Advances in Exact and Approximate Group Closeness Centrality Maximization

Christian Schulz
Faculty of Mathematics and Computer Science, Heidelberg University, Germany

Jakob Ternes
Faculty of Mathematics and Computer Science, Heidelberg University, Germany

Henning Woydt
Faculty of Mathematics and Computer Science, Heidelberg University, Germany

Abstract

In the NP-hard Group Closeness Centrality Maximization problem, the input is a graph $G = (V, E)$ and a positive integer k, and the task is to find a set $S \subseteq V$ of size k that minimizes group farness $f(S) = \sum_{v \in V} \min_{s \in S} \text{dist}(v, s)$. The state-of-the-art exact algorithm iteratively solves ILPs of increasing size until the final ILP can provably represent an optimal solution. We introduce a new data reduction technique that eliminates variables from the ILP by proving that certain vertices have their distance to any optimal solution structurally determined by a neighbor. Additionally, we bootstrap the exact solver with an approximate solution to produce near-sufficient ILPs from the first iteration, reducing the number of needed iterations. Our improvements yield a speedup by a factor of 4.5 over the next best exact algorithm and can achieve speedups by up to a factor of 34.1. Furthermore, we add reduction techniques to a 1/5-approximation algorithm, and show that these adaptations do not compromise its approximation guarantee. The improved algorithm achieves mean speedups of up to 1.6 and a maximum speedup of 9.6 times. Finally, we settle an open question by proving that a widely used greedy algorithm admits arbitrarily poor approximation ratios.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Group Closeness Centrality, Exact Algorithms, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.3

Related Version *Full Version*: `https://arxiv.org/abs/2603.25642`

Supplementary Material *Software (Source Code)*: `https://github.com/AEGH-Grover/Grover` archived at `swh:1:dir:9074d81e8040987b07cf1b78338b7377c5cf0f30`

Funding *Henning Woydt*: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – DFG SCHU 2567/6-1.

1 Introduction

A crucial task in the analysis of networks is the identification of vertices or groups of vertices which, in some sense, are important. Centrality measures quantify the importance by assigning a numeric value to a vertex or a group of vertices. They are frequently used in the analysis of social networks [9], biological networks [24, 10], electrical power grids [1] and, among many other uses, in facility location problems [15]. Various measures have been proposed to identify the centrality of an individual vertex in a network. For instance, closeness centrality measures how far a node is from the other nodes in the graph, betweenness centrality measures how often a node lies on the shortest path between other nodes, and degree centrality simply captures the number of neighbors. As noted by Everett and Borgatti [12], these measures fall short of identifying vertices which are collectively central as a group, leading them to propose generalizations of the aforementioned measures to groups of vertices. Indeed, it has been shown that the overlap between the top k nodes with the greatest individual closeness

centrality and groups of size k with high group closeness centrality is relatively small [8]. Naturally, the question arises of how to efficiently find groups of a given size k such that a certain group centrality measure is maximized. In the NP-hard [11, 22] group closeness centrality maximization (GCCM) problem, a positive integer k and an undirected connected graph $G = (V, E)$ are given, and the task is to find a group $S \subseteq V$ of size k that maximizes

$$c(S) \coloneqq \frac{|V| - k}{\sum_{v \in V} \min_{s \in S} \operatorname{dist}(v, s)}.$$

There are two prominent ways to exactly solve the problem. On the one hand, branch-and-bound algorithms specifically designed for this problem are used. These algorithms exhaustively explore a search-tree containing all possible solutions. To speed up the computation, many pruning techniques are used to cut off large parts of the explorable search-space. Such methods have been explored by Rajbhandari et al. [18], Staus et al. [22] and Woydt et al. [25]. On the other hand, ILP solvers can be used to solve GCCM. Since ILPs can capture a vast range of problems, a huge amount of research and engineering has flowed into ILP solvers. This has made ILP solvers an incredibly powerful tool for a large class of optimization problems, including GCCM. Bergamini et al. [8] were the first to propose an ILP for this problem with $\mathcal{O}(n^2)$ variables and constraints. Staus et al. [22] then improved on the proposed formulation and reduced the variable and constraint count to $\mathcal{O}(n \cdot \operatorname{diam}(G))$. Additionally, they proposed an iterative approach that starts by solving smaller ILPs and only if necessary enlarges them. Experimental evaluations in [22, 25] show that ILP solvers currently outperform the branch-and-bound approach.

When solution quality is not paramount, one may decide to use a heuristic or approximation algorithm. Chen et al. [11] considered a simple greedy algorithm that starts with the empty set and iteratively adds the vertex that improves the score the most. They claimed that the algorithm has a $(1 - 1/e)$-approximation guarantee. The runtime of the algorithm was massively improved by Bergamini et al. [8] and they observed very high empirical solution quality when compared to an optimal solution. However, they also showed that the $(1 - 1/e)$-approximation guarantee was erroneously assumed. Angriman et al. [2] later presented an algorithm with a 1/5-approximation ratio that is based on a local search algorithm for the related k-median problem [5]. The algorithm may start with any initial solution, which is then refined using a swap-based local search procedure. Vertices inside the solution are swapped with vertices outside of the solution if the swap improves the objective. If no more objective-improving swaps can be made then the algorithm terminates.

Our Contribution. We develop new algorithmic techniques for group closeness centrality maximization, both in the exact and approximate case. In the exact case, we introduce *absorbed vertices*, a data reduction that goes beyond the known dominated vertex removal proposed by Staus et al. [22]. Additionally, we bootstrap the iterative ILP framework with an approximate solution, producing near-optimal ILPs from the first iteration. For some instances, these improvements can lead to a speedup of 34.16×, while the overall mean speedup is 4.59×. In the approximate case, we prove that the search space of the 1/5-approximation algorithm of Angriman et al. [2] can be restricted to non-dominated vertices without compromising its approximation guarantee, achieving mean speedups of 1.61×. Finally, we show that the greedy algorithm proposed by Chen et al. [11] does not have an approximation guarantee for GCCM.

2 Preliminaries

Let $G = (V, E)$ be an undirected, unweighted, connected graph without self-loops. Two vertices $u, v \in V$ are *adjacent* if $\{u, v\} \in E$. The *neighborhood* of a vertex v is $N(v) = \{u \in V \mid \{u, v\} \in E\}$, and its *degree* is $\deg(v) = |N(v)|$. The *maximum degree* is $\Delta(G) = \max_{v \in V} \deg(v)$. The *closed neighborhood* is $N[v] = N(v) \cup \{v\}$, and v *dominates* u if $N[u] \subseteq N[v]$. The *distance* $\operatorname{dist}(u, v)$ is the length of a shortest u-v path, and for a set $S \subseteq V$, $\operatorname{dist}(u, S) = \min_{v \in S} \operatorname{dist}(u, v)$. The *eccentricity* of v is $\operatorname{ecc}(v) = \max_{u \in V} \operatorname{dist}(u, v)$, and the *diameter* of G is $\operatorname{diam}(G) = \max_{v \in V} \operatorname{ecc}(v)$. A vertex $w \in V$ is a *cut vertex* if removing it increases the number of connected components.

The *closeness centrality* measure [6, 7, 21] (see Eq. 1) scores each vertex $v \in V$ while the *group closeness centrality* measure [12] (see Eq. 2) generalizes this score to vertex sets $S \subseteq V$.

$$(1) \quad c(v) := \frac{|V| - 1}{\sum_{u \in V} \operatorname{dist}(u, v)} \qquad (2) \quad c(S) := \frac{|V| - |S|}{\sum_{u \in V} \operatorname{dist}(u, S)}$$

Note that other definitions instead use $n := |V|$ or 1 as the numerator. Regardless of the exact definition of group closeness centrality, due to the identical denominator one may equivalently minimize *group farness* $f(S) := \sum_{u \in V} \operatorname{dist}(u, S)$.

Note that in the literature, there are varying names used to refer to this problem. Some authors refer to the problem itself as *group closeness centrality* [22], while others refer to it as *group closeness maximization* [8]. We refer to the problem as *group closeness centrality maximization* (GCCM), to differentiate the problem from the function.

Let $\mathcal{U}$ be a universe of items and $f : 2^{\mathcal{U}} \to \mathbb{R}$ be a set function. The *marginal gain* of $e \in \mathcal{U}$ at $A \subseteq \mathcal{U}$ is $\Delta(e \mid A) = f(A \cup \{e\}) - f(A)$. A set function is *submodular* if $\Delta(e \mid A) \geq \Delta(e \mid B)$ and *supermodular* if $\Delta(e \mid A) \leq \Delta(e \mid B)$ for all $A \subseteq B \subseteq \mathcal{U}$, $e \in \mathcal{U} \setminus B$. The group farness function is supermodular [8].

3 Related Work

3.1 Exact Algorithms

To exactly solve GCCM, recent research has focused on branch-and-bound algorithms [22, 25] or ILP solvers [8, 22]. Branch-and-bound algorithms explore the space of possible solutions $S \subseteq V$ with $|S| = k$ by traversing a Set-Enumeration Tree [20]. The search tree has $\mathcal{O}\binom{n}{k}$ nodes, however Staus et al. [22] and Woydt et al. [25] present methods to cut off large parts of the search tree. They exploit the supermodularity of group farness by computing a lower bound on the achievable score in a subtree. If that bound is greater than the score of the best found solution then the subtree does not hold a better solution and can be cut off.

