Exact Algorithms for Group Closeness Centrality*

Luca Pascal Staus† Christian Komusiewicz‡ Nils Morawietz†§ Frank Sommer†¶

Abstract

The Group Closeness Centrality problem asks, given a graph G and an integer k, for a vertex set S of size k such that the sum of distances from the vertices of $V \setminus S$ to S is minimal. Being a generalization of the NP-hard Dominating Set problem, Group Closeness Centrality is NP-hard as well and W[2]-hard with respect to k meaning that it presumably has no algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$. We first show that, in contrast to Dominating Set, Group Closeness Centrality remains W[2]-hard when restricted to graphs with constant maximum degree. We then develop and evaluate two new exact algorithms for Group Closeness Centrality, one based on branch-and-bound and one based on a new ILP formulation. We further show how to embed both approaches in an iterative method that allows to solve the problem without computing all pairwise distances in G. Our experiments show that on small and medium-sized real-world networks, the new ILP formulation substantially outperforms a previous ILP formulation and that the branch-and-bound algorithm is competitive with the new ILP formulation for small values of k.

1 Introduction

The identification of important actors or groups of actors is a critical task in social network analysis [12, 11, 22, 33]. A standard approach for translating this task into a computational optimization problem is to define a centrality measure for vertex sets and to compute for a given number k a vertex set of size k that optimizes this measure. Numerous different group centrality measures have been proposed and investigated, for example degree centrality [22] (which measures how many edges emanate from a group), variants of Katz-centrality [4] (which measure how many short walks visit a group), or closeness centrality [3, 5, 10, 16] (which measures the distance of a group to the rest of the network). We study the problem of computing a group of fixed size k with optimal closeness centrality.

To define closeness centrality, consider an undirected connected graph G and a vertex set $S \subseteq V(G)$. The distance of a vertex u from S, denoted $\mathrm{dist}(u, S)$, is the length of a shortest path that has u as one endpoint and some vertex $v \in S$ as the other endpoint. The *group farness* of S is $\mathrm{c}(S) \coloneqq \sum_{v \in V(G)} \mathrm{dist}(v, S)$. Intuitively, the higher the group farness, the further the vertices of $V \setminus S$ are from S. In applications, one is often interested in maximizing the *closeness centrality* of S, defined as $\bar{\mathrm{c}}(S) \coloneqq \frac{n-|S|}{\mathrm{c}(S)}$. Here, $n \coloneqq |V(G)|$. Clearly, minimizing group farness is equivalent to maximizing the closeness centrality. Here, we use group farness since some of our improvements can be described easier with this notation. Accordingly, we say that a *k-closeness group* for a graph G and some $k \in \mathbb{N}$ is a subset $C \subseteq V(G)$ of size exactly k such that $c(C)$ is minimal among all size-k vertex sets. This leads to the following problem.

Group Closeness Centrality
Input: A graph $G = (V, E)$ and an integer $k \leq n$.
Task: Find a k-closeness group $C \subseteq V$.

Note that the centrality of a k-closeness group is $n - k$ if and only if S is a dominating set, that is, every vertex $v \notin S$ has some neighbor in S. Consequently, Group Closeness Centrality is NP-hard [23] and W[2]-hard with respect to k which suggests that a running time of $f(k) \cdot n^{\mathcal{O}(1)}$ cannot be achieved [19]. Thus, the brute-force $n^{k+\mathcal{O}(1)}$-time algorithm that considers all vertex sets of size k cannot be substantially improved in terms of worst-case running time. Hence, Group Closeness Centrality is computationally challenging even for small values of k.

Known Results. Closeness-related centrality definitions for assessing the centrality of single vertices date back to works of Bavelas [6], Beauchamp [7], and Sabidussi [31]. The extension of centrality notions from single vertices or edges to vertex sets was proposed by

*Some of the results of this work are also contained in the first author's Bachelor thesis [32].

†Philipps-Universität Marburg, Germany, staus@students.uni-marburg.de, {morawietz, fsommer}@informatik.uni-marburg.de

‡Friedrich Schiller University Jena, Germany, c.komusiewicz@uni-jena.de, Work done while affiliated with Philipps-Universität Marburg.

§Supported by the DFG, project OPERAH (KO 3669/5-1).

¶Supported by the DFG, project EAGR (KO 3669/6-1).

Everett and Borgatti [22] who also defined the group closeness measure.

Group Closeness Centrality was considered by Chen et al. [16] who observed NP-hardness and proposed a greedy heuristic. Subsequently, Bergamini et al. [10] developed an improved variant, Greedy++, which provided better solutions and was able to quickly compute solutions on large real-world networks. Later, local-search-based heuristics were developed which find even better solutions at the cost of an increased running time [5]. To our knowledge, the only existing exact algorithm is an ILP formulation which was used to assess the quality of heuristic solutions [10].

Further works are concerned with identifying the vertices with the top-k centralities [8, 27], or with approximating the best centrality within a factor of $(1+\varepsilon)$ via random sampling [21]. The latter approach was also evaluated experimentally [13, 18, 15]. Finally, there are also approaches for estimating the centrality of groups on massive graphs [34].

Group Closeness Centrality and its variants have also found application. For example, Adebayo and Sun [1] modified the closeness centrality measure to identify vertices which are liable to voltage instability in electric power networks; Chea and Livesay [14] used closeness centrality in the analysis of protein structures.

Our Results. We first examine whether certain exact algorithms for Dominating Set on sparse graphs can be translated to Group Closeness Centrality. For example, it is easy to see that Dominating Set can be solved in $f(\Delta + k) \cdot n^{\mathcal{O}(1)}$ time where Δ is the maximum degree of the input graph: when the graph has more than $\Delta \cdot k + k$ vertices, then the instance is a no-instance. This led to a fruitful line of research where instead of Δ, smaller parameters, for example the degeneracy of the input graph, were considered [2, 28, 26]. We show that such a line of attacking the problem difficulty is presumably fruitless for Group Closeness Centrality by showing that the problem is W[2]-hard even on graphs with constant maximum degree. The reduction also shows that if the Exponential Time Hypothesis [24] (ETH) holds, then a running time of $n^{\Theta(k)}$ is necessary for solving Group Closeness Centrality on graphs with constant maximum degree.

We then propose two new exact approaches for solving Group Closeness Centrality. The first is based on augmenting the brute-force algorithm that enumerates all size-k subsets of $V(G)$ by problem-specific pruning rules. The enumeration is organized in a search tree that enumerates the size-k vertex sets by gradually extending smaller vertex sets by adding one vertex at a time. The rules aim at either discarding a current vertex set and all of its extensions or at shrinking the set of candidates to add for a current vertex set. All of the rules make use of a lower bound for the achievable centrality of size-k sets containing the current smaller vertex set S: If the lower bound is higher than the current best solution, then S can be discarded. Moreover, if the lower bound is only slightly smaller, then one may discard vertices which give only a small centrality improvement as candidates for addition to S. Moreover, we identify some vertices which need not be considered based on the existence of better vertices which are guaranteed to give a centrality improvement which is at least as large. We show that these rules yield a considerable speed-up over the brute-force algorithm, making it now possible to solve networks with $\approx 2\,500$ vertices for k up to 10 and networks with $\approx 20\,000$ vertices for k up to 5.

The second approach is a new ILP formulation which is guaranteed to have fewer variables and constraints (than the previous ILP) on networks with small diameter. We show that this new ILP formulation is also faster in practice, making it now possible to solve real-world instances with $2\,500$ vertices and $k = 20$. For both algorithms we present an iterative approach that allows to solve the problem without computing the lengths of all shortest paths. The idea is to perform a truncated breadth-first search from each vertex and to only explore further layers when this may lead to solutions with lower group farness. This iterative approach leads to a further speed-up of both algorithms, with the ILP for example now being able to solve instances with $10\,000$ vertices and $k = 15$. The comparison between ILP and search tree algorithm shows that some larger instances with small k cannot be solved by the best ILP but by the search tree algorithm. Hence, in such circumstances the search tree algorithm is preferable to the ILP.

Since our new algorithms managed to solve instances which were previously not solved optimality, we re-examine the quality of previous heuristics for these instances. We find that Greedy++ [10] provides very good solutions and that the local search algorithms [5] are able to improve these solutions slightly.

Due to lack of space, several proofs (marked with $(\star)$) are deferred to a full version of this article.

Preliminaries. We consider undirected connected graphs G and let $V(G)$ denote the *vertex set* of G and $E(G)$ its *edge set*. Let n and m denote the number of vertices and edges in G, respectively. For a pair of vertices u and v, let $\mathrm{dist}(u, v)$ denote the *length of a shortest path between* u and v. Moreover, for a vertex set $S \subseteq V(G)$, let $\mathrm{dist}(u, S) \coloneqq \min_{v \in S} \mathrm{dist}(u, v)$ denote the *length of a shortest path from* u *to some vertex in* S. For a vertex v, let $N(v) \coloneqq \{u \in V(G) \mid \{u, v\} \in E(G)\}$ and $N[v] \coloneqq N(v) \cup \{v\}$ denote the *open*

and *closed neighborhood* of v, respectively. Moreover, define $N[S] \coloneqq \bigcup_{v \in S} N(v) \cup S$. The *eccentricity* $\mathrm{ecc}(v)$ is the largest distance between v and any other vertex in G. The *diameter* of a graph G, denoted by $\mathrm{diam}(G)$, is the largest distance between any pair of vertices in G, that is, $\mathrm{diam}(G) = \max_{v \in V(G)} \mathrm{ecc}(v)$. The *density* of a graph is $\mathrm{dens}(G) \coloneqq m/\binom{n}{2}$.

For $a \in \mathbb{N}_0$ and $b \in \mathbb{N}$, $a < b$, define $[a, b] \coloneqq \{a, \ldots, b\}$ and $[b] \coloneqq \{1, \ldots, b\}$.

2 A Tight Running Time Lower Bound

We first show that even on graphs of constant maximum degree, there is little hope to improve substantially over the worst-case running time of $n^{k+\mathcal{O}(1)}$ of the search tree algorithm presented in Section 3. Hence, on sparse graphs Group Closeness Centrality is considerably harder than Dominating Set which can be solved in $(\Delta+1)^k \cdot n^{\mathcal{O}(1)}$ time. To show the running time lower bound, we present a parameter preserving polynomial-time reduction from an instance $I = (\mathcal{U}, \mathcal{F}, k)$ of Hitting Set to an instance I' of the decision version of Group Closeness Centrality.

> Hitting Set
> **Input:** A universe $\mathcal{U}$, a collection $\mathcal{F}$ of non-empty subsets of $\mathcal{U}$, and an integer $k \leq |\mathcal{U}|$.
> **Question:** Is there a *hitting set* $H \subseteq \mathcal{U}$ of size at most k, that is, a set $H \subseteq \mathcal{U}$ of size at most k such that for each *hyperedge* $F \in \mathcal{F}$, H contains at least one element of F.

