the results show that Greedy++ always finds the best solution, for all graphs and group sizes. Interestingly, for all the four graphs but `flickr`, the set of nodes with maximum degree has a higher closeness than the solution found by OSA with $h = 1000$ samples. For the `gowalla` graph, Degree finds a better solution than OSA even with $h = 2000$ samples.

Figure 4 shows the running times of Greedy++ and OSA on the four graphs, for group size $k = 20$ (top) and $k = 100$ (bottom). On all graphs but `flickr`, Greedy++ is significantly faster than OSA (both with $h = 1000$ and $h = 2000$ samples). On the `flickr` graph, for group size $k = 20$, Greedy++ takes 85 seconds, whereas OSA with $h = 2000$ takes 77 seconds. However, when the group size increases ($k = 100$), Greedy++ becomes faster (102 seconds versus 182 seconds required by OSA with $h = 2000$).

Also, notice that the memory requirement of Greedy++ is significantly lower than that of OSA. In fact, Greedy++ only needs $\Theta(n)$ memory for its data structures, whereas OSA requires $\Theta(hn)$ to store the distances between the sampled nodes and the other nodes. This means that, using OSA with the number of samples suggested in [9], it needs about one thousand times more memory than Greedy++, which might be problematic for large graphs.

On average (geometric mean) over the 11 tested networks, Greedy++ is faster than OSA with $h = 1000$ by a factor 1.1 and than OSA with $h = 2000$ by a factor 1.7. Although our average running times are not very different from those of OSA with $h = 1000$, our accuracy is better on all tested networks (the same is true also for OSA with $h = 2000$). Also, on 7 out of the 11 tested networks, OSA with $h = 1000$ returns a result with a worse accuracy than choosing the k nodes with maximum degree, suggesting that OSA should be run using a larger number of samples. With $h = 2000$, the solution of OSA is worse than picking the k nodes with maximum degree on 4 out of 11 networks (the solution returned by Greedy++ is better on all tested networks).

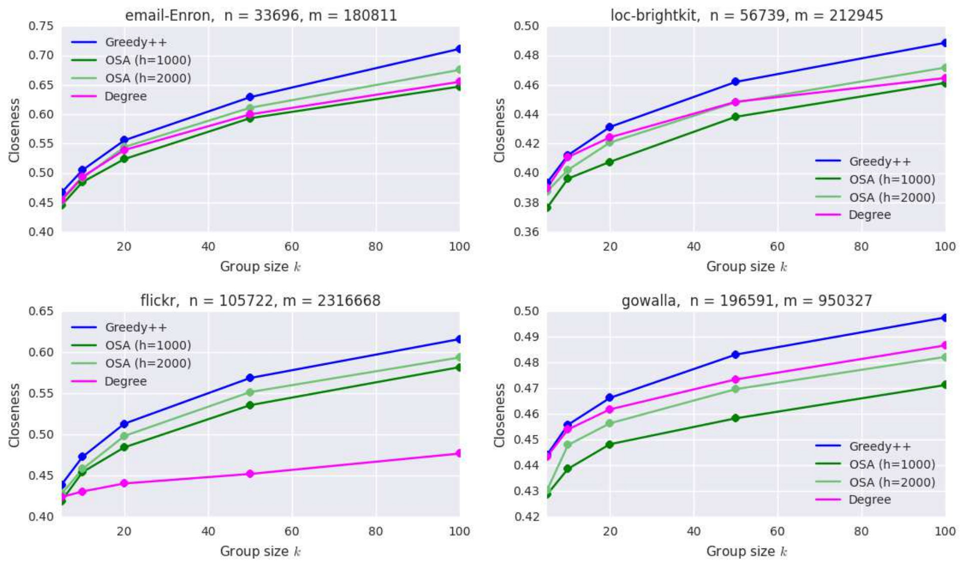


Figure 3 Closeness centrality of the solution found by the methods for different group sizes and different graphs. The plot shows the results of Greedy++, OSA with sample sizes of 1000 and 2000, and the group consisting of the k nodes with highest degree.