Bergamini et al. [8] were the first to propose an ILP formulation for GCCM. For every vertex $v \in V$, a variable $y_v \in \{0, 1\}$ indicates whether $v \in S^*$ ($y_v = 1$) or $v \notin S^*$ ($y_v = 0$). Furthermore, they define the variables $x_{u,v} \in \{0, 1\}$ for each vertex pair $(u, v) \in V \times V$. They say a vertex u is *assigned* to a vertex $v \in S^*$ if $\operatorname{dist}(u, v) = \operatorname{dist}(u, S^*)$. If $v \in S^*$ and u is assigned to v then $x_{u,v} = 1$, otherwise $x_{u,v} = 0$. The ILP is then written as

$$\text{minimize} \quad \sum_{\substack{u \in V \\ v \in V}} \operatorname{dist}(u, v)\, x_{u,v} \quad \text{subject to} \quad \sum_{v \in V} y_v = k \tag{3a}$$

$$\sum_{v \in V} x_{u,v} = 1 \quad \forall\, u \in V \tag{3b}$$

$$x_{u,v} \leq y_v \quad \forall\, u, v \in V \tag{3c}$$

Constraint 3a ensures that exactly k vertices are in the solution. Constraint 3b ensures that each vertex u is assigned to exactly one vertex, while Constraint 3c ensures that u can only be assigned to a vertex $v \in S^*$. The objective function only adds the distance $\text{dist}(u, v)$ if u is assigned to v. Since each vertex is assigned to exactly one vertex in the solution, minimizing the term minimizes group farness. The ILP has $\mathcal{O}(n^2)$ variables and constraints.

Staus et al. [22] proposed further ILP formulations that reduce the number of needed variables and constraints. The idea for all of their new formulations is to circumvent the creation of the variables $x_{u,v}$. After all, the information to which vertex v the vertex u is assigned is not inherently relevant, only the underlying distance to S^* is. Therefore, the variables $x_{u,v}$ are removed from the ILP and instead variables $x_{u,i} \in \{0, 1\}$ are added with $u \in V$ and $i \in \{0, \dots, \text{diam}(G)\}$. If vertex u has distance i to the optimal solution S^*, then $x_{u,i} = 1$, otherwise $x_{u,i} = 0$. The new ILP formulation then reads as follows:

$$\text{minimize} \sum_{\substack{u \in V \\ i \in \{0,\dots,\text{diam}(G)\}}} i \cdot x_{u,i} \quad \text{subject to} \quad \sum_{v \in V} x_{v,0} = k \tag{4a}$$

$$\sum_{i \in \{0,\dots,\text{diam}(G)\}} x_{v,i} = 1 \quad \forall\, v \in V \tag{4b}$$

$$x_{v,i} \leq \sum_{\substack{w \in V \\ \text{dist}(v,w)=i}} x_{w,0} \quad \forall_{\substack{v \in V \\ i \in \{0,\dots,\text{diam}(G)\}}} \tag{4c}$$

Constraints 4a and 4b encode that the solution has exactly k vertices and each vertex has only one active distance to S^*. Constraint 4c ensures that the active distance from v to S^* is chosen correctly, i.e., $x_{v,i}$ can only be 1 if there exists at least one $w \in S^*$ with distance $i = \text{dist}(v, w)$. After optimization $S^* = \{v \mid x_{v,0} = 1\}$ is a globally optimal solution. This ILP has $\mathcal{O}(n \cdot \text{diam}(G))$ variables and constraints, making it inherently well suited for small diameter networks like social networks. The ILP can further be reduced in size by removing unnecessary variables. Instead of using $i \in \{0, \dots, \text{diam}(G)\}$, one can instead use $i \in \{0, \dots, \text{ecc}(v)\}$ for each individual vertex v since $\text{dist}(v, S^*) \leq \text{ecc}(v)$.

However, Staus et al. [22] still noticed that many $x_{v,i}$ are not relevant for the ILP. For example, if a solution with $x_{v,i} = 1$ was determined, then all variables $x_{v,j}$ with $j > i$ do not impact the objective function. An ILP without those variables will determine the same solution in less time. This led them to design an iterative algorithm called ILPIND [22], which is the state-of-the-art algorithm to exactly solve GCCM. The idea of the algorithm is to start with a small formulation of the ILP which might not be able to find the optimal solution - in which case the ILP is iteratively enlarged until this is the case. Instead of creating all variables $x_{v,i}$ with $i \in \{0, \dots, \text{ecc}(v)\}$, they only create the variables with $i \in \{0, \dots, d(v) = 2\}$. The variables $x_{v,i}$ with $i < d(v)$ keep their original meaning, however the variables $x_{v,d(v)}$ change their meaning. If $x_{v,d(v)} = 1$ then $\text{dist}(v, S^*) \geq d(v)$, i.e., the distance from v to S^* may be greater than $d(v)$, and therefore it is not guaranteed that S^* is optimal. If the ILP is solved and there is a $x_{v,d(v)} = 1$ with $d(v) < \text{ecc}(v)$, then the ILP is called *insufficient* and another iteration is needed. For each vertex v with $x_{v,d(v)} = 1$ the value $d(v)$ is increased by 1 but capped at $\text{ecc}(v)$. The ILP is solved with the new $d(v)$-values and this is repeated until eventually a sufficient ILP is encountered. While not obviously advantageous over solving the original ILP once, experiments in [22] show it is significantly faster.

Another improvement they utilize is the use of dominating vertices. Remember that a vertex v is dominated by a vertex u if $N[v] \subseteq N[u]$. Staus et al. [22] show that a set of dominated vertices D can be removed from the search space, i.e., there exists an optimal solution S^* in $V \setminus D$. More precisely, for all vertices $v \in V$ it must hold that $v \in V \setminus D$ or v

is dominated by a vertex in $V \setminus D$. Let $D \subseteq V$ be the set of all vertices removed from the search space according to this rule. Then, for all vertices $v \in D$, the ILP does not need the variables $x_{v,0}$. Note that $V \setminus D$ is constructed to have at least size k.

3.2 Heuristic and Approximate Algorithms

Due to the NP-hardness of GCCM, many authors have proposed heuristic algorithms. Chen et al. [11] proposed a simple greedy heuristic, which we call Greedy. Essentially, Greedy starts with the empty set $S_0 = \emptyset$ and iteratively increases $S_{i+1} = S_i \cup \{v\}$ with the vertex v that yields the greatest marginal gain. Then, the computed solution is S_k. Greedy first computes the $\mathcal{O}(n^2)$ size distance matrix in $\mathcal{O}(n(n+m))$ time by BFS from each vertex. To determine v takes $\mathcal{O}(n^2)$ time in each of the k iterations. Chen et al. [11] claimed that Greedy has an approximation ratio of $(1-1/e)$ by using a well-known result for submodular set functions [17]. Bergamini et al. [8] however pointed out that group closeness centrality is not submodular and thus showed that this approximation claim does not hold. Consequently, the question of whether GCCM could be approximated was considered unsettled [2]. We later show that Greedy does not have an approximation guarantee. However, Bergamini et al. [8] observed very high solution quality and proposed Greedy++, an improved algorithm that computes the same solution as Greedy while using less time and space. They only maintain a vector storing the distance from each vertex to the current set S_i and use n BFS queries per iteration, many of which can be terminated early or skipped entirely.

Angriman et al. [4] proposed two local search algorithms, LocalSwap and GrowShrink, with no known approximation guarantee. Both start by uniformly picking an initial solution S. LocalSwap operates by swapping vertices $v \in S$ with their neighbors $u \in N(v)$ if it is expected that the group farness decreases. If no swap is found, then the algorithm terminates and returns the current set. Their second algorithm, GrowShrink, operates in two distinct phases, the *grow*- and *shrink*-phase. During the grow-phase the algorithm adds additional vertices and temporarily violates the constraint $|S| = k$. In the shrink-phase the algorithm discards vertices such that the constraint is again fulfilled. The idea is that the grow-phase adds beneficial vertices, while the shrink-phase then discards the least beneficial ones.

In a later work, Angriman et al. [2] propose an algorithm with a 1/5-approximation guarantee, thus settling the approximability question of GCCM. The local search algorithm is based on an algorithm for the related k-median problem analyzed in Arya et al. [5]. They start with any solution S and then perform swaps $(S \setminus \{u\}) \cup \{v\}$ with $u \in S$ and $v \in V \setminus S$ if the swap is beneficial. Once all possible swaps do not improve the objective anymore, the algorithm terminates. They show that the result of Arya et al. [5] applies, i.e. for the output set S it holds that $c(S) \geq \frac{1}{5}c(S^*)$ (i.e. $f(S) \leq 5f(S^*)$) with S^* being an optimal solution. Since the approximation guarantee does not depend on the initial solution, different algorithms to obtain the initial solutions can be chosen. Angriman et al. propose two algorithms Greedy-Ls-C and GS-LS-C, where the first one uses the solution provided by the Greedy algorithm and the second one utilizes the GrowShrink algorithm.

Finally, Rajbhandari et al. [18] proposed an anytime algorithm called Presto that finds heuristic solutions and, when run until termination, exact solutions.

4 Grover

The main contribution of this work is our exact algorithm Grover (**Gro**up Closeness Centrality Maximization Sol**ver**). It addresses two structural limitations of ILPind [22]: the ILP contains many variables whose values are fully determined by other variables, and the

iterative framework wastes many iterations before the ILP becomes sufficient. We resolve the first limitation through absorbed vertices, a new reduction that eliminates variables by folding their cost into their neighbor. The second is resolved by bootstrapping the ILP with an approximate solution, producing near-sufficient formulations from the first iteration.