Theorem 2.1. ($\star$) *Even on graphs of maximum degree six,* Group Closeness Centrality *is* W[2]*-hard when parameterized by k and cannot be solved in $n^{o(k)}$ time, unless the ETH fails.*

The idea of the construction is as follows: Each hyperedge $F \in \mathcal{F}$ is represented by a vertex v_F. Additionally, for each $i \in [k]$, there will be a vertex set V_i containing a vertex u^i for each element $u \in \mathcal{U}$. We will define an integer Z such that, for each vertex v_F and each vertex u^i, the distance between v_F and u^i is Z if u is contained in the hyperedge F and $Z + 1$, otherwise. To remain a constant maximum degree, this integer Z is very large. The idea is that each k-closeness group S contains for each $i \in [k]$ exactly one vertex from V_i. To ensure this, we will add for each element $u \in \mathcal{U}$ and each $i \in [k]$, a complete binary tree with root u^i of large depth. Finally, we have to ensure that each k-closeness group S corresponds to a hitting set. Hence, we have to ensure that the distance of each vertex v_F to some vertex of S has to be small, that is, $\mathrm{dist}(v_F, S) = Z$. To this end, we will add for each hyperedge $F \in \mathcal{F}$, a complete binary tree with root v_F of large depth.

The constructions yields an instance of Group Closeness Centrality with $\mathcal{O}(|\mathcal{U}|^{21} \cdot k^{21} \cdot |\mathcal{F}|^{10})$ vertices. If Group Closeness Centrality could be solved in $n^{o(k)}$ time on graphs of constant maximum degree, then one could solve Hitting Set in $|I|^{o(k)}$ time. Unless the ETH fails, this is not possible [17, 19].

3 A Search Tree Algorithm

We now describe a search tree algorithm for Group Closeness Centrality which at its heart is based on the naive brute-force algorithm with running time $n^{\Theta(k)}$. In light of the hardness result shown above, this seems unavoidable. We will then describe several speed-ups which allow us to obtain a much better running time performance in practice.

3.1 The Basic Search Algorithm. A straightforward exact algorithm to find a k-closeness group computes for every $S \subseteq V(G)$ of size exactly k the centrality $c(S)$ and returns the set such that $c(S)$ is minimal. More precisely, this algorithm uses a set enumeration tree [30] in which each node T corresponds to a vertex set S_T of size at most k. To enumerate each set exactly once, each node T is equipped with a set R_T of remaining vertices, which may be added to enlarge S_T. In other words, the vertices of $V(G) \setminus R_T$ are forbidden. For a non-root node T, we denote by T^* the *parent* of T and by $v_T \coloneqq S_T \setminus S_{T^*}$ the unique vertex which was added to the vertex set corresponding to T. Similarly, a node T' is a *child* of T if $v_{T'} \coloneqq S_{T'} \setminus S_T$ and $v_{T'} \in R_T$. By $C(T)$ we denote the set of all *children* of T. The vertex set S_T of a node T with *depth* d_T has size exactly d_T. Hence, the root corresponds to the empty set and each leaf, that is, each node T with depth k corresponds to a possible solution, that is, a vertex set of size exactly k. Clearly, if $|R_T| + d_T < k$, then there does not exist any further solution containing S_T and hence the algorithm can return to the parent T^* of T. This search tree contains $\Omega(\binom{n}{k})$ nodes and thus this algorithm is not practical, even for very small values of k. We refer to this algorithm as *Plain*.

We now present several speed-up techniques.

3.2 Lower Bounds based on Centrality-Improvements. Our first speed-ups are based on a measure of how much the centrality of a vertex set Z improves if some vertex v is added to Z. This is captured in the following definition.

Definition 3.1. *Let $Z \subseteq V(G)$ and let $v \in V(G)$. The* centrality-improvement *of v with respect to Z is* $\mathrm{ci}(Z, v) \coloneqq c(Z) - c(Z \cup \{v\})$.

Note that $\mathrm{ci}(Z, v)$ is also known as the *marginal*

gain of v with respect to Z [10]. Observe that if $v \in Z$, then $\mathrm{ci}(Z, v) = 0$. If $Z = \emptyset$, then we set $c(Z) = n^2$. Now, for a node T in the search tree, we define the centrality-improvement of T with respect to its parent T^*, that is, $\mathrm{ci}(T) \coloneqq \mathrm{ci}(S_{T^*}, v_T)$. In other words, $\mathrm{ci}(T)$ is the difference of the centrality of S_{T^*} and the centrality of S_T. The centrality function is supermodular [16, 10] which implies the following.

PROPOSITION 3.1. *Let $S \subseteq V(G)$ and $S' \subseteq S$ be vertex sets. Moreover, let $v \in V(G)$, then* $\mathrm{ci}(S', v) \geq \mathrm{ci}(S, v)$.

At each node T, the algorithm now computes the centrality-improvements of all children $C(T)$ of T. While this computation is time-consuming, it allows us to compute lower bounds to prune the search tree.

Centrality Lower Bound. In the *centrality lower bound* of a node T with respect to a vertex set R and an integer k, we aim to capture the smallest centrality of any vertex set Y of size k such that $S_T \subseteq Y \subseteq R$. Moreover, we make use of the more general case where we also consider subsets of size k' for some $k' < k$. In our algorithm, R will be the set of remaining vertices R_T. Furthermore, by c_{best} we denote the smallest centrality of any vertex set detected by the algorithm so far.

DEFINITION 3.2. *Let T be a node in the search tree, let $k' \leq k$ an integer, and let $R \subseteq V(G)$. Then the* centrality lower bound of T with respect to R and k' *is*

$$\mathrm{clb}(T, R, k') \coloneqq \mathrm{c}(S_T) - \max_{Y \subseteq R : |Y| = k' - |S_T|} \sum_{y \in Y} \mathrm{ci}(S_T, y).$$

The correctness of the centrality lower bound, as stated in the lemma below, follows from Proposition 3.1.

LEMMA 3.1. *Let T be a node in the search tree, let $k' \leq k$ an integer, and let $R \subseteq V(G)$. Then, for each Y with $|Y| = k'$ and $S_T \subseteq Y \subseteq R_T$, we have* $\mathrm{clb}(T, R, k) \leq \mathrm{c}(Y)$.

The algorithm can use the centrality lower bound as follows: If $\mathrm{clb}(T, R_T, k)$ is not smaller than the previous best centrality c_{best}, then one can immediately return to the parent T^* of T. Furthermore, when the algorithm returns from a child $T' \in C(T)$ to T, then the algorithm computes $\mathrm{clb}(T, R_T \setminus \{v_{T'}\}, k)$ since all solutions containing $S_T \cup \{v_{T'}\}$ have been considered.

A naive computation of the centrality lower bound requires finding the top k elements which takes at least linear time. To avoid this, we use the dynamic sorting of the centrality improvements.

Dynamic Vertex Ordering. In the brute-force algorithm from Section 3.1, the order in which the remaining vertices of R_T are added to S_T at a node T is irrelevant. With the centrality improvements of all vertices in R_T at hand, we now sort the vertices in R_T descendingly according to $\mathrm{ci}(S(T), v)$ each time the algorithm explores some node T. As a result, the algorithm first explores a child T' such that $\mathrm{ci}(T')$ is maximal among all children $C(T)$ of T. For the computation of the centrality lower bound this has the advantage that it is sufficient to compute the sum of the first $k - |S_T|$ vertices in this order. Even better, this sum can be updated in constant time: Store the sum z of the $k - |S_T|$ largest centrality improvements and each time the algorithm returns from a child $T' \in C(T)$ to T, subtract $\mathrm{ci}(T')$ from z and add the centrality improvement of the next vertex of R_T to z. Hence, the initial computation of the centrality lower bound at T requires $\mathcal{O}(k)$ time and each update needs $\mathcal{O}(1)$ time.

To further prune the search tree, we use two techniques which are based on the centrality improvements and use c_{best}. If c_{best} is too high, our lower bounds do not apply. Thus, a standard approach is to initially use a simple greedy heuristic to compute an initial solution.

A good candidate for this is the greedy algorithm proposed by Chen et al. [16] in which k times a vertex with the currently highest centrality improvement is chosen. Chen et al. [16] claimed that this gives a $(1 - 1/e)$-approximation which was disproved by Bergamini et al. [9]. Because in our algorithm the vertices are ordered descendingly by their centrality improvement, the first subset of size k is precisely the set found by the greedy algorithm of Chen et al [16]. Thus, with the dynamic vertex ordering, our algorithm implicitly uses this greedy algorithm to compute the first solution and hence c_{best} is already initially quite large.

Minimal Centrality Threshold. The *minimal centrality threshold* is a measure on how large the centrality improvement of each vertex v has to be at least so that the centrality of a vertex set of size k containing v and all vertices of S_T, is lower than the currently smallest centrality c_{best}.

DEFINITION 3.3. *Let T be a node in the search tree, and $S \subseteq V$ such that $|S| \geq k - |S_T|$. Then the* minimal centrality threshold *of T with respect to S, k, and c_{best} is* $\mathrm{mc}(T, S, k, c_{\text{best}}) \coloneqq \mathrm{clb}(T, S, k-1) - c_{\text{best}}$.

Next, we verify the intuition discussed above.

LEMMA 3.2. *Let T be a node in the search tree, $S \subseteq V$ such that $|S| \geq k - |S_T|$, and $L \coloneqq \{v \in S \mid \mathrm{ci}(S_T, v) \leq \mathrm{mc}(T, S, k, c_{\text{best}})\}$. Then, for each $R \subseteq S$ of size exactly $k - |S_T|$ such that $R \cap L \neq \emptyset$ we have* $\mathrm{c}(S_T \cup R) \geq c_{\text{best}}$.

Proof. Assume towards a contradiction that there exists a set $R \subseteq S$ with $R \cap L \neq \emptyset$ and $|R| = k - |S_T|$ such

that $c(S_T \cup R) < c_{\text{best}}$. Let $v \in R \cap L$. We obtain that

$$\begin{aligned} c(S_T \cup R) &< c_{\text{best}} \\ &= \text{clb}(T, S, k-1) - \text{mc}(T, S, k, c_{\text{best}}) \\ &\leq \text{clb}(T, S, k-1) - \text{ci}(S_T, v) \end{aligned}$$

From $c(S_T \cup R) = c(S_T \cup R \setminus \{v\}) - \text{ci}(S_T \cup R \setminus \{v\}, v)$ we obtain that

$$\begin{aligned} c(S_T \cup R \setminus \{v\}) < \text{clb}(T, S, k-1) &- \text{ci}(S_T, v) \\ &+ \text{ci}(S_T \cup R \setminus \{v\}, v) \end{aligned}$$

Now, from Proposition 3.1, we obtain $\text{ci}(S_T, v) \geq \text{ci}(S_T \cup R \setminus \{v\}, v)$ and thus

$$c(S_T \cup R \setminus \{v\}) < \text{clb}(T, S, k-1)$$

Since $|R \setminus \{v\}| = k - 1 - |S_T|$ this inequality is a contradiction to Lemma 3.1. □

In other words, each vertex set of size k containing S_T and a vertex v whose centrality improvement with respect to T is at most $\text{mc}(T, S, k, c_{\text{best}})$ can never have a lower centrality than c_{best}. In the algorithm we can use the minimal centrality threshold as follows: All vertices whose centrality improvement is at most $\text{mc}(T, S, k, c_{\text{best}})$ can be removed from R_T. Removing these vertices has the advantage that they can also not be added to the vertex set in any descendant of T. Furthermore, when the algorithm returns from a child $T' \in C(T)$ to T it can recompute $\text{mc}(T, S, k, c_{\text{best}})$ to remove further vertices from R_T.