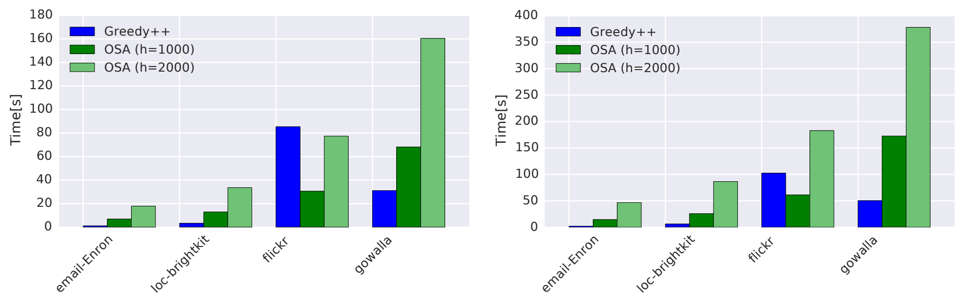


Figure 4 Running times of Greedy++ and OSA with sample sizes of 1000 and 2000 for $k = 20$ (top) and $k = 100$ (bottom).

6.4 Running time evaluation

To test the scalability of Greedy++, we now run it on all networks from Table 2 (for the comparison with OSA, only the first 11 networks could be used). The networks belong to different domains: In particular, `loc-brightkite`, `gowalla`, `com-lj`, `com-youtube`, `flickr`, `youtube-u-growth`, `soc-pokec-relationships`, `com-orkut` are friendship networks, `ca-HepPh`, `CA-AstroPh`, `com-dblp` are co-authorship and collaboration networks, `com-amazon` represents co-purchased articles, `email-Enron` is a communication network and `as-skitter` is an internet topology graph.

To further speed up the running time of Greedy++, we also implement a parallel version of it. The first element of $|S|$ is computed using the parallel top-k closeness implementation described in [4]. Then, in each iteration of Greedy++, Line 6 of Algorithm 1 is exectuted in parallel, i.e. each thread runs a pruned SSSP from the nodes assigned to it.

Table 2 reports the running times of Greedy++ for $k = 10$, for both the sequential and the parallel implementation (using 16 threads). On all networks with less than 10^5 nodes, our parallel implementation takes less than 1 second. On all remaining graphs, it always takes less than 1 hour, apart from the `com-orkut` graph ($>$ 3M nodes and $>$ 100M edges), where it takes a bit more than one and a half hours. The parallel speedup varies significantly among the tested networks, ranging from 5.4 (`com-youtube`) to 13.8 (`flickr`). These values should be appreciated in the context of complex networks, for which it is often difficult to obtain even higher speedups (see for example [21] and [25]). A low speedup is in our case probably due to the fact that, in some networks, the work done by the pruned SSSPs is extremely unbalanced (some nodes can be pruned early, whereas others need almost a full SSSP). Further load balancing mechanisms seem to require very fine-grained and inexpensive context switches between threads (and thus a different hardware architecture). Also, as expected, the parallel speedup decreases as k increases. Indeed, whereas the geometric mean of the speedups is 9.1 for $k = 10$, it is 8.7 for $k = 20$ and 5.6 for $k = 100$. This results from the fact that, for higher k values, more and more pruned SSSPs can be skipped because of submodularity. Since less work is done in each iteration, the overhead due to parallelism and imbalance becomes more significant. The fact that less and less work is done in each iteration as k increases is also confirmed by the fact that the running times do not increase linearly as k increases. For $k = 20$, the running times are only about 10% higher (on average) that they are for $k = 10$ and, for $k = 100$, they are about 50% higher than for $k = 10$ (running times for $k = 20$ and $k = 100$ can be found in Table 5 in the appendix).