Recap. Recall that the algorithm ILPIND from Staus et al. [22] starts by solving the ILP with variables $x_{v,i}$ for $v \in V$ and $i \in \{0, \dots, d(v) = 2\}$. This way, solutions with $\text{dist}(v, S^*) \leq 1$ or $\text{dist}(v, S^*) = 2$ and $\text{ecc}(v) = 2$ can be represented by the ILP. After the model is optimized with the current $d(v)$-values, we may find that $x_{v,d(v)} = 1$ for some vertices v, indicating that $\text{dist}(v, S^*) \geq d(v)$. If, in addition, $\text{ecc}(v) > d(v)$, then we cannot rule out the possibility that $\text{dist}(v, S^*) > d(v)$. For such vertices v, the value $d(v)$ is incremented by 1 to make sure that a possible solution S^* with $\text{dist}(v, S^*) > d(v)$ can be represented by the ILP model. This process repeats until eventually a sufficient ILP is obtained.

4.1 Reducing Iterations

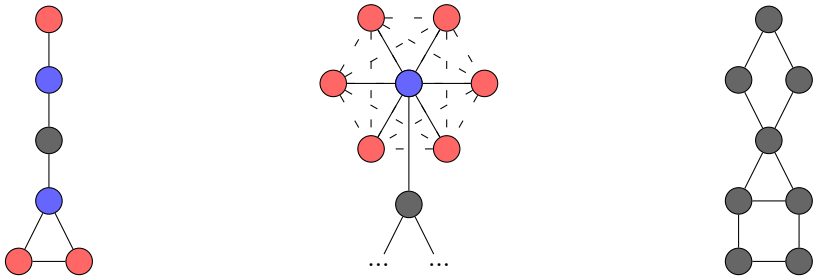
The iterative framework of ILPIND faces a fundamental problem. Setting $d(v) = 2$ produces a small ILP, but on graphs with a moderate to large diameter, the first ILPs cannot represent any optimal solution and many iterations are wasted incrementally enlarging the model. Conversely, setting $d(v) = \text{ecc}(v)$ immediately yields a sufficient ILP, but it has many unnecessary variables that slow down the solver. Neither extreme is satisfactory: the former pays in iteration count, the latter in per-iteration cost.

We resolve this problem by using an estimate $2 \leq \widetilde{d}(v) \leq \text{ecc}(v)$ for each vertex v. The closer these estimates are to the final $d(v)$-values needed to represent an optimal solution, the fewer iterations are needed. If indeed $\widetilde{d}(v) = \text{dist}(v, S^*) + 1$ and a unique optimal solution S^* exists, the ILP is sufficient in the first iteration with as few variables as possible. Note that if multiple optimal solutions exist, then additional iterations might be necessary to verify that no other optimal solution is strictly better. The key question is how to obtain good estimates $\widetilde{d}(v)$ without knowing S^*. Heuristic algorithms for GCCM produce solutions empirically very close to optimal on real-world graphs [2, 3, 8], so $\text{dist}(v, \widetilde{S})$ closely tracks the unknown $\text{dist}(v, S^*)$ and can be used to determine $\widetilde{d}(v)$. Concretely, we derive $\widetilde{S}$ using GS-LS-C from Angriman et al. [3] due to its 5-approximation and set $\widetilde{d}(v) = \max\{\text{dist}(v, \widetilde{S}) + 1, 2\}$ for the first iteration. While we use GS-LS-C for its approximation guarantee, any algorithm producing an initial solution $\widetilde{S}$ can be used, since the iterative enlargement of the ILP ensures that optimality is guaranteed regardless of the quality of $\widetilde{S}$.

4.2 Absorbed Vertices

Staus et al. [22] showed that a set $D \subseteq V$ of dominated vertices can be removed from the search space without compromising the optimality guarantee. This is expressed by removing the variables $x_{v,0}$ for each $v \in D$ from the ILP. However, this reduction only affects the search space, as distance variables $x_{v,1}, \dots, x_{v,d(v)}$ and their constraints remain, since they are needed to compute the objective function, and the size of the ILP is largely unchanged.

Absorbed vertices go fundamentally further by eliminating *all* variables of a vertex from the ILP. The key insight is that certain vertices have their distance to any optimal solution fully determined by a neighbor's distance. This allows us to fold their associated costs into the neighbor and completely remove them from the ILP. For example, consider a vertex v with $\deg(v) = 1$ and let u be its neighbor. Since all shortest paths from v to S^* must pass through the edge $\{u, v\}$, we know $\text{dist}(v, S^*) = \text{dist}(u, S^*) + 1$. We therefore can deduce $x_{u,i} = x_{v,i+1}$ in any solved ILP. Since we are able to deduce the assignment of $x_{v,i+1}$ from $x_{u,i}$ we can remove it from the ILP. However, removing the variable alters the objective



(a) (b) (c)

Figure 1 Figure showing absorbed vertices. In case (a) and (b) the red vertices can be absorbed by their neighboring blue vertex. In case (b) any combination of the dashed edges results in the blue vertex absorbing its neighbors. In case (c) no vertex can be absorbed.

function of the ILP. Therefore, the cost associated with $x_{v,i+1}$ must be folded into the cost of $x_{u,i}$, such that the same function is still computed. The variable $x_{u,i}$ also has to capture the distance to v, so instead of associating a cost of i with it, a cost of $2i+1$ is now associated with the variable. In this case, we say that a vertex u *absorbs* a vertex v and, in general, a vertex u may absorb $\alpha(u) \in \mathbb{N}$ many neighbors. In that case, u can fold the cost of these vertices into it by associating the cost $\alpha(u)(i+1)+i$ with $x_{u,i}$ in the objective function. The set $A \subseteq V$ denotes all absorbed vertices in G.

There are even more cases when a vertex u can absorb its neighbors. Let u be a cut vertex, i.e., $G-u$ has multiple connected components. If there is a component $C = \{w_1, \dots, w_\ell\} \subseteq D$ in $G-u$ such that all its vertices are dominated by u, then u can absorb $w_1, \dots, w_\ell$. In that case, each shortest path from $w_1, \dots, w_\ell$ to S^* must pass through u and the same argument from above applies. Figure 1 shows multiple cases where vertices can and cannot be absorbed. A proof that this search space reduction is correct is given in the next section.

Runtime Analysis. The dominated vertices can be determined in $\mathcal{O}(m\Delta(G))$ time by sorting each vertex's neighborhood and performing pairwise linear scans. To compute the set of absorbed vertices A, Tarjan's [23] $\mathcal{O}(n+m)$ biconnectivity algorithm identifies cut vertices, and for each cut vertex we check if it dominates a complete component. Since the vertex has at most $\Delta(G)$ neighbors and we already computed the domination relation between all neighboring vertices, this costs at most $\mathcal{O}(\Delta(G))$ time per vertex. For at most n cut-vertices this takes at most $\mathcal{O}(n\Delta(G))$ time. In total, a runtime of $\mathcal{O}(m\Delta(G))$ is required.

4.3 The Complete Algorithm

Grover starts by determining the set of dominated vertices D and the set of absorbed vertices $A \subseteq D$. Afterwards, GS-LS-C [3] is used to compute an initial solution $\widetilde{S}$. The values $\widetilde{d}(v) = \max\{\text{dist}(v, \widetilde{S}) + 1, 2\}$ are determined for each vertex v for the first iteration of our ILP. The ILP then reads as follows:

$$\sum_{v \in V \setminus D} x_{v,0} = k \tag{5a}$$

$$\min \sum_{\substack{v \in V \setminus A \\ i \in \{0,\dots,\widetilde{d}(v)\}}} x_{v,i}\left(\alpha(v)\cdot(i+1)+i\right) \quad \text{s. t.} \sum_{i \in \{0,\dots,\widetilde{d}(v)\}} x_{v,i} = 1 \quad \forall\, v \in V \setminus A \tag{5b}$$

$$\sum_{\substack{w \in V \setminus D \\ \text{dist}(v,w)=i}} x_{w,0} \geq x_{v,i} \quad \forall_{\substack{v \in V \setminus A \\ i \in \{0,\dots,\widetilde{d}(v)\}}} \tag{5c}$$

Note that in the worst case, the reduced ILP still has the same number of variables and constraints as ILPIND, namely $\mathcal{O}(n \cdot \mathrm{diam}(G))$. If, after solving the ILP, we encounter $x_{v,\widetilde{d}(v)} = 1 \ \wedge \ \mathrm{ecc}(v) > \widetilde{d}(v)$ for any $v \in V \setminus A$, then the ILP may be insufficient to represent an optimal solution. In this case, we increase $\widetilde{d}(v) \leftarrow \widetilde{d}(v) + 1$ for all vertices that did not pass the check and solve the ILP again with the new $\widetilde{d}(v)$-values. This is repeated until the ILP is sufficient and then the optimal solution $S^* = \{v \in V \setminus D \mid x_{v,0} = 1\}$ can be extracted.

Proof of Correctness. To prove the correctness of our algorithm, we show that the new ILP formulation indeed finds an optimal set S^*. We accomplish this by showing that the new ILP formulation and the ILPIND formulation (see Section 3.1) optimize the same function. Additionally, we show that we can infer the variable assignment of one ILP based on the variable assignment of the other. Since ILPIND computes an optimal set S^* (see [22] for a proof), so will our algorithm. Note that reducing the number of iterations by estimating the initial $\widetilde{d}(v)$-values does not impact the correctness guarantee.