To evaluate the minimal centrality threshold efficiently, the algorithm can use the sum of the $k - |S_T|$ best centrality improvements which is already known after computing the centrality lower bound (Definition 3.2). Now, to determine the minimal centrality threshold, only the smallest term has to be subtracted from this sum. Thus, the minimal centrality threshold can be computed in constant time, if the centrality lower bound is already computed. Removing the vertices whose centrality improvement is below the minimal centrality threshold can be done in $\mathcal{O}(\ell)$ time where ℓ is the number of vertices whose centrality improvement is too small: Since the centrality improvements are sorted descending, the algorithm first compares $\text{mc}(T, S, k, c_{\text{best}})$ with the worst centrality improvement and removes this vertex if possible. Then, it considers the vertex with the second smallest centrality improvement. This continues until the first vertex with a higher centrality improvement is detected.

Parent Threshold. Next, we extend the minimal centrality threshold to remove some vertices in $R_{T'}$ for some child $T' \in C(T)$. This threshold has the advantage that we can remove vertices without having to recompute their centrality improvements.

DEFINITION 3.4. *Let T be a search tree node, $T' \in C(T)$ a child of T, and $S \subseteq V$ such that $|S| \geq k - |S_T|$. Then the* parent threshold *of T with respect to T', S, k, and c_{best} is* $\text{pt}(T, T', S, k, c_{\text{best}}) := \text{clb}(T, S, k-2) - \text{ci}(T') - c_{\text{best}}$.

The parent threshold is similar to the minimal centrality threshold, the difference are that 1) we use the centrality lower bound of parent node T and not of T' and 2) we use the centrality lower bound of size $k-2$.

LEMMA 3.3. ($\star$) *Let T be a node in the search tree, $T' \in C(T)$ a child of T, $S \subseteq V$ such that $|S| \geq k - |S_T|$, and $L := \{v \in S \mid \text{ci}(S_T, v) \leq \text{pt}(T, T', S, k, c_{\text{best}})\}$. Then, for each $R \subseteq S$ of size $k - d_{T'}$ such that $R \cap L \neq \emptyset$ we have $c(S_{T'} \cup R) \geq c_{\text{best}}$.*

Note that for each vertex which is removed by the parent threshold we save $\Omega(n)$ time. Since the minimal centrality threshold provides a better threshold, we only have to compute the parent threshold once. Similar to the minimal centrality threshold, the parent threshold can also be computed in constant time provided that the centrality lower bound is already computed. In the following, we denote the algorithm using these three lower bounds as *CI*.

3.3 Dominating Vertices. Currently, we only use lower bounds (see Section 3.2) to prune the search tree. In this section, we additionally use neighborhood relations. The pruning is based on the observation for two vertices $u, w \in V(G)$ with $N[u] \subseteq N[w]$: If there exists a k-closeness group containing u, then there exists another one containing w. Here, we extend this observation to scenarios in which we search for a solution that contains a set $S_T \subseteq V(G)$ since this is what we aim to find at node T of the search tree.

DEFINITION 3.5. *Let $u, w \in V(G)$ and let $S \subseteq V(G)$. Then, u is* dominated by w with respect to S *if*

1. $N[u] \setminus N[S] \subseteq N[w] \setminus N[S]$ *or*
2. $N(u) = N(w)$.

In the following, we denote the set of vertices dominated by v with respect to S by $D_S(v)$. The aim of dominating vertices is to identify in the subtree rooted at a node T a smaller subset of S_T which is sufficient to detect the best vertex set containing S_T.

DEFINITION 3.6. *Let $S, R \subseteq V$. A set $X \subseteq R$ is an* (S, R)-sufficient set *if*

1. *each $r \in R$ is dominated by at least one $x \in X$ with respect to S, and*

2. for each $Z \subseteq R$ with Property 1, we have $|Z| \geq |X|$.

THEOREM 3.1. ($\star$) *Let $k \in \mathbb{N}$, $S, R \subseteq V$, $|S| \leq k$, and let X be an (S,R)-sufficient set of size at least $k - |S|$. For each $C \subseteq R \setminus S$ with $|C| = k - |S|$, there is a set $D \subseteq X$ of the same size such that $\mathrm{c}(D \cup S) \leq \mathrm{c}(C \cup S)$.*

At each search tree node T, we thus only need to consider an (S_T, R_T)-sufficient set of size at least k. Preliminary experiments showed that the restriction to these sets at every node of the search tree increases the running time. Thus, we only use dominating vertices in the root, that is, we initially compute a $(\emptyset, V(G))$-sufficient set and restrict our search to this set. The algorithm using the improvements so far and the dominating vertices is denoted as *DV*.

4 A new ILP Formulation

Bergamini et al. [10] presented an ILP formulation for GROUP CLOSENESS CENTRALITY using $n^2 + n$ binary variables and $n^2 + n + 1$ constraints. We denote this ILP formulation as *ILPold*.

A More Compact ILP for Small World Networks. We now present a new ILP formulation with only $n \cdot (\mathrm{diam}(G)+1)$ binary variables and $n \cdot \mathrm{diam}(G) + n+1$ constraints. Since $\mathrm{diam}(G)$ is much smaller than n in most real-world graphs (see Table 1) our ILP formulation is thus much smaller than the one of Bergamini et al. [10]. As we will show, it also performs better in terms of running time.

The idea of the ILP is as follows. For each vertex $v \in V(G)$ and each $i \in [0, \mathrm{diam}(G)]$ we introduce a binary variable $x_{v,i}$. The meaning of this variable is that $x_{v,i} = 1$ if and only if $\mathrm{dist}(v, S) = i$ where S is an optimal solution computed by the ILP. In particular, for every vertex v, we have $x_{v,0} = 1$ if and only if $v \in S$.

The ILP formulation reads as follows:

$$\min \sum_{v \in V(G), i \in [0,\mathrm{diam}(G)]} i \cdot x_{v,i} \quad \text{subject to} \tag{4.1}$$

$$k = \sum_{v \in V(G)} x_{v,0} \tag{4.2}$$

For each $v \in V(G)$ we have:

$$1 = \sum_{i \in [0,\mathrm{diam}(G)]} x_{v,i} \tag{4.3}$$

For each $v \in V(G)$ and each $i \in [0, \mathrm{diam}(G)]$ we have:

$$x_{v,i} \leq \sum_{w \in V(G): \mathrm{dist}(v,w) = i} x_{w,0} \tag{4.4}$$

$$x_{v,i} \in \{0, 1\} \tag{4.5}$$

The objective function (Equation (4.1)) equals the centrality of the chosen vertex set S, assuming that $x_{v,i} = 1$ if and only if $\mathrm{dist}(S, v) = i$. Thus, to show correctness we have to argue that this is the case. Assuming that S consists of all vertices $v \in V(G)$ such that $x_{v,0} = 1$, Equation (4.2) guarantees that S has size k. Now, Equation (4.3) ensures that each vertex contributes exactly one value to the objective function. Furthermore, Equation (4.4) guarantees that this value is not too small. More precisely, $x_{v,i} = 1$ only if there exist at least one vertex $w \in S$ which has distance exactly i to v which implies that $\mathrm{dist}(S, v) \leq i$. Finally, Equation (4.5) ensures that all variables are binary.

One can further reduce the number of variables in this ILP formulation as follows: if a vertex $v \in V(G)$ has distance at most $\mathrm{ecc}(v) < \mathrm{diam}(G)$ to all other vertices of G, we only need the variables $x_{v,i}$ for $i \in [0, \mathrm{ecc}(v)]$ since for larger values of i, the sum in Equation (4.4) is empty which implies that the corresponding variable always equals 0. We refer to this formulation as *ILPnew*.

5 Iterative Approach

5.1 An Iterative ILP Approach. Despite having less variables than *ILPold*, *ILPnew* may still contain some unnecessary variables: If for example G contains a dominating set of size k, then for each vertex $v \in V(G)$, only $x_{v,0}$ or $x_{v,1}$ will be set to 1 and for each $i \geq 2$, each variable $x_{v,i}$ will be set to 0. Ideally, we want to avoid the creation of the variables $x_{v,i}$ for $i \geq 2$ in such cases. More generally, we only want to create the variables which are necessary to obtain an optimal solution.

To exploit this observation we use several iterations of an ILP formulation described below. In one iteration of this formulation, for each vertex $v \in V(G)$ we have variables $x_{v,0}, \ldots, x_{v,d(v)}$. The idea is that if $x_{v,d(v)} = 1$, then our current variables are not sufficient to compute the distance of v and S, where S is the solution consisting of k vertices. More precisely, if $x_{v,j} = 1$ for some $j < d(v)$, then $\mathrm{dist}(v, S) = j$ and otherwise, if $x_{v,d(v)} = 1$, then $\mathrm{dist}(v, S) \geq d(v)$. Now, the ILP formulation reads as follows:

$$\min \sum_{v \in V(G)} \sum_{i \in [0,d(v)]} i \cdot x_{v,i} \quad \text{subject to} \tag{5.6}$$

$$k = \sum_{v \in V(G)} x_{v,0} \tag{5.7}$$

For each $v \in V(G)$ and each $i \in [0, d(v)]$ we have:

$$x_{v,i} \in \{0, 1\} \tag{5.8}$$

For each $v \in V(G)$ and each $i \in [0, d(v) - 1]$ we have:

$$x_{v,i} \leq \sum_{w \in V(G): \mathrm{dist}(v,w)=i} x_{w,0} \tag{5.9}$$

For each $v \in V(G)$ we have:

$$1 = \sum_{i \in [0,d(v)]} x_{v,i} \tag{5.10}$$

Initially, we set $d(v) = 2$ for each $v \in V(G)$; in other words, the first iteration of the ILP formulation corresponds to Dominating Set as described above. To avoid adding all variables we use the following fact.

Theorem 5.1. ($\star$) *An optimal solution for the ILP is also an optimal solution for the* Group Centrality Closeness *instance* (G, k) *if each vertex fulfills the* distance property*, that is, if for each vertex* $v \in V(G)$*, we have* $d(v) = \mathrm{ecc}(v)$ *or* $x_{v,i} = 1$ *for some* $i < d(v)$.

If at least one vertex does not fulfill the distance property, then an optimal solution for this ILP is not necessarily an optimal solution for (G, k). Now, there are several natural ways of selecting vertices for which we add variables. In Section 6.3, we describe the two variants which we implemented and evaluated.

This iterative approach also allows us to avoid the computation of the full distance matrix: Initially, only the neighborhood of each vertex has to be known to check whether G has a dominating set of size k. Then, if $x_{v,2} = 1$ for some vertex $v \in V(G)$, the computation of $N_2(v)$ is necessary. More generally, if for some vertex $v \in V(G)$ the variable $x_{v,i}$ exists for each $i \in [0, d(v)]$ and $x_{v,d(v)} = 1$, then we have to compute the $d(v)$th neighborhood of v. These observations can be exploited as follows: Before the initial ILP we use BFS to compute the neighbors of each vertex in the graph. Thereby, we store the BFS-queues containing the neighbors. Now, if for some vertex $v \in V(G)$ we have $x_{v,2} = 1$ after the initial ILP, we use the existing BFS-queue of v to compute $N_2(v)$ and also save the new BFS-queue of v consisting of $N_2(v)$. More generally, whenever some vertex $v \in V(G)$ violates the distance property, that is, if $x_{v,d(v)} = 1$ and $d(v) < \mathrm{ecc}(v)$, then we use the BFS-queue of v to update this information. Note that in the worst case, we need $\Omega(n^2)$ space since each of the n BFS-queues may have size $\Omega(n)$.