6.5 Group closeness versus top-k closeness

A natural question is how many elements of the group of nodes with highest closeness have high closeness individually. We investigate this on the networks of Table 2. In particular, for a given group size k, we compute the overlap (i. e. the size of the intersection) between the group returned by Greedy++ and the set of the top-k nodes with highest closeness, computed using the algorithm described in [3] (already implemented in NetworKit). The percentage overlap is then the overlap divided by k and multiplied by 100. Figure 5 in the appendix shows the results. In addition to the percentage overlap between Greedy++ and the top k nodes with highest closeness, it also shows the one between Greedy++ and the k nodes with highest degree. The plot on the bottom right corner shows the average over all networks of Table 2, whereas the other three plots refer to the `com-youtube` graph, to `soc-pokec-relationships` and to `com-orkut`, respectively. As it appears from the plots, the overlap changes significantly among the graphs. For the `com-youtube` graph, the percentage overlap decreases as the group size increases, and the overlap with Degree is always larger

Table 2 Networks used in the experiments and performance of Greedy++ for $k = 10$. The forth and fifth columns report the sequential and parallel running times with 16 threads, respectively. The last column reports the speedup of the parallel implementation on the sequential one.

Graph	Nodes	Edges	Time seq. [s]	Time par. [s]	Speedup
`ca-HepPh`	11204	117649	7.70	0.58	13.4
`email-Enron`	33696	180811	1.94	0.20	9.9
`CA-AstroPh`	17903	197031	3.78	0.32	12.0
`loc-brightkite`	56739	212945	5.74	0.55	10.5
`com-lj`	303526	427701	127.35	17.00	7.5
`com-amazon`	334863	925872	808.70	88.37	9.2
`gowalla`	196591	950327	60.14	8.74	6.9
`com-dblp`	317080	1049866	232.51	30.99	7.5
`flickr`	105722	2316668	314.11	22.76	13.8
`com-youtube`	1134890	2987624	1323.31	245.50	5.4
`youtube-u-growth`	3216075	9369874	22298.52	2196.42	10.2
`as-skitter`	1694616	11094209	12014.09	1611.09	7.5
`soc-pokec-relationships`	1632803	22301964	11912.29	1104.82	10.8
`com-orkut`	3072441	117185083	60252.10	5792.81	10.4

than the one with Top-k. Partially similar are the results for `soc-pokec-relationships`, although there is more fluctuation in the overlap of Degree and the initial overlap of Top-k is higher than it is for `com-youtube` ($\approx 80\%$ vs. $\approx 60\%$). On the other hand, the results for `com-orkut` are quite different: The overlap with Degree increases with the group size, and is lower than the one with Top-k.

On average, the overlap with both Degree and Top-k tends to decrease as the group size increases (as expected), with Degree having a higher overlap than Top-k (except for $k = 5$). Also, on average the overlap ranges between 30% and 60%. This clearly indicates that there is a dependence between the group with maximum closeness and the degrees of nodes and their centralities. However, the strength of this dependence varies significantly among the tested networks and suggests that picking the k nodes with highest closeness or highest degree is not always a good heuristic for finding the group with maximum closeness.

7 Conclusions

In this paper we have studied the problem of finding the group with maximum closeness in large complex networks. Our algorithm is the first that scales to networks with tens or hundreds of millions of edges *and* delivers a guaranteed approximation ratio of $(1 - 1/e)$ at the same time. Pruning the SSSP searches and exploiting the submodularity of the objective function allows us to reduce the amount of work done by the greedy algorithm proposed in [9] by orders of magnitude. In a comparison with the optimum on several small real-world networks, the empirical approximation ratio is never lower than 0.97.

Also, using our approach, we have been able to study the relation between a group with high closeness and nodes that have individually high closeness or degree in large complex networks. Future work includes an extension of our approach to disk-resident graphs, for which a comparison with the heuristic proposed in [28] would be interesting.