▶ **Theorem 1.** *Let a reduced optimized ILP be sufficient, i.e., after optimizing the ILP it holds that $x_{v,\widetilde{d}(v)} = 0$ with $\widetilde{d}(v) < ecc(v)$ or $x_{v,\widetilde{d}(v)} = 1$ with $\widetilde{d}(v) \geq ecc(v)$. Then $S^* = \{v \in V \setminus D \mid x_{v,0} = 1\}$ is a globally optimal solution.*

Proof. Let $\{\widetilde{x}_{v,i} \mid 0 \leq i \leq \widetilde{d}(v)\}_{v \in V \setminus A}$ be the variables of the reduced model with values $\{\widetilde{d}(v)\}_{v \in V \setminus A}$ of the last iteration. By the definition of our algorithm, these values correspond to an optimum solution of the ILP and are sufficient, i.e., for each $v \in V \setminus A$, we have $\widetilde{x}_{v,\widetilde{d}(v)} = 0$ or both $\widetilde{x}_{v,\widetilde{d}(v)} = 1$ and $\mathrm{ecc}(v) \leq \widetilde{d}(v)$.

Consider the original ILP model used in ILPIND, for which we introduce variables $\{x_{v,i} \mid 0 \leq i \leq d(v)\}_{v \in V}$. Let $d(v) = \widetilde{d}(v)$ if $v \in V \setminus A$ and $d(v) = \widetilde{d}(\rho(v)) + 1$ if $v \in A$ where $\rho(v)$ denotes the vertex which absorbed v. We assign

$$x_{v,i} = \begin{cases} 0 & v \in A, i = 0 \\ \widetilde{x}_{\rho(v),i-1} & v \in A, i \geq 1 \\ \widetilde{x}_{v,i} & v \in V \setminus A \end{cases} \qquad \text{for } 0 \leq i \leq d(v). \tag{6}$$

This construction yields a feasible solution of the original ILP model (which is also sufficient by construction). It remains to show that this variable assignment corresponds to an optimum solution. First note that by the reduction rules, the identity

$$\sum_{v \in V \setminus A} \sum_{i \in \{0,\ldots,\widetilde{d}(v)\}} \alpha(v) \cdot (i+1) \cdot \widetilde{x}_{v,i} = \sum_{v \in A} \sum_{i \in \{0,\ldots,\widetilde{d}(\rho(v))\}} (i+1) \cdot \widetilde{x}_{\rho(v),i} \tag{7}$$

holds. Roughly speaking, this identity expresses that the absorbed cost can be counted either by considering the absorbing vertices, or by considering the absorbed vertices. For the objective values $\widetilde{f}_*$ of the solution of the reduced ILP, and f_* of the ILP used in ILPIND, we then get

$$
\begin{aligned}
\widetilde{f}_* &= \sum_{v \in V \setminus A} \sum_{i \in \{0,\dots,\widetilde{d}(v)\}} (\alpha(v) \cdot (i+1) + i) \cdot \widetilde{x}_{v,i} \\
&= \sum_{v \in V \setminus A} \sum_{i \in \{0,\dots,\widetilde{d}(v)\}} i \cdot \widetilde{x}_{v,i} + \sum_{v \in V \setminus A} \sum_{i \in \{0,\dots,\widetilde{d}(v)\}} \alpha(v) \cdot (i+1) \cdot \widetilde{x}_{v,i} \\
&\overset{(7)}{=} \sum_{v \in V \setminus A} \sum_{i \in \{0,\dots,\widetilde{d}(v)\}} i \cdot \widetilde{x}_{v,i} + \sum_{v \in A} \sum_{i \in \{0,\dots,\widetilde{d}(\rho(v))\}} (i+1) \cdot \widetilde{x}_{\rho(v),i} \\
&\overset{(\star)}{=} \sum_{v \in V \setminus A} \sum_{i \in \{0,\dots,d(v)\}} i \cdot x_{v,i} + \sum_{v \in A} \sum_{i \in \{0,\dots,d(v)\}} i \cdot x_{v,i} \\
&= \sum_{v \in V} \sum_{i \in \{0,\dots,d(v)\}} i \cdot x_{v,i} = f_*
\end{aligned}
\tag{8}
$$

where $(\star)$ follows by substituting in $d(v)$ and (6) together with an index shift. Hence, the constructed solution of the original ILP used in ILPIND has the same objective value $f_* = \widetilde{f}_*$ as the solution of the reduced ILP. However, we have yet to show that no strictly better solution of the original ILP exists. Therefore, assume for a contradiction that the constructed solution of the original model is *not* optimum, i.e., there is a feasible assignment of the variables of the original model with objective value $f' < f_*$. We refer to the corresponding variable assignment as $\{\hat{x}_{v,i} \mid 0 \leq i \leq d(v)\}_{v \in V}$. We argue that from this *better* solution, we can construct a *better* solution of the reduced ILP, contradicting its optimality. Indeed, using $\widetilde{x}_{v,i} = \hat{x}_{v,i}$ for all $v \in V \setminus A$ and $0 \leq i \leq d(v)$ as an assignment for the reduced ILP, we obtain a feasible solution of the reduced ILP. Applying the same rearrangements as in (8), we conclude that this solution has objective value $f' < \widetilde{f}_*$, a contradiction to the optimality of the solution of the reduced ILP. ◀

5 Approximation Results

In this section, we present two results concerning the approximation of GCCM. The first one is the use of dominating vertices in GS-LS-C [3], a 1/5-approximation algorithm. The second result is a proof that the GREEDY algorithm does not have an approximation guarantee.

Dominating Vertices in GS-LS-C. Due to the NP-hardness of GCCM, heuristic and approximation algorithms are the only viable option to compute solutions on graphs with millions of vertices. Angriman et al. [2] present an approximation algorithm that is based on a 5-approximation algorithm [5] for the k-median problem. They show that the algorithm translates to a 5-approximation of group farness and therefore a 1/5-approximation for GCCM. The algorithm is based on objective-improving vertex swaps. That is, as long as two vertices $s \in S$ and $o \in V \setminus S$ exist such that $f(\{o\} \cup S \setminus \{s\}) < f(S)$, then the current solution S is updated to $\{o\} \cup S \setminus \{s\}$. Only if no more improving swaps exist, the algorithm terminates and the aforementioned approximation can be guaranteed.

Angriman et al. [2] already restrict the search space by not considering swaps with degree-1 vertices. This optimization can be done since using the neighbor of a degree-1 vertex, instead of the vertex itself, never worsens the solution quality. The same idea can be extended to the previously introduced set of dominated vertices D. For each swap that considers a vertex $v \in D$ there is an equally good or better improving swap that utilizes a vertex $u \in V \setminus D$. Therefore, the approximation guarantee of Arya et al. [5] remains valid when restricting the search space to $V \setminus D$.

Greedy Does Not Approximate. As mentioned in the related work, Chen et al. [11] presented the GREEDY algorithm, which starts from the empty set $S_0 = \emptyset$ and iteratively adds the vertex that improves the score the most. That is, $S_i = S_{i-1} \cup \{v\}$ with $v = \arg\max_{v \in V \setminus S_{i-1}} c(S_{i-1} \cup \{v\})$. They mistakenly assumed that $c(\cdot)$ is submodular and that the $(1 - 1/e)$-approximation for submodular functions from Nemhauser et al. [17] holds. However, Bergamini et al. [8] showed that $c(\cdot)$ is not submodular and therefore the approximation claim cannot be supported by this proof. Note that the lack of submodularity does not necessarily mean the greedy strategy performs poorly. Bergamini et al. [8] observed very high empirical approximation ratios never lower than 0.97 in experiments on real-world graphs. While GREEDY performs well in practice, whether it does have an approximation guarantee was left unanswered. Here, we show that GREEDY can give solutions with arbitrarily bad approximation ratios, and it therefore has no constant factor approximation guarantee.

▶ **Theorem 2.** *GREEDY does not have a constant factor approximation guarantee.*

Proof sketch. We sketch the construction for $k = 2$ and refer the reader to Appendix (Section A) for the full proof. We construct a family of graphs parameterized by $r \in \mathbb{N}$ on which GREEDY can perform arbitrarily poorly. Let c be a vertex, and attach to c two vertex-disjoint paths of length $r - 1$ ending in vertices e_1 and e_2. Finally, attach r^2 degree-1 vertices to each of e_1 and e_2. In the first step, GREEDY chooses the central vertex c as it initially minimizes group farness the most. Afterwards e_1 and e_2 are equally well suited and w.l.o.g. GREEDY chooses e_1. The group farness of this solution is in $\Theta(r^3)$ as each of the r^2 degree-1 vertices of e_2 incurs a cost of r. The optimal solution $\{e_1, e_2\}$ only has group farness in $\Theta(r^2)$. Hence the approximation ratio is $\Theta(r)$ and becomes unbounded as $r \to \infty$, so GREEDY admits no constant-factor approximation guarantee. ◀

6 Experimental Evaluation

We experimentally evaluate our developed techniques to assess their impact on real-world graphs. First, we evaluate each technique on its own (Section 6.1) before comparing GROVER to current state-of-the-art solvers (Section 6.2). In Section 6.3, we evaluate the impact of using dominating vertices in GS-LS-C [2] to restrict the search space.

Data and Experimental Setup. We use 48 graphs from Konect [16] and the Network Data Repository [19], covering diverse real-world networks. A subset of them was also used in the experimental analysis of ILPIND [22]. They range from small graphs with about 50 vertices to graphs with more than 50 000 vertices. Table 1 in the Appendix holds a complete list with detailed statistics. The experiments are run on an Intel Xeon Silver 4216 (2.10GHz, 92 GiB RAM, Ubuntu 24.04.3). We test $k \in \{2, \ldots, 20\}$ with a 10-minute limit per instance, totaling 912 instances. We exclude $k = 1$ since all algorithms reduce to GREEDY, and report runtimes (excluding I/O) averaged over five runs.