Dominating Vertices. Finally, we also use dominating vertices (see Section 3.3) in all iterative ILP formulations: For each vertex u which is dominated by another vertex v with respect to $\emptyset$, we do *not* create the variable $x_{u,0}$. In other words, we only add the variable $x_{v,0}$ for each vertex v which is contained in an $(\emptyset, V(G))$-sufficient set. The correctness of this approach follows directly from Theorem 3.1.

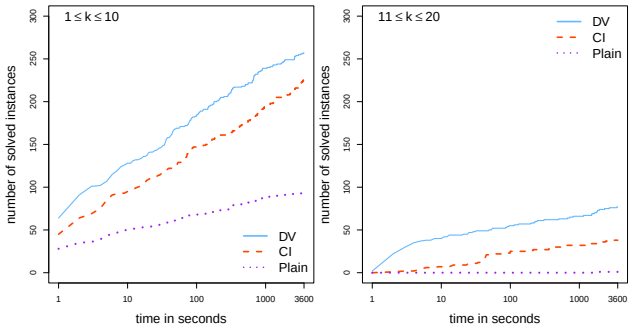

Figure 1: Comparison of *Plain*, *CI* and *DV*.

5.2 An Iterative Search Tree Algorithm. A similar approach to avoid the initial computation of the distance matrix is also possible for our search tree algorithm described in Section 3: Initially, we have $d(v) = 2$ for each vertex $v \in V(G)$. Recall that this means, that we initially search for an dominating set of size k and whenever some vertex $v \in V(G)$ has distance $d(v) < \mathrm{ecc}(v)$ to the solution, we have to compute more distances. Similar to the iterative ILP, we use BFS to compute the neighborhood of each vertex and we store the BFS-queues containing them, for the case that more distances have to be computed. Now, if for some vertex $v \in V(G)$ the current $d(v)$-value was not sufficient, we use the existing BFS-queue of v to compute $N_{d(v)}$ and store this set in the queue. Note that in contrast to the iterative ILP, we here use a $n \times n$ matrix to store the pairwise distances of the vertices. For the search tree, this information is necessary to compute the lower bounds, described in Section 3.2. Thus, we still need $\Theta(n^2)$ space. The main advantage of this approach is to avoid the computation of the distance matrix which requires $\mathcal{O}(nm)$ time by invoking BFS from each vertex.

A Sparse Variant. To avoid having to store the pairwise distances of the vertices, we use another variant: For each vertex $v \in V(G)$ we use an dynamic array A having length $d(v)$. Now, in $A[i]$ we store all vertices with distance i to v. These arrays can be efficiently maintained whenever we increase some $d(v)$-value and use the BFS-queue of v to compute $N_{d(v)}$. By using this approach, we only store the distances which are necessary to find an optimal solution.

6 Experiments

Each experiment was performed on a single thread of an Intel(R) Xeon(R) Silver 4116 CPU with 2.1 GHz, 12 CPUs, 24 threads, and 128 GB RAM running Java openjdk 17.0.4. Our algorithms are implemented in Kotlin, using JGraphT (`https://jgrapht.org`)

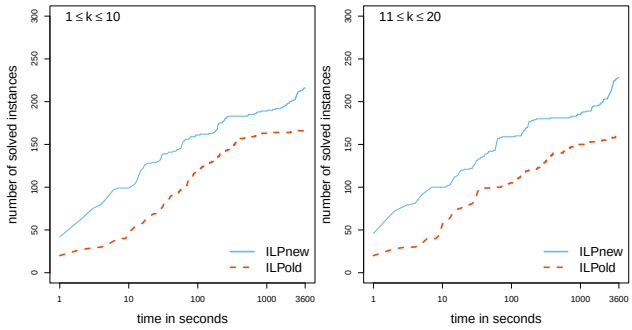

Figure 2: Comparison of *ILPold* and *ILPnew*.

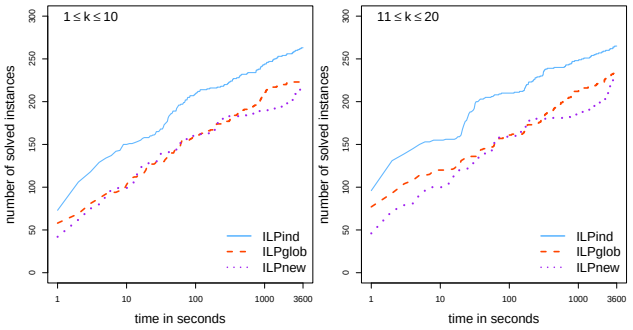

Figure 3: Comparison of *ILPnew*, *ILPind* and *ILPglob*.

as graph data structure. To solve the ILPs (see Sections 4 and 5.1), we use Gurobi 10.0 (`https://www.gurobi.com`). Bergamini et al. [10] used IBM-CPLEX (`https://www.ibm.com/de-de/products/ilog-cplex-optimization-studio`) to solve their ILP formulation. Since all ILP formulations are faster with Gurobi than with IBM-CPLEX we used Gurobi. Our source code and result files can be downloaded at `https://www.uni-marburg.de/en/fb12/research-groups/algorith/closeness-centrality.zip`.

We used 30 social and technical networks from Konect [25] and the Network Repository [29]. The graphs have between 50 and 22 000 vertices; an overview is given in Table 1. For disconnected graphs, we only considered the largest connected component and in Table 1 only the size of the largest connected component is stated. Each variant of our algorithms was tested for each $k \in [20]$. Furthermore, we used a time limit of 60 minutes in our experiments.

The first step in the non-iterative variants (*Plain*, *CI*, *DV*, *ILPold*, and *ILPnew*) is to compute the distance matrix of the graph. This is a limitation of these variants as it leads to memory errors for graphs with more than 100 000 vertices. The distance matrix computation is done rather naively, via breadth-first search from each vertex. For all graphs shown in Table 1, the running time for this step is less than 10 minutes and thus probably not crucial for determining whether or not the instance is solved within the time limit.

6.1 Search Tree Variants. As shown in Figure 1, the lower bound and the lower-bound based thresholds used in algorithm *CI* give a tremendous speed-up over the brute-force algorithm *Plain*. For small k, the number of instances solved within the time limit increases from roughly 25 % to roughly 75 % and for large k, now more than 10 % of the instances can be solved, compared

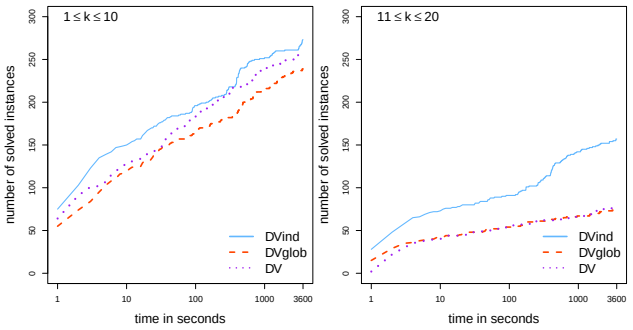

Figure 4: Comparison of *DV*, *DVind* and *DVglob*.

to almost no instances for *Plain*. Roughly 30 % of the instances with small k were solved within 100 seconds.

Figure 1 also shows that using dominating vertices in algorithm *DV* on top of the improvements from algorithm *CI* gives another large speed-up. For small k, the number of instances solved within the time limit increases from roughly 75 % to roughly 85 % and for large k, the number of solved instances roughly doubles so that *DV* solves more than 20 % of the instances.

6.2 Comparison with the Previous ILP. As shown in Figure 2, *ILPnew* is roughly 10 times faster than *ILPold* [10] for small and large k. For small k, *ILPnew* solves 75 % of the instances compared to roughly 55 % for *ILPold*; for large k, *ILPnew* solves 75 % compared to roughly 50 % for *ILPold*.

6.3 Iterative Algorithms. First, we describe the two variants on how new variables are added to the ILP formulation described in Section 5.1: In the first variant, denoted as *ILPglob*, we add for each vertex $v \in V(G)$ such that $d(v) < \mathrm{ecc}(v)$ a new variable $x_{v,d(v)+1}$ to the ILP. In the second variant, denoted as *ILPind*, we

Table 1: Overview of the graphs used in our experiments. The last seven columns show the total number of instances for which the corresponding algorithm finds a solution within the time limit of 60 minutes. A bold number indicates that no other algorithm solved more instances within the time limit; a value of N/A means that for every k no solution was found within the time limit. For the search tree variants (*Plain*, *DV*, *DVind*, *DVindSp*) the number of solved instances equals the largest integer k for which a solution was found.

graph name	n	m	diam	dens	*Plain*	*DV*	*DVind*	*DVindSp*	*ILPold*	*ILPnew*	*ILPind*
contiguous-usa	49	107	11	0.091	11	**20**	**20**	**20**	**20**	**20**	**20**
brain_1	65	730	3	0.351	9	**20**	**20**	**20**	**20**	**20**	**20**
arenas-jazz	198	2 742	6	0.141	5	16	16	16	**20**	**20**	**20**
ca-netscience	379	914	17	0.013	4	**20**	**20**	**20**	**20**	**20**	**20**
robot24c1_mat5	404	14 261	3	0.175	4	9	8	8	**20**	**20**	**20**
reptilia-tortoise	496	984	21	0.001	4	11	**20**	**20**	**20**	**20**	**20**
econ-beause	507	39 428	3	0.307	4	**20**	**20**	**20**	**20**	**20**	**20**
bio-diseasome	516	1 188	15	0.009	4	18	**20**	**20**	**20**	**20**	**20**
soc-wiki-Vote	889	2 914	13	0.007	3	11	**20**	**20**	**20**	**20**	**20**
ca-CSphd	1 025	1 043	28	0.002	3	9	**20**	**20**	**20**	**20**	**20**
arenas-email	1 133	5 451	8	0.009	3	10	10	8	6	**20**	**20**
econ-mahindas	1 258	7 513	8	0.010	3	9	12	13	**20**	**20**	**20**
bio-yeast	1 458	1 948	19	0.002	3	11	**20**	19	**20**	**20**	**20**
comsol	1 500	48 119	12	0.043	3	15	16	16	14	**20**	**20**
medulla_1	1 770	8 905	6	0.006	3	**20**	**20**	**20**	**20**	**20**	**20**
heart2	2 339	340 229	6	0.124	2	13	13	15	17	**20**	**20**
econ-orani678	2 529	86 768	5	0.027	2	11	13	14	**20**	**20**	**20**
inf-openflights	2 905	15 645	14	0.004	2	11	13	7	10	**20**	**20**
ca-GrQc	4 158	13 422	17	0.002	2	8	11	8	N/A	13	**20**
inf-power	4 941	6 594	46	0.001	2	4	15	13	N/A	**18**	17
ca-Erdos992	4 991	7 428	14	0.001	2	9	13	11	N/A	15	**20**
soc-advogato	5 054	39 374	9	0.003	2	10	15	16	1	**20**	**20**
bio-dmela	7 393	25 569	11	0.001	2	7	8	4	N/A	N/A	**15**
ia-escorts-dynamic	10 106	39 016	8	0.001	2	6	7	4	N/A	1	**19**
ca-HepPh	11 204	117 619	13	0.002	2	6	**7**	4	N/A	N/A	5
soc-anybeat	12 645	49 132	9	0.001	2	9	**20**	**20**	N/A	12	**20**
econ-poli-large	15 575	17 468	15	< 0.001	2	9	**20**	19	N/A	5	**20**
ca-AstroPh	17 903	196 972	14	0.001	2	**5**	**5**	1	N/A	N/A	N/A
ca-CondMat	21 363	91 286	13	< 0.001	1	6	**7**	4	N/A	N/A	N/A
ca-cit-HepTh	22 271	2 444 642	7	0.002	1	1	1	5	N/A	N/A	**12**

add for each vertex $v \in V(G)$ such that v violates the distance property a new variable $x_{v,d(v)+1}$ to the ILP. Intuitively, for *ILPglob* we would expect fewer iterations whereas for *ILPind*, we would expect that the solved ILPs contain fewer variables.