It would also be interesting to investigate whether our work could be extended to other centrality measures, such as current-flow closeness [5]. Finally, we plan to study how an extension of our greedy algorithm would perform on the p-median problem with node weights.

References

1 *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016, Arlington, Virginia, USA, January 10, 2016*. SIAM, 2016.

2 P. Avella, M. Boccia, S. Salerno, and I. Vasilyev. An aggregation heuristic for large scale p-median problem. *Computers & OR*, 39(7):1625–1632, 2012.

3 E. Bergamini, M. Borassi, P. Crescenzi, A. Marino, and H. Meyerhenke. Computing top-k closeness centrality faster in unweighted graphs. In *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016, Arlington, Virginia, USA, January 10, 2016* [1], pages 68–80.

4 E. Bergamini, M. Borassi, P. Crescenzi, A. Marino, and H. Meyerhenke. Computing top-k closeness centrality faster in unweighted graphs. *CoRR*, abs/1704.01077, 2017.

5 E. Bergamini, M. Wegner, D. Lukarski, and H. Meyerhenke. Estimating current-flow closeness centrality with a multigrid laplacian solver. In *Proceedings of the Seventh SIAM Workshop on Combinatorial Scientific Computing, CSC 2016, Albuquerque, NM, USA, October 10-12, 2016*, 2016.

6 M. Borassi, P. Crescenzi, and A. Marino. Fast and simple computation of top-k closeness centralities. *CoRR*, abs/1507.01490, 2015.

7 U. Brandes and C. Pich. Centrality estimation in large networks. *I. J. Bifurcation and Chaos*, 17(7):2303–2318, 2007.

8 S. Chechik, E. Cohen, and H. Kaplan. Average distance queries through weighted samples in graphs and metric spaces: High scalability with tight statistical guarantees. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015*, volume 40 of *LIPIcs*, pages 659–679. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

9 C. Chen, W. Wang, and X. Wang. Efficient maximum closeness centrality group identification. In *Databases Theory and Applications - 27th Australasian Database Conference, ADC 2016, Sydney, NSW, September 28-29, 2016, Proceedings*, volume 9877 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2016.

10 E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Computing classic closeness centrality, at scale. In *Proceedings of the second ACM conference on Online social networks, COSN 2014, Dublin, Ireland, October 1-2, 2014*, pages 37–50. ACM, 2014.

11 P. Crescenzi, G. D'Angelo, L. Severini, and Y. Velaj. Greedily improving our own centrality in A network. In *Experimental Algorithms - 14th International Symposium, SEA 2015, Paris, France, June 29 - July 1, 2015, Proceedings*, volume 9125 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2015.

12 Z. Drezner. *Facility Location*. Springer, 1995.

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

14 M. G. Everett and S. P. Borgatti. The centrality of groups and classes. *The Journal of Mathematical Sociology*, 23(3):181–201, 1999.

15 S. L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13(3):462–475, 1965.

16 C. A. Irawan and S. Salhi. Solving large p-median problems by a multistage hybrid approach using demand points aggregation and variable neighbourhood search. *J. Global Optimization*, 63(3):537–554, 2015.

17 D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 137–146. ACM, 2003.

18 J. Kunegis. KONECT: the koblenz network collection. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume*, pages 1343–1350. International World Wide Web Conferences Steering Committee / ACM, 2013.

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

20 A. Mahmoody, C. E. Tsourakakis, and E. Upfal. Scalable betweenness centrality maximization via sampling. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1765–1773. ACM, 2016.

21 R. McColl, O. Green, and D. A. Bader. A new parallel algorithm for connected components in dynamic graphs. In *20th Annual International Conference on High Performance Computing, HiPC 2013, Bengaluru (Bangalore), Karnataka, India, December 18-21, 2013*, pages 246–255. IEEE Computer Society, 2013.