Solvers. ILPIND [22] is the state-of-the-art algorithm for GCCM. We modified it to use Gurobi 12.0.1 [14]. DVIND [22] is the fastest explicit branch-and-bound algorithm to solve GCCM and it also deploys dominating vertices. Both ILPIND and DVIND are implemented in Kotlin and run using Java openjdk 21.0.8. SUBMODST [25] is a branch-and-bound algorithm for general submodular function maximization. It employs more pruning techniques and optimizations, but does not utilize dominating vertices. It is implemented in C++17. Our algorithm GROVER also utilizes Gurobi 12.0.1 and is implemented in C++20. In the

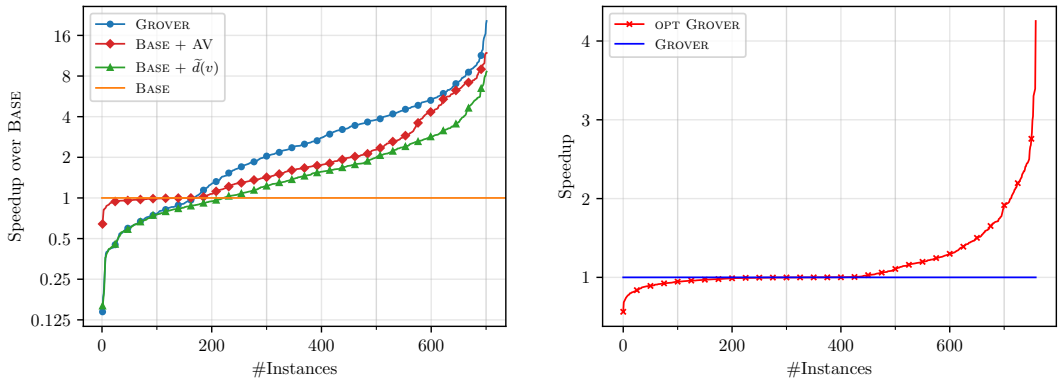


Figure 2 (a) Speedup of different optimizations over Base. Base uses dominating vertices and $d(v) = 2$. Absorbing vertices AV and $\widetilde{d}(v)$ are evaluated separately. Grover uses both. (b) Speedup of Grover initialized with an optimal solution compared to the default initialization with GS-LS-C [3]. Speedups are determined per instance and then sorted.

approximate setting we compare against GS-LS-C [3], which is implemented in the C++ library NetworKit [3]. All C++ solvers are compiled using gcc 15.1.0 and -O3 optimization. Note that PRESTO [18] is not publicly available and their experiments show that it does not scale for $k > 7$.

6.1 Absorbed Vertices and Improved $d(v)$

In this section, we evaluate the effect of absorbing vertices and initially approximating the $d(v)$-values. Recall that absorbing vertices is a data reduction technique that allows us to remove variables and corresponding constraints from the ILPs. Approximating the $d(v)$-values should overall result in fewer iterations of the algorithm and therefore improve the running time. Additionally, we compare the runtime of Grover when it is initialized by GS-LS-C [3] and when it is initialized with an optimal solution. This provides insight into how much more speedup one can expect by utilizing better initial solutions.

First, we compare the effect of absorbing vertices (AV) and approximating the $d(v)$-values. The Base algorithm only uses dominating vertices and the default $d(v) = 2$ initialization. We compare to Base + AV which additionally enables absorbing vertices, but keeps $d(v) = 2$. The algorithm Base + $\widetilde{d}(v)$ approximates the $d(v)$-values, but does not enable absorbing vertices. Finally, Grover enables absorbing vertices and approximates the $d(v)$-values. Figure 2a shows the speedup of each configuration over Base. For each instance the speedup is determined and these are sorted increasingly. Note that these speedups are determined on the 701 instances all configurations are able to solve. Base + AV, Base + $\widetilde{d}(v)$ and Grover solve 713, 750 and 759 instances respectively. Table 3 in the Appendix shows more detailed results for each graph. Base is overall the slowest configuration as the other configurations have a mean speedup of 1.18×, 1.53× and 2.19× respectively. Note that these mean speedups are however skewed by many easy-to-solve instances, for which the overhead of computing GS-LS-C has a detrimental effect on the total runtime. This can be seen for Base + $\widetilde{d}(v)$ and Grover in Figure 2a. Therefore, we also compare the speedup measured as total sum of time needed to solve all 701 instances, and simply refer to this measure as *speedup*, while we refer to the geometric mean speedup as *mean speedup*. Then, relative to Base, Base + AV has a speedup of 2.07×, while Base + $\widetilde{d}(v)$ has a speedup of 1.41×.

GROVER has a speedup of 3.20×. For harder-to-solve instances that require more time, our improvements have a greater effect in reducing the runtime. While GROVER has the largest speedup, it is not the fastest configuration for each instance. In total, 760 instances were solved by at least one configuration. Out of these, GROVER is the fastest on 411, while BASE + AV is the fastest on 185 instances. BASE is the fastest configuration on 109, while BASE + $\widetilde{d}(v)$ is the fastest on 55 instances. For larger instances, GROVER is the fastest algorithm, however for smaller instances, there is no clear winner.

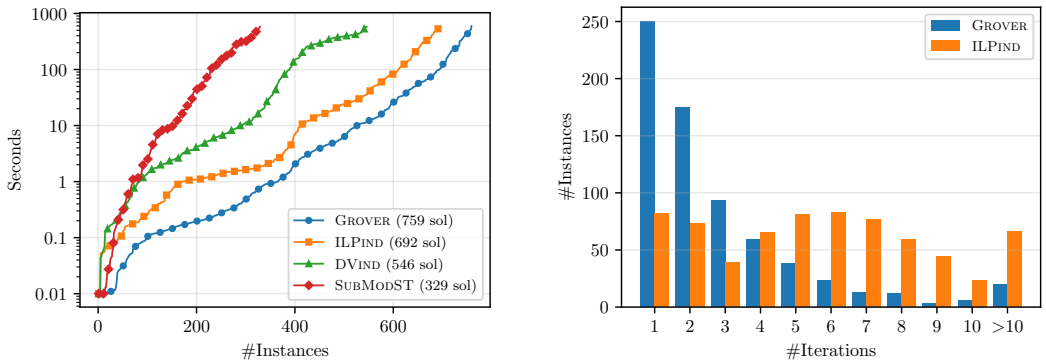
Secondly, we compare GROVER when initialized with GS-LS-C versus an optimal solution, to assess how much room for improvement remains in the initialization. A large difference between the running times indicates that better initial solutions will improve the running time of GROVER. However, small differences show that the running time cannot be further improved by other initializations. For this experiment, we removed the time it costs GROVER to run GS-LS-C, since OPT GROVER has no additional cost to obtain the optimal solution. Therefore, only the time it costs to solve the ILPs is measured. Figure 2b shows the corresponding speedups, while Table 3 in the Appendix shows a more detailed view. The speedups are computed per instance and sorted afterwards. Out of the 759 instances, 401 are solved faster when initialized with an optimal solution while 358 are solved slower. The smallest speedup achieved is 0.56× for INF-EUROROAD and $k = 2$, however the mean speedup across all slower instances is 0.95×, the loss in time therefore is negligible. Overall, the speedup is 1.14× while the mean speedup is 1.13×. Initialization with an optimal solution does not greatly improve the overall running time of the ILP, indicating that the approximate initialization with GS-LS-C [3] is already of high quality.

6.2 Comparison to SOTA

We compare our algorithm GROVER against ILPIND [22], DVIND [22] and SUBMODST [25]. Figure 3a shows the number of instances each algorithm is able to solve. Table 2 in the Appendix holds more detailed running times. While SUBMODST performs very fast for the first few instances, it is overall the slowest algorithm and only solves 329 instances. DVIND performs better as it solves 546 instances, but still falls behind both of the iterative ILP algorithms. ILPIND solves 692 instances while GROVER solves 759 out of the 912 possible instances. GROVER has a mean speedup of 4.59× compared to ILPIND and a maximum speedup of 34.16×. Only on 22 out of the 692 instances solved by both solvers is GROVER slower. In this case, GROVER's worst speedup is 0.40× while the mean speedup on those 22 instances is 0.74×. However, each instance solved by ILPIND is also solved by GROVER.

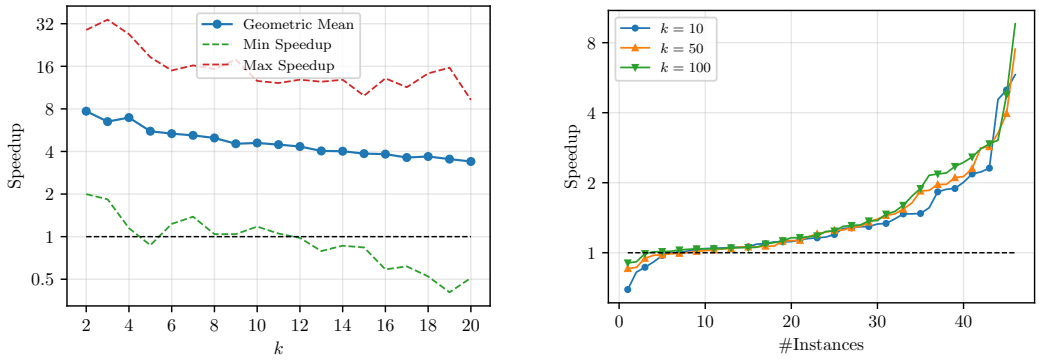
Figure 3b shows how many instances GROVER and ILPIND solve and how many ILP iterations are necessary. This plot only includes the 692 instances that both GROVER and ILPIND are able to solve. GROVER is able to solve significantly more instances in fewer iterations than ILPIND. About 36% of instances are solved within one iteration, while ILPIND only solves about 11% within one iteration. For more than 50% of instances, ILPIND needs 6 or more iterations while GROVER needs 6 or more iterations for only about 11% of all instances. Note that there is one instance, ECON-MAHINDAS and $k = 20$, where ILPIND only requires three iterations, while GROVER needs four. In total, GROVER greatly reduces the number of needed iterations, which was the goal of approximating the $d(v)$-values.