We use similar variants for the iterative search tree algorithms described in Section 5.2: In *DVglob* we increase the $d(v)$ value for each vertex $v \in V(G)$ with $d(v) < \text{ecc}(v)$ and in *DVind* we increase the $d(v)$ value of each vertex $v \in V(G)$ violating the distance property. We use similar variants, denoted by *DVglobSp* and *DVindSP*, for the sparse variant of the search tree algorithm.

As shown in Figure 3, *ILPglob* performs overall similar to *ILPnew*; it only performs slightly better for the more difficult instances but the total number of solved instances is roughly the same. In contrast, *ILPind* is much faster: it is roughly 10 times faster than *ILPnew* for the more difficult instances. The speed-up can also be observed in the number of solved instances: for both small and large k, *ILPind* solved roughly 85 % of the instances compared to the 75 % of *ILPnew*.

As shown in Figure 4, *DVglob* performs slightly worse than *DV* for small k while it performs very similar to *DV* for large k. In contrast, *DVind* improves upon *DV*: for small k it performs slightly better than *DV* but for large k it is more than a 100-times faster than *DV* and solves overall twice as many instances.

Since the number of iterations for *ILPglob* is upper-bounded by the diameter of the graph, it is illustrative to put the number of iterations in relation to diam. On average, *ILPglob* needed $0.45 \cdot \text{diam}$ iterations for small k and $0.35 \cdot \text{diam}$ iterations for large k. The minimum number of iterations was $0.17 \cdot \text{diam}$ for small k and $0.09 \cdot \text{diam}$ for large k; the maximum was $0.83 \cdot \text{diam}$ iterations for small k and $0.67 \cdot \text{diam}$ iterations for large k. In comparison, for *ILPind* the average number of iterations is $0.48 \cdot \text{diam}$ for small k and $0.40 \cdot \text{diam}$ iterations for

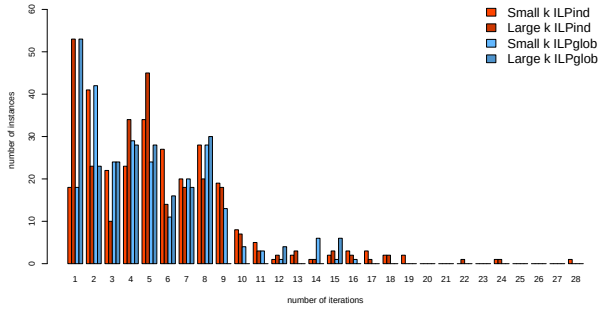

Figure 5: Number of iterations of *ILPind* and *ILPglob*.

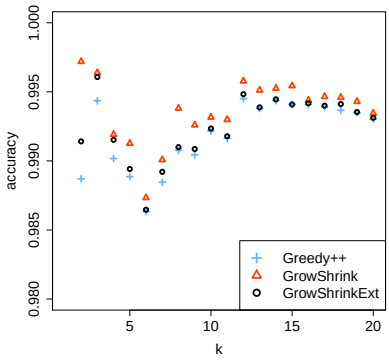

Figure 7: Accuracy of Greedy++ and GrowShrink.

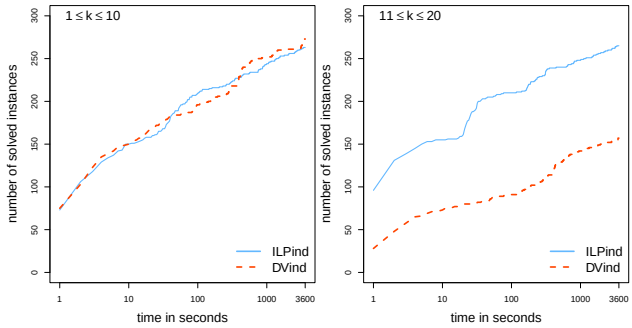

Figure 6: Comparison of *ILPind* and *DVind*.

large k. The minimum number was $0.17 \cdot \text{diam}$ for small k and $0.09 \cdot \text{diam}$ for large k; the maximum was $1.17 \cdot \text{diam}$ for small k and $0.67 \cdot \text{diam}$ for large k.

Both ILP formulations and iterative search tree algorithms need fewer iterations for larger k. This may be viewed as desirable since instances with large k are harder for the search tree algorithm. *ILPglob* and *DVglob* need fewer iterations than *ILPind* and *DVind*. However, the number of iterations for *ILPind* and *DVind* is not *much* higher. This explains the superiority of *ILPind* and *DVind* as their instances are on average smaller than the ones solved by *ILPglob* and *DVglob*.

6.4 Comparison of Search Tree and ILP. Finally we compare the best search tree algorithm *DVind* and the best ILP formulation *ILPind*. As shown in Figure 6, they both perform very similar for small k. However for large k, *ILPind* is much faster than *DVind*. While *DVind* solves roughly 50 % of all instances with large k, *ILPind* can solve roughly 90 % of those instances.

6.5 Accuracy of the Heuristics. With our two exact solvers at hand, we evaluated the accuracy of Greedy++ [10] on all instances for which we found an optimal solution. Here, the *accuracy* is the quotient of the closeness centrality of the solution found by the algorithm and the optimal closeness centrality. We also evaluated the accuracy of GrowShrink [5] and GrowShrinkExt [5], two local search algorithms which used the Greedy++ solution as starting solution. Figure 7 shows the average accuracy of all solved instances. Greedy++ is very close to the optimum (98 %). Furthermore, both local search algorithm are able to improve the solution provided by Greedy++; the margin of improvement is particularly large for small k.

It should be noted that Greedy++ achieved low accuracy on some instances, the worst being comsol.edges with $k = 2$, where only an accuracy of 85 % was achieved. For $k \geq 10$, the worst case is better than for $k < 10$: here, Greedy++ achieved an accuracy of at least 95 % on each instance.

7 Conclusion

We have provided two new algorithms (in several variants) for computing optimal solutions for Group Centrality Closeness. Our new ILP formulation improves substantially over the state-of-the art and our search tree algorithm is a viable alternative for small k. To further accelerate the search tree algorithm, one approach could be to use better starting upper bounds. However, providing the search tree with a starting solution computed by Greedy++ instead of the simple greedy solution and did not give a speed-up. Hence, it seems more promising to find further lower bounds and further cases for domination relations between vertices. One could investigate in particular those instances which need to be solved during the iterative approach. These may contain for example some vertices which have distance 1 or 2 to all other vertices. A further interesting direction is to adapt our algorithms to other group centrality measures like harmonic centrality [3], GED-Walk centrality [4], or betweenness centrality [22].

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.

[2] Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009.

[3] 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 *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX '21)*, pages 154–168. SIAM, 2021.

[4] Eugenio Angriman, Alexander van der Grinten, Aleksandar Bojchevski, Daniel Zügner, Stephan Günnemann, and Henning Meyerhenke. Group centrality maximization for large-scale graphs. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX '20)*, pages 56–69. SIAM, 2020.

[5] Eugenio Angriman, Alexander van der Grinten, and Henning Meyerhenke. Local Search for Group Closeness Maximization on Big Graphs. In *Proceedings of the International Conference on Big Data, (IEEE Big Data '19)*, pages 711–720. IEEE, 2019.

[6] Alex Bavelas. Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, 1950.

[7] Murray A Beauchamp. An improved index of centrality. *Behavioral science*, 10(2):161–163, 1965.

[8] Elisabetta Bergamini, Michele Borassi, Pierluigi Crescenzi, Andrea Marino, and Henning Meyerhenke. Computing top-k Closeness Centrality Faster in Unweighted Graphs. *ACM Transactions on Knowledge Discovery from Data*, 13(5):53:1–53:40, 2019.

[9] Elisabetta Bergamini, Tanya Gonser, and Henning Meyerhenke. Scaling up Group Closeness Maximization. *CoRR*, abs/1710.01144, 2017.

[10] Elisabetta Bergamini, Tanya Gonser, and Henning Meyerhenke. Scaling up Group Closeness Maximization. In *Proceedings of the 20th Workshop on Algorithm Engineering and Experiments (ALENEX '18)*, pages 209–222. SIAM, 2018.

[11] Paolo Boldi and Sebastiano Vigna. Axioms for Centrality. *Internet Mathematics*, 10(3-4):222–262, 2014.

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

[13] Ulrik Brandes and Christian Pich. Centrality Estimation in Large Networks. *International Journal of Bifurcation and Chaos*, 17(7):2303–2318, 2007.

[14] Eric Chea and Dennis R. Livesay. How accurate and statistically robust are catalytic site predictions based on closeness centrality? *BMC Bioinformatics*, 8, 2007.

[15] Shiri Chechik, Edith Cohen, and Haim Kaplan. Average Distance Queries through Weighted Samples in Graphs and Metric Spaces: High Scalability with Tight Statistical Guarantees. In *Proceedings of the 18th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, (APPROX/RANDOM '15)*, volume 40 of *LIPIcs*, pages 659–679. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[16] Chen Chen, Wei Wang, and Xiaoyang Wang. Efficient Maximum Closeness Centrality Group Identification. In *Proceedings of the 27th Australasian Database Conference, (ADC '16)*, volume 9877 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2016.

[17] Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Inf. Comput.*, 201(2):216–231, 2005.

[18] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F. Werneck. Computing Classic Closeness Centrality, at Scale. In *Proceedings of the second ACM conference on Online social networks, (COSN '14)*, pages 37–50. ACM, 2014.

[19] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[20] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[21] David Eppstein and Joseph Wang. Fast Approximation of Centrality. *Journal of Graph Algorithms and Applications*, 8:39–45, 2004.

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

[23] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[24] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[25] Jérôme Kunegis. KONECT: The Koblenz Network Collection. In *Proceedings of the 22nd International World Wide Web Conference, (WWW '13)*, pages 1343–1350. International World Wide Web Conferences Steering Committee / ACM, 2013.