22 P. W. Olsen, A. G. Labouseur, and J. Hwang. Efficient top-k closeness centrality search. In I. F. Cruz, E. Ferrari, Y. Tao, E. Bertino, and G. Trajcevski, editors, *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 196–207. IEEE Computer Society, 2014.

23 P. Rebreyend, L. Lemarchand, and R. Euler. A computational comparison of different algorithms for very large p -median problems. In *Evolutionary Computation in Combinatorial Optimization - 15th European Conference, EvoCOP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings*, volume 9026 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2015.

24 J. Reese. Solution methods for the p-median problem: An annotated bibliography. *Networks*, 48(3):125–142, 2006.

25 C. L. Staudt and H. Meyerhenke. Engineering parallel algorithms for community detection in massive networks. *IEEE Trans. Parallel Distrib. Syst.*, 27(1):171–184, 2016.

26 C. L. Staudt, A. Sazonovs, and H. Meyerhenke. NetworKit: A tool suite for large-scale complex network analysis. *Network Science*, 4:508–530, 2016.

27 Y. Yoshida. Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 1416–1425. ACM, 2014.

28 J. Zhao, J. C. S. Lui, D. Towsley, and X. Guan. Measuring and maximizing group closeness centrality over disk-resident graphs. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 689–694. ACM, 2014.

A Additional pseudocodes

Algorithm 2: Top-k closeness centrality [4].

```
Input  : A graph $G = (V, E)$, a number $k$
Output : Top $k$ nodes with highest closeness
$c(v) \leftarrow 0 \ \ \forall v \in V$;
$x_k \leftarrow 0$;
for $v \in V$ in decreasing order of degree do
    $c(v) \leftarrow$ BFScut$(v, x_k)$;
    if $c(v) \neq 0$ then
        $x_k \leftarrow$ Kth$(c)$;
    end
end
return TopK$(c)$;
```

Algorithm 3: Greedy algorithm for GCM [9].

```
Input  : A graph $G = (V, E)$, a number $k$
Output : A set $S$ of nodes of size $k$ such that $c(S) \geq (1 - 1/e)c(S^\star)$
$d \leftarrow$ APSP$(G)$;
$M \leftarrow$ APSP$(G)$;
$Score \leftarrow \{c(u) | u \in V\}$;
$s \leftarrow \arg\max_{u \in V \setminus S} Score[u]$;
$S \leftarrow \{s\}$;
while $|S| < k$ do
    foreach $u \in V \setminus S$ do
        foreach $w \in V$ do
            if $d[u, w] > d[s, w]$ then
                $M[u, w] \leftarrow d[s, w]$;
            end
        end
        /* $Score[u]$ is set to $c(S \cup \{u\})$ */
        $Score[u] \leftarrow (n - |S| - 1)/\sum_{w \in V \setminus S} M[u, w]$;
        $s \leftarrow \arg\max_{w \in V \setminus S} Score[w]$;
        $S \leftarrow S \cup \{s\}$;
    end
end
return $S$;
```

B Additional experimental results

Table 3 Comparison with optimum on small real-world networks, for $k = 2$. The fourth and fifth columns show the objective function of Eq. (2) for the optimum and Greedy++, respectively.

Graph	Nodes	Edges	Category	Optimum	Greedy++	Approx. ratio
`karate`	35	78	friendship	37	37	1.0
`contiguous-usa`	49	107	transport.	99	99	1.0
`easyjet`	136	755	transport.	143	143	1.0
`jazz`	198	2742	collaboration	259	261	0.992
`coli1-1Inter`	328	456	metabolic	780	780	1.0
`pro-pro`	1458	1993	metabolic	5573	5573	1.0
`hamster-friend`	1788	12476	social	3596	3596	1.0
`dnc-temporal`	1833	4366	communicat.	3236	3236	1.0
`caenorhab-eleg`	4428	9659	metabolic	12535	12631	0.992

Table 4 Comparison with optimum on small real-world networks, for $k = 20$. The fourth and fifth columns show the objective function of Eq. (2) for the optimum and Greedy++, respectively. The results for `caenorhab-eleg` are not included, because the CPLEX solver did not find the optimum within 13 hours.