Figure 4a shows the geometric mean, minimum, and maximum speedup of GROVER over ILPIND across k. For each k all instances solved by both algorithms are included in this comparison. The mean speedup peaks at 7.91× for $k = 2$ and decreases to 3.80× for $k = 20$, with a similar trend for min and max values. The maximum speedup reaches 34.16× at $k = 3$ and drops to 9.25× at $k = 20$. For $k \leq 11$ (except $k = 5$), the minimum speedup exceeds 1, meaning GROVER is never slower, while for $k \geq 12$ it can be slower, with a minimum of 0.40 at $k = 19$. Note however that GROVER is only slower on 22 out of 692 instances.



(a) (b)

Figure 3 (a) Number of solved instances for GROVER, ILPIND [22], DVIND [22], and SUBMODST [25]. (b) Distribution of the number of iterations that GROVER (ours) and ILPIND [22] need to solve all instances.



(a) (b)

Figure 4 (a) Speedup of GROVER over ILPIND [22] for different values of k. (b) Runtime ratio of GS-LS-C [2] without dominating vertices and with dominating vertices.

6.3 Dominating Vertices in GS-LS-C

Since using dominating vertices provably restricts the search space in the exact case, we also want to explore the search space restriction in an approximate setting. We therefore compare the running time and solution quality of GS-LS-C [2] with and without the restriction. We test the values of $k \in \{10, 50, 100\}$ on each graph with more than 100 vertices. These group sizes were also used in the experimental evaluation of GS-LS-C. Each graph-k instance is run 5 times and the required time and achieved group farness are averaged. Figure 4b shows the speedups. For $k = 10$, the mean speedup is 1.54× (min 0.69×, max 5.83×), with slowdowns on small instances due to the overhead of identifying dominated vertices. This overhead diminishes for larger instances: for $k = 50$ and $k = 100$, mean speedups increase to 1.59× and 1.71×, with minima 0.85×, 0.90× and maxima 7.51×, 9.64×. The solution quality largely remains unaffected. In fact, the average increase in group farness is smaller than 0.01% for all values of k. Overall, restricting the search space in GS-LS-C [2] improves the running time without altering solution quality. Including the restriction in other approximation algorithms will likely lead to the same results.

7 Conclusion and Future Work

Group closeness centrality maximization asks for a set of k vertices, such that the distance to all other vertices in the graph is minimized. In this work, we introduced two new techniques. The first one, absorbed vertices, proves that certain vertices have their distance to any optimal solution determined by a neighbor. This leads to a smaller ILP as the variables associated with the absorbed vertices can be removed. The second technique bootstraps the ILP using an approximate solution to produce near-sufficient ILPs. This leads the algorithm to solve many instances in only one iteration and overall greatly reduces the number of required iterations. Our algorithm Grover has a mean speedup of 4.59× and can achieve speedups of up to 34.16× over the state-of-the-art algorithm ILPind [22]. We proved that restricting the search space of GS-LS-C [3] to non-dominated vertices preserves its 1/5-approximation guarantee, achieving speedups of up to 9.64×. Additionally, we settled an open question by proving that a well-known greedy algorithm [11] does not give a constant factor approximation. Future work could explore if the techniques proposed here can be applied to more general problems, such as the weighted version of the problem studied here.

References

1 Isaiah G. Adebayo and Yanxia Sun. A novel approach of closeness centrality measure for voltage stability analysis in an electric power grid. *International Journal of Emerging Electric Power Systems*, 21(3):20200013, 2020. doi:10.1515/ijeeps-2020-0013.

2 Eugenio Angriman, Ruben Becker, Gianlorenzo D'Angelo, Hugo Gilbert, Alexander van der Grinten, and Henning Meyerhenke. Group-Harmonic and Group-Closeness Maximization - Approximation and Engineering. In Martin Farach-Colton and Sabine Storandt, editors, *Proceedings of the 23rd Symposium on Algorithm Engineering and Experiments, ALENEX 2021, Virtual Conference, January 10-11, 2021*, pages 154–168. SIAM, 2021. doi:10.1137/1.9781611976472.12.

3 Eugenio Angriman, Alexander van der Grinten, Michael Hamann, Henning Meyerhenke, and Manuel Penschuck. Algorithms for Large-scale Network Analysis and the NetworKit Toolkit. In Hannah Bast, Claudius Korzen, Ulrich Meyer, and Manuel Penschuck, editors, *Algorithms for Big Data - DFG Priority Program 1736*, volume 13201 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2022. doi:10.1007/978-3-031-21534-6_1.

4 Eugenio Angriman, Alexander van der Grinten, and Henning Meyerhenke. Local Search for Group Closeness Maximization on Big Graphs. In Chaitanya K. Baru, Jun Huan, Latifur Khan, Xiaohua Hu, Ronay Ak, Yuanyuan Tian, Roger S. Barga, Carlo Zaniolo, Kisung Lee, and Yanfang (Fanny) Ye, editors, *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*, pages 711–720. IEEE, 2019. doi:10.1109/BIGDATA47090.2019.9006206.

5 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local Search Heuristics for k-Median and Facility Location Problems. *SIAM J. Comput.*, 33(3):544–562, 2004. doi:10.1137/S0097539702416402.

6 Alex Bavelas. A mathematical model for group structures. *Applied Anthropology*, 7(3):16–30, 1948. URL: http://www.jstor.org/stable/44135428.

7 Murray A. Beauchamp. An improved index of centrality. *Behavioral Science*, 10(2):161–163, 1965. doi:10.1002/bs.3830100205.

8 Elisabetta Bergamini, Tanya Gonser, and Henning Meyerhenke. Scaling up Group Closeness Maximization. *CoRR*, abs/1710.01144, 2017. doi:10.48550/arXiv.1710.01144.

9 Stephen P. Borgatti, Martin G. Everett, and Jeffrey C. Johnson. *Analyzing social networks*. SAGE, 2013.

10 Eric Chea and Dennis R. Livesay. How accurate and statistically robust are catalytic site predictions based on closeness centrality? *BMC Bioinform.*, 8, 2007. doi:10.1186/1471-2105-8-153.

11 Chen Chen, Wei Wang, and Xiaoyang Wang. Efficient Maximum Closeness Centrality Group Identification. In Muhammad Aamir Cheema, Wenjie Zhang, and Lijun Chang, editors, *Databases Theory and Applications - 27th Australasian Database Conference, ADC 2016, Sydney, NSW, Australia, September 28-29, 2016, Proceedings*, volume 9877 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2016. doi:10.1007/978-3-319-46922-5_4.

12 M. G. Everett and S. P. Borgatti. The centrality of groups and classes. *The Journal of Mathematical Sociology*, 23(3):181–201, 1999. doi:10.1080/0022250X.1999.9990219.

13 Suning Gong, Qingqin Nong, Tao Sun, Qizhi Fang, Ding-Zhu Du, and Xiaoyu Shao. Maximize a monotone function with a generic submodularity ratio. *Theor. Comput. Sci.*, 853:16–24, 2021. doi:10.1016/J.TCS.2020.05.018.

14 Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, 2025. Available at. URL: https://www.gurobi.com.

15 Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, Stefan Richter, Dagmar Tenfelde-Podehl, and Oliver Zlotowski. *Centrality Indices*, pages 16–61. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. doi:10.1007/978-3-540-31955-9_3.

16 Jérôme Kunegis. KONECT: the koblenz network collection. In Leslie Carr, Alberto H. F. Laender, Bernadette Farias Lóscio, Irwin King, Marcus Fontoura, Denny Vrandecic, Lora Aroyo, José Palazzo M. de Oliveira, Fernanda Lima, and Erik Wilde, editors, *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. doi:10.1145/2487788.2488173.

17 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

18 Baibhav Rajbhandari, Paul Olsen Jr., Jeremy Birnbaum, and Jeong-Hyon Hwang. PRESTO: Fast and Effective Group Closeness Maximization. *IEEE Trans. Knowl. Data Eng.*, 35(6):6209–6223, 2023. doi:10.1109/TKDE.2022.3178925.

19 Ryan A. Rossi and Nesreen K. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 4292–4293. AAAI Press, 2015. doi:10.1609/AAAI.V29I1.9277.

20 Ron Rymon. Search through Systematic Set Enumeration. In Bernhard Nebel, Charles Rich, and William R. Swartout, editors, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, USA, October 25-29, 1992*, pages 539–550. Morgan Kaufmann, 1992.

21 Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966. doi:10.1007/BF02289527.

22 Luca Pascal Staus, Christian Komusiewicz, Nils Morawietz, and Frank Sommer. Exact Algorithms for Group Closeness Centrality. In Jonathan W. Berry, David B. Shmoys, Lenore Cowen, and Uwe Naumann, editors, *SIAM Conference on Applied and Computational Discrete Algorithms, ACDA 2023, Seattle, WA, USA, May 31 - June 2, 2023*, pages 1–12. SIAM, 2023. doi:10.1137/1.9781611977714.1.