[26] Daniel Lokshtanov and Vaishali Surianarayanan. Dominating set in weakly closed graphs is fixed parameter tractable. In *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '21)*, volume 213 of *LIPIcs*, pages 29:1–29:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[27] Paul W. Olsen, Alan G. Labouseur, and Jeong-Hyon Hwang. Efficient top-k Closeness Centrality Search. In *Proceedings of the 30th International Conference on Data Engineering, (ICDE '14)*, pages 196–207. IEEE

Computer Society, 2014.

[28] Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Trans. Algorithms*, 9(1):11:1–11:23, 2012.

[29] Ryan A. Rossi and Nesreen K. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the 29th Conference on Artificial Intelligence, (AAAI '15)*, pages 4292–4293. AAAI Press, 2015.

[30] Ron Rymon. Search through Systematic Set Enumeration. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR '92)*, pages 539–550. Morgan Kaufmann, 1992.

[31] Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

[32] Luca Pascal Staus. Algorithm engineering für group closeness centrality. Bachelorarbeit, Philipps-Universität Marburg, 2021.

[33] Alexander van der Grinten, Eugenio Angriman, and Henning Meyerhenke. Scaling up network centrality computations - A brief overview. *it – Information Technology*, 62(3-4):189–204, 2020.

[34] Junzhou Zhao, John C. S. Lui, Don Towsley, and Xiaohong Guan. Measuring and Maximizing Group Closeness Centrality over Disk-Resident Graphs. In *Proceedings of the 23rd International World Wide Web Conference, (WWW '14)*, pages 689–694. ACM, 2014.

A Appendix – Proofs

A.1 Proof of Theorem 2.1

Proof. We present a parameterized reduction from Hitting Set which is W[2]-hard when parameterized by k [20].

Hitting Set
Input: A universe $\mathcal{U}$, a collection $\mathcal{F}$ of non-empty subsets of $\mathcal{U}$, and an integer $k \leq |\mathcal{U}|$.
Question: Is there a *hitting set* $H \subseteq \mathcal{U}$ of size at most k, that is, a set $H \subseteq \mathcal{U}$ of size at most k such that for each *hyperedge* $F \in \mathcal{F}$, H contains at least one element of F.

Unless the ETH fails, Dominating Set cannot be solved $n^{o(k)}$ time [17, 19]. By the standard reduction from Dominating Set to Hitting Set this implies that Hitting Set cannot be solves in $|I|^{o(k)}$ time, unless the ETH fails.

We reduce to the decision version of Group Closeness Centrality where we ask for an integer t, if the group farness of any k-closeness group in G is at most t. The hardness and the running-time lower bound then directly extend to Group Closeness Centrality since computing the group farness of a k-closeness group can be done in polynomial time.

Let $I := (\mathcal{U}, \mathcal{F}, k)$ be an instance of Hitting Set. We describe how to obtain an instance $I' := (G = (V, E), k)$ of Group Closeness Centrality in polynomial time and an in integer t, such that there is hitting of size at most k for I if and only if the group farness of each k-closeness group S in G is at most t, that is, the sum of distances of all vertices of V to S is at least t. Figure 8 shows the construction of G.

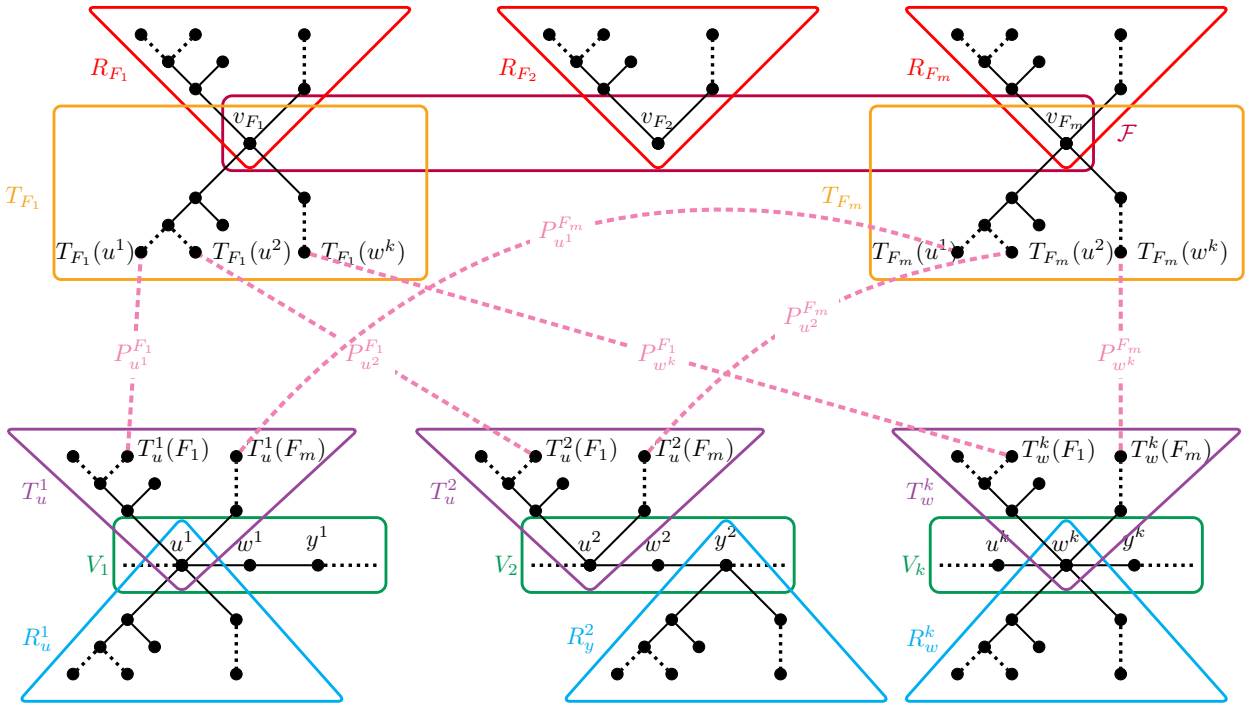

Figure 8: An illustration of the construction of the proof of Theorem 2.1. Only some of the binary trees of the construction are depicted.

Let $n := |\mathcal{U}|$ and let $m := |\mathcal{F}|$ and assume without loss of generality that n is odd. We initialize G as the empty graph. For each $i \in [k]$ and each element $u \in \mathcal{U}$, we add a binary tree T^i_u with m leaves and root u^i to G, such that the leaves of T^i_u are bijectively labeled with the hyperedges of $\mathcal{F}$ and that u^i has distance $\lceil \log(m) \rceil$ to each leaf of T^i_u. For each $i \in [k]$, let $V_i := \{u^i \mid u \in \mathcal{U}\}$. For each $i \in [k]$, we add a minimal number of edges to G such that $G[V_i]$ is an arbitrary cycle of length n. For each hyperedge $F \in \mathcal{F}$,we add a binary tree T_F with $n \cdot k$ leaves and root v_F to G, such that the leaves of T_F are bijectively labeled with the vertices $\{u^i \mid u \in V, i \in [k]\}$

and that v_F has distance $\lceil \log(k \cdot n) \rceil$ to each leaf of T_F. Next, for each element $u \in \mathcal{U}$, each $i \in [k]$, and each hyperedge $F \in \mathcal{F}$, let $T_u^i(F)$ be the leaf of T_u^i labeled with F and let $T_F(u^i)$ be the leaf of T_F labeled with u^i. If u is contained in the hyperedge F, we add a path $P_{u^i}^F$ of length $n^4 \cdot k^4 \cdot m$ between $T_u^i(F)$ and $T_F(u^i)$ to G. Otherwise, if u is not contained in the hyperedge F, we add a path $P_{u^i}^F$ of length $n^4 \cdot k^4 \cdot m + 1$ between $T_u^i(F)$ and $T_F(u^i)$ to G.

Note that v_F and u^i have distance $Z \coloneqq \lceil \log(m) \rceil + n^4 \cdot k^4 \cdot m + \lceil \log(k \cdot n) \rceil$ if u is contained in F and distance $Z + 1$, otherwise. Let V' be the set of all vertices currently in G. Note that each tree T_u^i contains at most $4 \cdot m$ vertices, each tree T_F contains at most $4 \cdot n \cdot k$ vertices and each path $P_{u^i}^F$ contains at most $n^4 \cdot k^4 \cdot m + 1$ vertices. Hence, $|V'| \leq 8 \cdot n \cdot k \cdot m + n \cdot k \cdot m \cdot (n^4 \cdot k^4 \cdot m + 1) \in \mathcal{O}(n^5 \cdot k^5 \cdot m^2)$.

We set $X \coloneqq \lceil \log(|V'|^2) \rceil + 1$. Hence $2^X \in \mathcal{O}(n^{10} \cdot k^{10} \cdot m^4)$. For each hyperedge $F \in \mathcal{F}$, we extend G by a complete binary tree R_F of depth X with root v_F. Each of these trees is a *$R_{\mathcal{F}}$-trees*. Let V'' be the set of all vertices currently in G. Hence, $|V''| \leq |V'| + m \cdot 2^{X+1} \in \mathcal{O}(n^{10} \cdot k^{10} \cdot m^5)$.

We set $Y \coloneqq \lceil \log(|V''|^2) \rceil + 1$. Hence $2^Y \in \mathcal{O}(n^{20} \cdot k^{20} \cdot m^{10})$. For each $i \in [k]$ and each element $u \in \mathcal{U}$, we extend G by a complete binary tree R_u^i of depth Y with root u^i. Each of these trees is a *$R_{\mathcal{U}}$-trees*. Hence, $|V| \leq |V''| + n \cdot k \cdot 2^{Y+1} \in \mathcal{O}(n^{21} \cdot k^{21} \cdot m^{10})$. Finally, we set

$$t \coloneqq |V'|^2 + m \cdot (2^{X+1} \cdot X - 2^{X+1} + 2 + Z \cdot (2^{X+1} - 1)) + k \cdot (n \cdot (2^{Y+1} \cdot Y - 2^{Y+1} + 2) + 2 \cdot \binom{\lfloor n/2 \rfloor}{2} \cdot (2^{Y+1} - 1))$$

and show that there is hitting of size at most k for I if and only if the group farness of each k-closeness group in G is at most t.

Note that for each $i \in [k]$, each vertex $u^i \in V_i$ has two neighbors in $G[V_i]$ and is the root of both binary trees T_u^i and R_u^i. Hence, u^i has degree six. Moreover, for each hyperedge $F \in \mathcal{F}$, each vertex v_F is the root of both binary trees T_F and R_F. Hence, v_F has degree four. All remaining vertices of V have degree at most three. Hence, G has a maximum degree of six.

Before we show the correctness of the reduction, we first make some observations about distances of vertices in the graph G.

Observation A.1. *a) The subgraph $G[V']$ is connected and for each vertex set S containing at least one vertex of V', $\sum_{v \in V'} \operatorname{dist}(v, S) \leq |V'|^2 < 2^X$. b) The subgraph $G[V'']$ is connected and for each vertex set S containing at least one vertex of V'', $\sum_{v \in V''} \operatorname{dist}(v, S) \leq |V''|^2 < 2^Y$.*

Hence, if a vertex set S contains at least one vertex of V_i for some $i \in [k]$, then $\sum_{v \in V'} \operatorname{dist}(v, S) \leq |V'|^2$ and $\sum_{v \in V''} \operatorname{dist}(v, S) \leq |V''|^2$.