Graph	Nodes	Edges	Category	Optimum	Greedy++	Approx. ratio
`karate`	35	78	friendship	15	15	1.0
`contiguous-usa`	49	107	transport.	29	29	1.0
`easyjet`	136	755	transport.	116	116	1.0
`jazz`	198	2742	collaboration	178	178	1.0
`coli1-1Inter`	328	456	metabolic	367	373	0.984
`pro-pro`	1458	1993	metabolic	3488	3518	0.991
`hamster-friend`	1788	12476	social	2556	2573	0.993
`dnc-temporal`	1833	4366	communicat.	2066	2082	0.992

Table 5 Performance of Greedy++ for $k = 20$ and $k = 100$, using 16 threads.

Graph	Nodes	Edges	Time $k = 20$ [s]	Time $k = 100$ [s]
`ca-HepPh`	11204	117649	0.61	0.7
`email-Enron`	33696	180811	0.26	0.6
`CA-AstroPh`	17903	197031	0.34	0.5
`loc-brightkite`	56739	212945	0.67	1.2
`com-lj`	303526	427701	18.16	24.2
`com-amazon`	334863	925872	94.56	116.2
`gowalla`	196591	950327	9.09	11.2
`com-dblp`	317080	1049866	34.26	49.5
`flickr`	105722	2316668	23.04	24.6
`com-youtube`	1134890	2987624	263.17	473.9
`youtube-u-growth`	3216075	9369874	2412.60	2901.9
`as-skitter`	1694616	11094209	1620.43	2024.6
`soc-pokec-relationships`	1632803	22301964	1179.33	1288.1
`com-orkut`	3072441	117185083	6233.67	8387.0

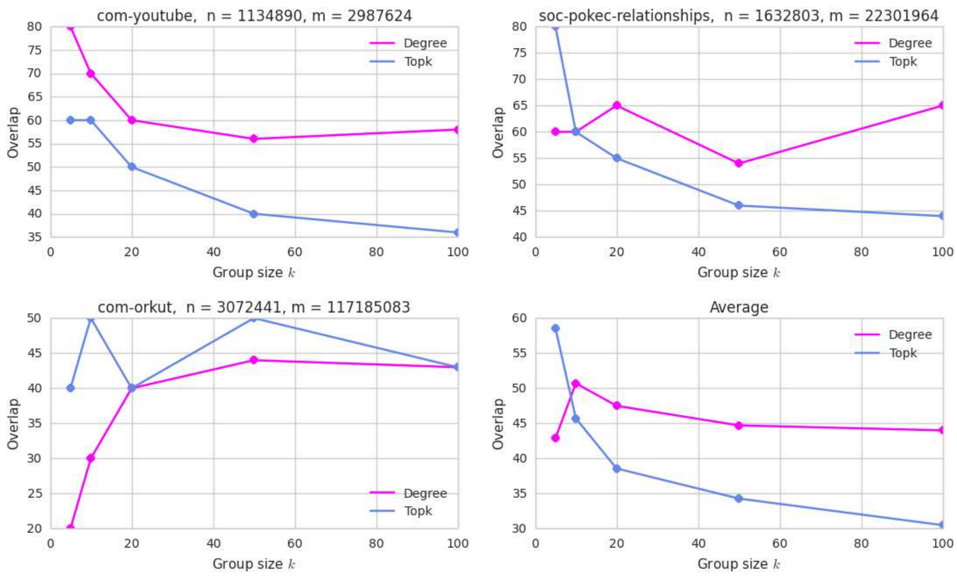


Figure 5 Percentage overlap between the group found by Greedy++ and the k nodes with highest closeness (Top-k) and between the group found by Greedy++ and the k nodes with highest degree (Degree).