23 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

24 Martijn P. van den Heuvel and Olaf Sporns. Network hubs in the human brain. *Trends in Cognitive Sciences*, 17(12):683–696, 2013. doi:10.1016/j.tics.2013.09.012.

25 Henning Martin Woydt, Christian Komusiewicz, and Frank Sommer. SubModST: A Fast Generic Solver for Submodular Maximization with Size Constraints. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, Royal Holloway, London, United Kingdom, September 2-4, 2024*, volume 308 of *LIPIcs*, pages 102:1–102:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ESA.2024.102.

A Greedy Does Not Approximate

▶ **Theorem 3.** *Greedy does not have a constant factor approximation guarantee.*

Proof. We show the claim by constructing a family of graphs $(G_r)_{r\in\mathbb{N}}$ for which Greedy may produce arbitrarily poor approximations for $k = 2$. A graph G_r of the family contains a path P with end vertices e_1 and e_2. P has an odd number of vertices (precisely $2r - 1$) and therefore has a central vertex c. Attached to e_1 and e_2 are r^2 leaves (degree 1 vertices), respectively. Figure 5 shows an illustration of such a graph. Let us denote by S_r^G and S_r^* the solution produced by Greedy and the exact solution for G_r with $k = 2$, respectively. Recall that $f(S) = \sum_{v\in V(G)} \mathrm{dist}(v, S)$ denotes the group farness of S. Now consider the solution of Greedy. The algorithm first chooses the vertex c as it is the most central one. Afterwards, the vertices e_1 and e_2 are equally well suited to be chosen and w.l.o.g. we assume the algorithm chooses e_1 and get $S_r^G = \{c, e_1\}$. The r^2 leaves attached to e_2 each contribute a cost of r to $f(S_r^G)$. Therefore, we get $f(S_r^G) \geq r^3$. On the other hand, consider the solution $S_e \coloneqq \{e_1, e_2\}$. Clearly, $f(S_r^*) \leq f(S_e)$. We argue that $f(S_e)$ is upper bounded by a quadratic expression in r and thus $f(S_r^*)$ is as well. Indeed, the leaves attached to e_1 and e_2 incur a total cost of $2r^2$ to $f(S_e)$. It remains to consider the cost incurred by the vertices in P, which is upper bounded by $2\sum_{i=1}^{r-1} i = r^2 - r$. Therefore, overall $f(S_r^*) \leq f(S_e) \leq 3r^2 - r$ and hence we get

$$\frac{f(S_r^G)}{f(S_r^*)} \geq \frac{r^3}{3r^2 - r} \to \infty \quad \text{for} \quad r \to \infty$$

and similarly $\frac{c(S_r^*)}{c(S_r^G)} \to \infty$ for $r \to \infty$, showing the claim. ◀

The proof is of course independent of the numerator x used to define $c(S) = x/f(S)$. Note that while we assumed $k = 2$, similar families of graphs can be constructed for any integer constant $k > 2$: Let a connected component of $G_r - c$ be called a *flower*. By starting with c and attaching k flowers to c, we obtain graphs to which the same argument applies. Since c is always chosen as the first vertex, there is at least one flower from which no vertex is

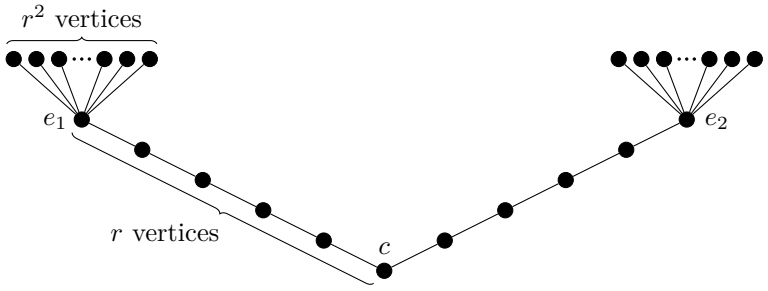


Figure 5 Sketch of a graph G_r of the family $(G_r)_{r\in\mathbb{N}}$ used to prove that Greedy does not have an approximation guarantee.

chosen, which incurs a cost in $\Theta(r^3)$. An optimal strategy chooses the $e_i, i = 1, \ldots, k$ vertices and therefore only incurs a cost in $\Theta(r^2)$.

Gong et al. [13] analyze the simple greedy algorithm for non-negative monotone set functions with *generic submodularity ratio* γ, a scalar measuring how close a function is to being submodular with $\gamma \in [0, 1]$, where $\gamma = 1$ if and only if the function is submodular. While Bergamini et al. [8] already showed that $c(\cdot)$ is not submodular, i.e. $\gamma < 1$, our result actually implies the following stronger result.

▶ **Corollary 4.** *The group closeness centrality function $c(\cdot)$ can have a submodularity ratio γ arbitrarily close to zero.*

Proof. Let g be a non-negative monotone set function with generic submodularity ratio γ. Let S_G be the set computed by the simple greedy algorithm with $k \in \mathbb{N}$. Gong et al. [13] show that the simple greedy algorithm obtains the approximation guarantee $g(S_G) \geq (1 - e^{-\gamma})g(S^*)$, where S^* denotes the optimal solution. By Theorem 2, in the expression $c(S_G) = \alpha \cdot c(S^*)$, $\alpha \in \mathbb{R}$ may get arbitrarily close to zero, implying the same for γ. ◀

B Graphs and Results

Table 1 Graphs used in the experimental evaluation. #dom and #abs denote the numbers of dominated and absorbed vertices, respectively; dens is the graph density and diam its diameter.

Graph	n	m	dens	diam	#dom	#abs	Graph	n	m	dens	diam	#dom	#abs
contiguous-usa	49	107	0.091	11	13	1	fb-pages-tvshow	3 892	17 239	0.002	20	1 690	688
brain_1	65	730	0.351	3	17	0	ca-GrQc	4 158	13 422	0.002	17	2 712	1 171
arenas-jazz	198	2 742	0.141	6	93	5	inf-power	4 941	6 594	<0.001	46	1 487	1 278
ca-netscience	379	914	0.013	17	306	128	ca-Erdos992	4 991	7 428	<0.001	14	3 861	3 516
robot24c1_mat5	404	14 261	0.175	3	10	0	soc-advogato	5 054	39 374	0.003	9	1 721	1 080
tortoise	496	984	0.008	21	232	140	fb-pages-politician	5 908	41 706	0.002	14	1 922	644
econ-beause	507	39 427	0.307	3	490	1	ia-reality	6 809	7 680	<0.001	8	6 391	6 284
bio-diseasome	516	1 188	0.009	15	379	204	bio-dmela	7 393	25 569	<0.001	11	2 042	2 005
soc-wiki-Vote	889	2 914	0.007	13	239	193	rt_onedirection	7 987	8 100	<0.001	11	7 725	7 717
ca-CSphd	1 025	1 043	0.002	28	697	693	bio-grid-human	9 186	31 038	<0.001	14	3 425	2 843
inf-euroroad	1 039	1 305	0.002	62	132	127	rt_lolgop	9 765	10 075	<0.001	10	9 282	9 142
email-univ	1 133	5 451	0.009	8	232	153	ia-escorts-dynamic	10 106	39 016	<0.001	10	1 857	1 838
econ-mahindas	1 258	7 513	0.010	8	4	0	ca-HepPh	11 204	117 619	0.002	13	7 014	1 904
bio-yeast	1 458	1 948	0.002	19	800	735	web-indochina-2004	11 358	47 606	<0.001	27	9 794	7 234
comsol	1 500	48 119	0.043	12	1 485	0	soc-anybeat	12 645	49 132	<0.001	10	8 430	6 320
medulla_1	1 770	8 905	0.006	6	836	392	fb-pages-company	14 113	52 126	<0.001	15	4 568	2 511
rt_assad	2 139	2 786	0.001	14	1 577	1 562	econ-poli-large	15 575	17 468	<0.001	15	12 970	12 515
bio-WormNet-v3	2 274	78 328	0.030	11	2 080	66	ca-AstroPh	17 903	196 972	0.001	14	10 462	1 701
heart2	2 339	340 229	0.124	6	2 327	0	ca-CondMat	21 363	91 286	<0.001	15	14 282	3 386
econ-orani678	2 529	86 768	0.027	5	42	0	ca-cit-HepTh	22 721	2 444 642	0.009	9	10 536	317
road-minnesota	2 640	3 302	<0.001	99	97	95	fb-pages-media	27 917	205 964	<0.001	15	6 330	2 364
inf-openflights	2 905	15 645	0.004	14	1 783	806	soc-gemsec-RO	41 773	125 826	<0.001	19	6 626	5 490
web-edu	3 031	6 474	0.001	11	2 768	2 715	soc-gemsec-HU	47 538	222 887	<0.001	14	3 768	2 725
econ-psmigr3	3 140	410 781	0.083	3	227	0	fb-pages-artist	50 515	819 090	<0.001	11	4 967	3 168

Table 2 Per-graph comparison of SubModST [25], DVind [22], ILPind [22] and Grover (ours). For each graph, #Solved reports the number of k values for which an algorithm solved the instance, and $\sum$Seconds is the cumulative runtime over all successfully solved instances. Note that different algorithms may solve different subsets of k values. Bold entries indicate the algorithm that solved the most instances and, in case of a tie, achieved the lowest total runtime.