Next, we bound the distances of vertices in $R_{\mathcal{U}}$-trees and $R_{\mathcal{F}}$-trees to vertex sets S that contain no vertices of these trees.

Observation A.2. *Let S be a set of vertices, let u be an element of $\mathcal{U}$, let $i \in [k]$, and let F be a hyperedge of $\mathcal{F}$. Moreover, let $d_u \coloneqq \operatorname{dist}(u^i, S)$ and let $d_F \coloneqq \operatorname{dist}(v_F, S)$. a) If $S \setminus \{u^i\}$ contains no vertex of R_u^i, then $\sum_{w \in V(R_u^i)} \operatorname{dist}(w, S) = (2^{Y+1} \cdot Y - 2^{Y+1} + 2) + d_u \cdot (2^{Y+1} - 1)$. b) If $S \setminus \{v_F\}$ contains no vertex of R_F, then $\sum_{w \in V(R_F)} \operatorname{dist}(w, S) = (2^{X+1} \cdot X - 2^{X+1} + 2) + d_F \cdot (2^{X+1} - 1)$.*

($\Rightarrow$) Let $H \coloneqq \{u_1, \ldots, u_k\}$ be a hitting set of size k for I. We set $S \coloneqq \{u_i^i \in V_i \mid i \in [k]\}$ and show that $c(S) \leq t$. Note that S contains a vertex of V'. Hence, since $G[V']$ is connected, $\sum_{v \in V'} \operatorname{dist}(v, S) \leq |V'|^2$.

Since H is a hitting set for I, for each hyperedge $F \in \mathcal{F}$, there is some $u_i \in H$ such that u_i is contained in F. Hence, by construction, for each hyperedge $F \in \mathcal{F}$, $\operatorname{dist}(v_F, S) = Z$ since u_i^i is contained in S. Consequently, due to Observation A.2, the distances of vertices of $R_{\mathcal{F}}$-trees to S is

$$\begin{aligned} \sum_{v \in V'' \setminus V'} \operatorname{dist}(v, S) &\leq \sum_{F \in \mathcal{F}} \sum_{w \in V(R_F)} \operatorname{dist}(w, S) \\ &\leq m \cdot (2^{X+1} \cdot X - 2^{X+1} + 2 + Z \cdot (2^{X+1} - 1)). \end{aligned}$$

Finally, since for each $i \in [k]$, S contains exactly one vertex of V_i and $G[V_i]$ is a cycle of length n, where n is odd, for each $j \in [1, \lfloor n/2 \rfloor]$, there are two vertices of V_i of distance j with u_i^i in G. Hence, due to Observation A.2,

the distances of vertices of $R_{\mathcal{U}}$-trees to S is

$$\begin{aligned}\sum_{v\in V\setminus V''}\operatorname{dist}(v,S) &\leq \sum_{i=1}^{k}\sum_{u^i\in V_i}\sum_{w\in V(R_u^i)}\operatorname{dist}(w,S)\\ &\leq k\cdot(n\cdot(2^{Y+1}\cdot Y-2^{Y+1}+2)+2\cdot\sum_{j=1}^{\lfloor n/2\rfloor}j\cdot(2^{Y+1}-1))\\ &= k\cdot(n\cdot(2^{Y+1}\cdot Y-2^{Y+1}+2)+2\cdot\binom{\lfloor n/2\rfloor}{2}\cdot(2^{Y+1}-1)).\end{aligned}$$

As a consequence,

$$\sum_{v\in V}\operatorname{dist}(v,S)=\sum_{v\in V'}\operatorname{dist}(v,S)+\sum_{v\in V''\setminus V'}\operatorname{dist}(v,S)+\sum_{v\in V\setminus V''}\operatorname{dist}(v,S)\leq t.$$

($\Leftarrow$) Let S be a k-closeness group of G with $c(S)\leq t$. We show that there is a hitting set of size at most k for I.

Fist, we show that for each $i\in[k]$, $\operatorname{dist}(u^i,S)\leq n^4\cdot k^4\cdot m$ for each vertex $u^i\in V_i$. Assume towards a contradiction, that there exists some $i\in[k]$ such that $\operatorname{dist}(u^i,S)>n^4\cdot k^4\cdot m$ for some vertex $u^i\in V_i$. Recall that 2^Y is upper-bounded by some polynomial function of $\frac{n^4\cdot k^4\cdot m}{n^3\cdot k^3}=n\cdot k\cdot m$. Hence, if $n\cdot k\cdot m$ is sufficiently large, $Y\cdot n^3\cdot k^3\leq n^4\cdot k^4\cdot m$. Since each vertex of R_u^i has distance at most Y to u^i, S contains no vertex of R_u^i. Hence, due to Observation A.2 and since $2^{Y+1}+2>Y$,

$$\begin{aligned}\sum_{v\in V}\operatorname{dist}(v,S) &\geq \sum_{w\in V(R_u^i)}\operatorname{dist}(w,S)\\ &\geq (2^{Y+1}\cdot Y-2^{Y+1}+2)+(n^4\cdot k^4\cdot m)\cdot(2^{Y+1}-1)\\ &\geq (Y\cdot n^3\cdot k^3)\cdot(2^{Y+1}-1)\\ &\geq (Y\cdot n\cdot k+n^2\cdot k^2)\cdot(2^{Y+1}-1)\\ &= n\cdot k\cdot(2^{Y+1}\cdot Y-Y)+n^2\cdot k^2\cdot(2^{Y+1}-1)\\ &\geq k\cdot(n\cdot(2^{Y+1}\cdot Y-2^{Y+1}+2)+n^2\cdot(2^{Y+1}-1))+(2^{Y+1}-1)\\ &\geq t+(2^{Y+1}-1)-(|V'|^2+m\cdot(2^{X+1}\cdot X-2^{X+1}+2+Z\cdot(2^{X+1}-1))).\end{aligned}$$

Since $2^Y>|V''|^2$ and $|V''|\geq|V'|+m\cdot 2^{X+1}$, we get $2^{Y+1}-1>|V'|^2+m\cdot(2^{X+1}\cdot X-2^{X+1}+2+Z\cdot(2^{X+1}-1))$. Hence $\sum_{v\in V}\operatorname{dist}(v,S)\geq\sum_{w\in V(R_u^i)}\operatorname{dist}(w,S)>t$, a contradiction.

Hence, for each $i\in[k]$, for each vertex $u^i\in V_i$, $\operatorname{dist}(u^i,S)\leq n^4\cdot k^4\cdot m$. Note that for each vertex $u^i\in V_i$ and each vertex $w^j\in V_j$ with $i\neq j$, each shortest path between u^i and w^j contains at least one vertex of the tree T_F for some hyperedge F. Since the distance between u^i (respectively w^j) and any vertex T_F is at least the length of $P_{u^i}^F$ (respectively $P_{w^j}^F$). Consequently, u^i and w^j have distance more than $2\cdot n^4\cdot k^4\cdot m$. Hence, for each $i\in[k]$, there is a unique vertex $s_i\in S$ with $\operatorname{dist}(s_i,u^i)\leq n^4\cdot k^4\cdot m$ for each vertex $u^i\in V_i$. We show that for each $i\in[k]$, the vertex s_i is from V_i. Assume towards a contradiction that there is some $i\in[k]$ such that s_i is not a vertex of V_i. Let u^i be the vertex of V_i of smallest distance to s_i. Hence, s_i is either a vertex of T_u^i, a vertex of R_u^i, or a vertex of the path between T_u^i and T_F for some hyperedge $F\in\mathcal{F}$. In all three cases, $\operatorname{dist}(S,w^i)=\operatorname{dist}(s_i,w^i)=\operatorname{dist}(s_i,u^i)+\operatorname{dist}(u^i,w^i)$ for each element $w\in\mathcal{U}$. Let $S'\coloneqq(S\setminus\{s_i\})\cup\{u^i\}$. We show that $c(S')<c(S)$.

Note that for each vertex $u'\in V(R_u^i)$, $\operatorname{dist}(S',u')\leq\operatorname{dist}(S,u')-\operatorname{dist}(s^i,u^i)$, and that for each vertex $w^i\in V_i\setminus\{u_i\}$ and each vertex $y\in V(R_w^i)$, $\operatorname{dist}(S',y)=\operatorname{dist}(S,y)-\operatorname{dist}(s^i,u^i)$. Since $\operatorname{dist}(s^i,u^i)>0$ and $n\geq 3$, this

implies that

$$\begin{aligned}
&\sum_{w^i \in V_i} \sum_{w \in V(R^i_w)} \operatorname{dist}(w, S') \\
&\quad \leq \left(\sum_{w^i \in V_i} \sum_{w \in V(R^i_w)} \operatorname{dist}(w, S) \right) + 2^{Y+1} - 1 - (n-1) \cdot (2^{Y+1} - 1) \\
&\quad \leq -2^{Y+1} + 1 + \sum_{w^i \in V_i} \sum_{w \in V(R^i_w)} \operatorname{dist}(w, S) \\
&\quad < -|V''|^2 + \sum_{w^i \in V_i} \sum_{w \in V(R^i_w)} \operatorname{dist}(w, S).
\end{aligned}$$

Moreover, for each $j \in [k] \setminus \{i\}$, for each vertex $v^j \in V_j$, and for each vertex $w \in V(R^j_v)$, $\operatorname{dist}(w, S) \geq \operatorname{dist}(w, S')$ since $\operatorname{dist}(w, s^j) < \operatorname{dist}(w, s^i)$. Finally, since $G[V'']$ is connected and u^i is an element of V'', $\sum_{v \in V''} \operatorname{dist}(v, u^i) \leq |V''|^2$. Hence,

$$\begin{aligned}
c(S') &= \sum_{v \in V} \operatorname{dist}(v, S') \\
&= \sum_{v \in V \setminus V''} \operatorname{dist}(v, S') + \sum_{v \in V''} \operatorname{dist}(v, S') \\
&< -|V''|^2 + \sum_{v \in V \setminus V''} \operatorname{dist}(v, S) + \sum_{v \in V''} \operatorname{dist}(v, S') \leq \sum_{v \in V} \operatorname{dist}(v, S) = c(S).
\end{aligned}$$

This contradicts the fact that S is a k-closeness group. Hence, for each $i \in [k]$, S contains exactly one vertex of V_i. Hence, since for each $i \in [k]$, $G[V_i]$ is a cycle of length n, Observation A.2 implies

$$\begin{aligned}
&\sum_{i=1}^{k} \sum_{u^i \in V_i} \sum_{w \in V(R^i_u)} \operatorname{dist}(w, S) \\
&\qquad = k \cdot (n \cdot (2^{Y+1} \cdot Y - 2^{Y+1} + 2) + 2 \cdot \sum_{j=1}^{\lfloor n/2 \rfloor} j \cdot (2^{Y+1} - 1)) \\
&\qquad = k \cdot (n \cdot (2^{Y+1} \cdot Y - 2^{Y+1} + 2) + 2 \cdot \binom{\lfloor n/2 \rfloor}{2} \cdot (2^{Y+1} - 1)).
\end{aligned}$$