Graph	SubModST		DVind		ILPind		Grover	
	#Solved	$\sum$Seconds	#Solved	$\sum$Seconds	#Solved	$\sum$Seconds	#Solved	$\sum$Seconds
contiguous-usa	19	737.35	19	19.09	19	1.45	**19**	**0.18**
brain_1	19	110.89	19	23.51	19	1.93	**19**	**0.21**
arenas-jazz	12	261.39	14	678.52	19	3.82	**19**	**0.63**
ca-netscience	12	899.63	19	8.39	19	4.67	**19**	**1.03**
robot24c1_mat5	8	475.52	6	163.96	19	15.60	**19**	**7.20**
tortoise	9	325.62	18	1194.34	19	30.27	**19**	**9.77**
econ-beause	8	657.02	19	9.35	19	9.74	**19**	**1.20**
bio-diseasome	11	808.09	19	37.80	19	9.28	**19**	**2.47**
soc-wiki-Vote	11	582.12	19	897.86	19	29.54	**19**	**3.45**
ca-CSphd	6	360.66	19	43.00	19	64.20	**19**	**6.93**
inf-euroroad	4	137.41	13	681.26	19	747.87	**19**	**285.02**
email-univ	9	282.53	8	600.03	19	1580.24	**19**	**698.38**
econ-mahindas	9	489.03	9	1235.96	19	59.13	**19**	**14.12**
bio-yeast	9	246.89	18	1414.46	19	173.75	**19**	**40.10**
comsol	2	45.50	13	233.12	19	33.33	**19**	**24.88**
medulla_1	19	41.25	19	120.37	19	24.75	**19**	**3.78**
rt_assad	11	726.40	19	104.49	19	49.99	**19**	**3.54**
bio-WormNet-v3	5	632.09	11	401.14	19	48.85	**19**	**37.33**
heart2	2	252.99	10	212.30	19	33.35	**19**	**16.66**
econ-orani678	12	1494.00	10	652.86	19	68.09	**19**	**16.49**
road-minnesota	2	274.75	**8**	**2606.73**	-	-	2	681.46
inf-openflights	10	544.23	9	1070.32	19	333.10	**19**	**53.83**
web-edu	19	208.18	19	46.54	19	33.03	**19**	**4.70**
econ-psmigr3	8	292.08	6	489.65	19	1215.21	**19**	**779.61**
fb-pages-tvshow	7	351.56	12	1750.34	19	1068.56	**19**	**246.12**
ca-GrQc	6	357.37	8	756.79	19	4482.46	**19**	**1409.26**
inf-power	2	91.74	12	5160.91	16	3048.41	**18**	**1490.94**
ca-Erdos992	6	422.38	10	606.08	19	865.37	**19**	**265.08**
soc-advogato	10	1267.51	12	1882.45	19	489.18	**19**	**94.97**
fb-pages-politician	6	676.17	9	2611.87	19	1703.48	**19**	**358.77**
ia-reality	8	1172.90	19	161.17	19	48.34	**19**	**4.52**
bio-dmela	7	1532.54	4	1670.15	2	1094.27	**12**	**2582.48**
rt_onedirection	5	880.90	19	132.52	19	28.15	**19**	**2.61**
bio-grid-human	6	1770.32	-	-	19	6525.33	**19**	**700.97**
rt_lolgop	19	6029.30	19	192.43	19	29.90	**19**	**4.73**
ia-escorts-dynamic	6	2483.43	-	-	9	4267.95	**19**	**1915.29**
ca-HepPh	4	1944.25	3	1693.44	-	-	**5**	**1627.93**
web-indochina-2004	1	498.81	5	1098.12	19	389.42	**19**	**177.32**
soc-anybeat	-	-	19	7603.85	19	659.06	**19**	**125.06**
fb-pages-company	-	-	-	-	-	-	**15**	**3474.49**
econ-poli-large	-	-	16	4605.45	19	416.15	**19**	**72.00**
ca-AstroPh	-	-	-	-	-	-	-	-
ca-CondMat	-	-	-	-	-	-	**4**	**1727.81**
ca-cit-HepTh	-	-	-	-	-	-	**17**	**4769.13**
fb-pages-media	-	-	-	-	-	-	**2**	**1088.01**
soc-gemsec-RO	-	-	-	-	-	-	-	-
soc-gemsec-HU	-	-	-	-	-	-	-	-
fb-pages-artist	-	-	-	-	-	-	-	-

Table 3 Per-graph comparison of GROVER and its different configurations. Bold entries indicate the configuration that solved the most instances and in case of a tie, achieved the lowest total runtime. OPT. GROVER is not included in this comparison.

Graph	BASE		BASE + AV		BASE + $\tilde{d}(v)$		GROVER		OPT. GROVER	
	#Solved	∑Seconds	#Solved	∑Seconds	#Solved	∑Seconds	#Solved	∑Seconds	#Solved	∑Seconds
contiguous-usa	19	0.18	19	0.19	19	0.18	**19**	**0.18**	19	0.16
brain_1	19	0.18	**19**	**0.18**	19	0.20	19	0.21	19	0.21
arenas-jazz	19	0.58	**19**	**0.52**	19	0.53	19	0.63	19	0.40
ca-netscience	19	1.67	19	1.08	19	1.34	**19**	**1.03**	19	0.79
robot24c1_mat5	**19**	**5.55**	19	5.57	19	7.33	19	7.20	19	2.99
tortoise	19	22.84	19	13.91	19	15.55	**19**	**9.77**	19	7.65
econ-beause	**19**	**0.59**	19	0.62	19	1.15	19	1.20	19	0.70
bio-diseasome	19	5.02	**19**	**2.44**	19	3.89	19	2.47	19	1.68
soc-wiki-Vote	19	17.31	19	8.22	19	8.06	**19**	**3.45**	19	2.76
ca-CSphd	19	43.84	19	13.42	19	25.26	**19**	**6.93**	19	6.15
inf-euroroad	19	633.12	19	526.58	19	325.68	**19**	**285.02**	19	292.31
email-univ	19	1596.33	19	736.96	19	1285.73	**19**	**698.38**	19	365.96
econ-mahindas	19	21.32	19	21.32	**19**	**13.95**	19	14.12	19	8.51
bio-yeast	19	150.92	19	79.70	19	70.81	**19**	**40.10**	19	32.03
comsol	**19**	**20.76**	19	20.80	19	25.50	19	24.88	19	10.14
medulla_1	19	9.99	19	5.27	19	6.29	**19**	**3.78**	19	2.39
rt_assad	19	26.74	19	7.23	19	12.33	**19**	**3.54**	19	2.72
bio-WormNet-v3	19	21.84	**19**	**16.00**	19	38.27	19	37.33	19	6.07
heart2	**19**	**6.41**	19	6.73	19	16.47	19	16.66	19	6.10
econ-orani678	**19**	**16.14**	19	16.24	19	16.44	19	16.49	19	10.22
road-minnesota	-	-	2	1091.03	1	289.35	**2**	**681.46**	2	630.75
inf-openflights	19	248.31	19	165.27	19	66.68	**19**	**53.83**	19	39.03
web-edu	19	12.02	**19**	**1.95**	19	8.13	19	4.70	19	0.81
econ-psmigr3	19	942.70	19	941.32	19	792.62	**19**	**779.61**	19	516.21
fb-pages-tvshow	19	831.11	19	519.97	19	350.16	**19**	**246.12**	19	202.62
ca-GrQc	19	4205.63	19	2528.51	19	1903.46	**19**	**1409.26**	19	1002.81
inf-power	16	2614.84	17	2195.14	17	1819.69	**18**	**1490.94**	18	1367.21
ca-Erdos992	19	956.03	19	419.74	19	574.49	**19**	**265.08**	19	224.66
soc-advogato	19	280.71	19	165.65	19	114.31	**19**	**94.97**	19	86.78
fb-pages-politician	19	1103.28	19	809.97	19	391.78	**19**	**358.77**	19	319.89
ia-reality	19	24.71	**19**	**3.60**	19	20.28	19	4.52	19	2.22
bio-dmela	2	1058.13	8	3491.01	11	2891.56	**12**	**2582.48**	13	2289.61
rt_onedirection	19	12.14	**19**	**1.23**	19	10.10	19	2.61	19	0.86
bio-grid-human	19	4382.76	19	3414.61	19	1413.84	**19**	**700.97**	19	568.92
rt_lolgop	19	15.81	**19**	**2.11**	19	16.09	19	4.73	19	1.33
ia-escorts-dynamic	18	7058.83	19	4815.43	19	2035.28	**19**	**1915.29**	19	1841.93
ca-HepPh	-	-	-	-	3	828.59	**5**	**1627.93**	5	1362.08
web-indochina-2004	19	346.68	**19**	**143.52**	19	247.44	19	177.32	19	52.91
soc-anybeat	19	496.25	19	286.42	19	161.94	**19**	**125.06**	19	103.91
fb-pages-company	-	-	2	996.48	15	4061.42	**15**	**3474.49**	16	3403.25
econ-poli-large	19	340.90	19	89.17	19	226.82	**19**	**72.00**	19	41.11
ca-AstroPh	-	-	-	-	-	-	-	-	-	-
ca-CondMat	-	-	-	-	2	1084.54	**4**	**1727.81**	4	1688.73
ca-cit-HepTh	-	-	-	-	17	5801.30	**17**	**4769.13**	18	4096.97
fb-pages-media	-	-	-	-	-	-	**2**	**1088.01**	2	1034.82
soc-gemsec-RO	-	-	-	-	-	-	-	-	-	-
soc-gemsec-HU	-	-	-	-	-	-	-	-	-	-
fb-pages-artist	-	-	-	-	-	-	-	-	-	-