Let $H \coloneqq \{u \in \mathcal{U} \mid \exists i \in [k] : u^i \in S\}$. We show that H is a hitting set for I. Assume towards a contradiction, that there is a hyperedge $F \in \mathcal{F}$ such that H contains no element of F. Hence, $\operatorname{dist}(v_F, S) = Z + 1$ and by construction, for each hyperedge $F' \in \mathcal{F} \setminus \{F\}$, $\operatorname{dist}(v_{F'}, S) \geq Z$. Since S contains no vertex of $R_{F'}$ for any hyperedge $F' \in \mathcal{F}$, Observation A.2 implies that the distance to all $R_{\mathcal{F}}$-trees is

$$\begin{aligned}
&\sum_{w \in V(R_F)} \operatorname{dist}(w, S) + \sum_{F' \in \mathcal{F} \setminus \{F\}} \sum_{w \in V(R_{F'})} \operatorname{dist}(w, S) \\
&\qquad \geq (2^{X+1} \cdot X - 2^{X+1} + 2) + (Z+1) \cdot (2^{X+1} - 1) \\
&\qquad\quad + (m-1) \cdot ((2^{X+1} \cdot X - 2^{X+1} + 2) + Z \cdot (2^{X+1} - 1)) \\
&\qquad = m \cdot (2^{X+1} \cdot X - 2^{X+1} + 2) + Z \cdot (2^{X+1} - 1)) + 2^{X+1} - 1.
\end{aligned}$$

Since $2^{X+1} - 1 > |V'|^2$, we conclude

$$\begin{aligned}
\sum_{v \in V} \operatorname{dist}(v, S) \geq \sum_{i=1}^{k} \sum_{u \in \mathcal{U}} \sum_{w \in V(R^i_u)} \operatorname{dist}(w, S) + \sum_{F \in \mathcal{F}} \sum_{w \in V(R_F)} \operatorname{dist}(w, S) \\
\geq t - |V'|^2 + 2^{X+1} - 1 > t.
\end{aligned}$$

A contradiction. Hence, S is a hitting set for I. □

A.2 Proof of Lemma 3.3

Proof. Assume towards a contradiction that there exists a set $R \subseteq S$ with $R \cap L \neq \emptyset$ and $|R| = k - d_{T'}$ such that $\mathrm{c}(S_{T'} \cup R) < c_{\text{best}}$. Let $v \in R \cap L$. We obtain that

$$\begin{aligned}\mathrm{c}(S_{T'} \cup R) &< c_{\text{best}} \\ &= \mathrm{clb}(T, S, k-2) - \mathrm{ci}(T') \\ &\quad - \mathrm{pt}(T, T', S, k, c_{\text{best}}) \\ &\leq \mathrm{clb}(T, S, k-2) - \mathrm{ci}(T') - \mathrm{ci}(S_T, v)\end{aligned}$$

From $\mathrm{c}(S_{T'} \cup R) \geq \mathrm{c}(S_T \cup R \setminus \{v\}) - \mathrm{ci}(S_T \cup R \setminus \{v\}, v) - \mathrm{ci}(S_T \cup R \setminus \{v, v_{T'}\}, v_{T'})$ we obtain that

$$\begin{aligned}\mathrm{c}(S_T \cup R \setminus \{v\}) &< \mathrm{clb}(T, S, k-2) - \mathrm{ci}(S_T, v) \\ &\quad + \mathrm{ci}(S_T \cup R \setminus \{v\}, v) \\ &\quad - \mathrm{ci}(T') + \mathrm{ci}(S_T \cup R \setminus \{v, v_{T'}\}, v_{T'})\end{aligned}$$

Now, from Proposition 3.1, we obtain $\mathrm{ci}(S_T, v) \geq \mathrm{ci}(S_T \cup R \setminus \{v\}, v)$ and $\mathrm{ci}(T') = \mathrm{ci}(S_T, v_{T'}) \geq \mathrm{ci}(S_T \cup R \setminus \{v, v_{T'}\}, v_{T'})$. Thus,

$$\mathrm{c}(S_T \cup R \setminus \{v\}) < \mathrm{clb}(T, S, k-2)$$

Since $|R \setminus \{v\}| = k - 2 - |S_T|$ this inequality is a contradiction to Lemma 3.1. ∎

A.3 Proof of Theorem 3.1

Proof. Let $C \subseteq R \setminus S$ with $|C| = |S| - k$. If $C \subseteq X$, then we are done. Otherwise, we replace each vertex of $C \setminus X$ by a vertex in X without increasing the centrality. Let $u \in C \setminus X$ and let $w \in X$ be a vertex dominating u with respect to S.

Case 1: $w \notin C$**.** We replace u by w. Let $A \coloneqq C \cup S$ and $B \coloneqq (A \setminus \{u\}) \cup \{w\}$.

First, consider the case that $u \in D_S(v)$ fulfills the Property 1 of Definition 3.5. Assume that $uw \; in E(G)$. Observe that $\mathrm{dist}(u, A) = 0$, $\mathrm{dist}(u, B) = 1$ since $uw \in E(G)$, $\mathrm{dist}(w, A) = 1$, and $\mathrm{dist}(w, B) = 0$. Furthermore, for each further vertex z we have $\mathrm{dist}(z, B) \leq \mathrm{dist}(z, A)$: If u is the only vertex of A such that $\mathrm{dist}(z, A) = \mathrm{dist}(u, z)$, then u is not contained in $N[S]$ and the shortest path from A to z does not contain a vertex from $N[S]$. Consequently, $\mathrm{dist}(u, z) \geq \mathrm{dist}(w, z)$ since $N[u] \setminus N[S] \subseteq N[w] \setminus N[S]$ and A and B only differ in u and w, respectively. Otherwise, consider the case that $uw \notin E(G)$. Since $N[u] \setminus N[S] \subseteq N[w] \setminus N[S]$, we conclude that $u \in N[s]$. Thus, $\mathrm{dist}(u, A) = 0$, $\mathrm{dist}(u, B) = 1$, $\mathrm{dist}(w, B) = 0$, and $\mathrm{dist}(w, A) \geq 1$. For each vertex in $z \in N(u) \setminus S$ we have $\mathrm{dist}(z, u) = 1 = \mathrm{dist}(z, w)$ and for each remaining vertex in the graph the distance to B compared to A does not increase. Thus, $\mathrm{c}(B) \leq \mathrm{c}(A)$.

Second, consider the case that $u \in D_S(v)$ fulfills Property 2 of Definition 3.5. Observe that $\mathrm{dist}(u, A) = 0$ and $\mathrm{dist}(w, B) = 0$. Furthermore, since $N(u) = N(w)$, if $S \cap (N(u) \cap N(w)) \neq \emptyset$, then $\mathrm{dist}(u, B) = 1 = \mathrm{dist}(w, A)$ and otherwise $\mathrm{dist}(u, B) = \mathrm{dist}(w, A)$. Furthermore, for each further vertex $z \in R$ we have $\mathrm{dist}(z, B) \leq \mathrm{dist}(z, A)$, by the same arguments as above. Thus, $\mathrm{c}(B) \leq \mathrm{c}(A)$.

Case 2: $w \in C$**.** Observe that $u \in N[(C \setminus \{u\}) \cup S]$: This is obviously true if $u \in N[S]$. Otherwise, u fulfills Property 1 and thus, $u \in N[u] \setminus N[S] \subset N[w]$. Since $w \in C$, the claim follows. Now, we replace u by some vertex $v \in X \setminus C$. Note that v exists since $|X| \geq k - |S|$ and thus $|X \cap C| < k - |S|$. Now, let $A \coloneqq C \cup S$ and $B \coloneqq (A \setminus \{u\}) \cup \{v\}$. Observe that $\mathrm{dist}(u, A) = 0$, $\mathrm{dist}(u, B) = 1$ since $u \in N[(C \setminus \{u\}) \cup S]$, $\mathrm{dist}(v, A) \geq 1$, and $\mathrm{dist}(v, B) = 0$. Furthermore, for each further vertex z we have $\mathrm{dist}(z, B) \leq \mathrm{dist}(z, A)$, by the same arguments as above. Thus, $\mathrm{c}(B) \leq \mathrm{c}(A)$. ∎

A.4 Proof of Theorem 5.1

Proof. First, observe that if each vertex fulfills the distance property for some optimal solution with objective value c, then the set S containing the vertices with $x_{v,0} = 1$ has centrality exactly c. Now, assume towards a contradiction that there exists an optimal solution Z for the instance (G, k) with centrality $c' < c$. Consider the following solution of the ILP: For each $v \in Z$, set $x_{v,0} = 1$. Observe that for each vertex $v \in V(G) \setminus Z$, $x_{v,i} = 1$ for some $i \leq \mathrm{dist}(v, Z)$. Thus, this solution has an objective value of $c'' \leq c' < c$ in the ILP, a contradiction to the optimality of the solution with objective value c. ∎

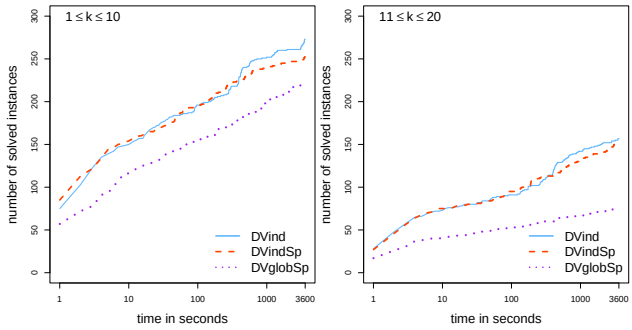

Figure 9: Comparison of *DVind*, *DVindSp* and *DVglobSp*.

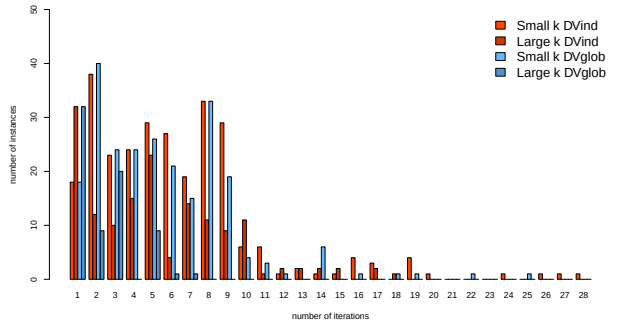

Figure 10: Number of iterations of *DVind* and *DVglob*.

B Appendix – Further Experimental Results

Figure 9 compares the search tree variants *DVindSp* and *DVglobSp* using the sparse matrix representation with *DVind*. Both algorithms perform very similar to their non sparse counterpart which is not surprising since they only differ in the way the distance matrix is represented. *DVindSp* only performs slightly worse than *DVind* on the more difficult instances.

We also evaluated how many iterations *ILPglob* and *ILPind* need to obtain an optimal solution (see Figure 10). As a rule of thumb, a lower number of iterations should give a smaller running time.

As shown in Figure 10, the results are very similar for *DVind* and *DVglob*. Hence, both ILP formulations and both iterative search tree algorithms need fewer iterations for larger k.