Algorithms for Group Closeness Centrality Maximization

Jakob Ternes

December 19, 2025

4262376

Bachelor Thesis

at

Algorithm Engineering Group Heidelberg
Heidelberg University

Supervisor:
Univ.-Prof. PD. Dr. rer. nat. Christian Schulz

Co-Supervisor:
Henning Woydt

Acknowledgments

I am immensely grateful for your guidance, Christian. Your lectures have been a true joy, and combinatorial optimization will forever have a special place in my heart. Henning, your constant feedback and advice have been incredibly valuable. You have made the whole process of writing my thesis feel as smooth as if I had done it before. To my parents, I am truly lucky for you to always support my academic ambitions. Your constant reassurance in what I am passionate about has made it easy for me to navigate my journey. I am thankful to my brother Jona for proofreading this thesis and so much more. I'd also like to thank Lukas, who I am sure will go on to become a superb physicist, for stimulating conversations and his careful proofreading of this thesis.

Hiermit versichere ich, dass ich die von mir am 19.12.2025 als PDF-Datei (mit Standard-PDF-Lesesoftware lesbar und durchsuchbar) eingereichte Bachelor-Arbeit mit dem Titel *Algorithms for Group Closeness Centrality Maximization* betreut von Prof. Dr. Christian Schulz selbstständig und ohne unzulässige fremde Hilfe verfasst habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und sämtliche Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Werken – einschließlich solcher aus digitalen oder KI-basierten Quellen – übernommen wurden, als solche kenntlich gemacht. Ich bestätige, dass die etwaige Nutzung von KI-Tools vorab mit der Betreuerin/dem Betreuer der Abschlussarbeit abgesprochen wurde und dass die besprochenen Regelungen eingehalten wurden. Ich übernehme die volle Verantwortung für die wissenschaftliche Qualität und den Inhalt der vorliegenden Arbeit, die gewählte Methodik und den Erstellungsprozess, sowie die zitierte Literatur. Ich bin damit einverstanden, dass meine Arbeit im Rahmen universitärer Verfahren auf Plagiate sowie auf den möglichen Missbrauch automatisierter Text- und Codegenerierung überprüft wird.

Heidelberg, 19.12.2025

Jakob Ternes

Abstract

Identifying central vertices or groups of central vertices in a graph is an important aspect of network analysis. Group closeness centrality is a centrality measure for groups of vertices which considers the inverse of the total distances from all nodes to the respective closest vertex within a given group. The problem of finding a subset S of size k of the vertices with maximum group closeness centrality is NP-hard. In light of this result, heuristic approaches have been considered. Other authors have proposed exact algorithms to obtain results for situations in which highest solution quality is paramount. We examine state-of-the-art algorithms for both situations and propose new methods to reduce their runtime. In experiments on real-world graphs, our exact algorithm solves more instances overall and achieves a more than 4-fold speedup over the next fastest exact algorithm. Additionally, we generalize the methods of the state-of-the-art exact algorithm and our proposed adaptations to weighted graphs, thereby broadening the applicability of the resulting algorithm. For the approximation case, we consider a local search algorithm for which we show that restricting the search space in a particular way retains the approximation guarantee while giving comprehensive speedups. In addition, we establish the significance of algorithms with quality guarantees by proving that a widely used greedy algorithm may yield arbitrarily poor results.

Abstract (German)

Die Identifikation zentraler Knoten oder Knotengruppen ist ein wichtiger Aspekt der Netzwerkanalyse. Gruppen-Closeness-Zentralität ist ein Zentralitätsmaß für Gruppen von Knoten, welches den Kehrwert der aufsummierten Distanzen aller Knoten zu dem jeweils nächsten Knoten einer gegebenen Gruppe betrachtet. Das Finden einer Untermenge S von Größe k der Knoten mit maximaler Gruppen-Closeness-Zentralität ist NP-schwer. Angesichts dieses Resultats wurden heuristische Ansätze in Erwägung gezogen. Andere Autoren schlugen exakte Algorithmen für Situationen vor, in denen höchste Qualität von übergeordneter Bedeutung ist. Wir untersuchen für beide Situationen Algorithmen, die dem Stand der Technik entsprechen und schlagen neue Methoden vor, um deren Laufzeit zu reduzieren. In Experimenten an Graphen aus praktischen Anwendungen löst unser exakter Algorithmus mehr Probleminstanzen insgesamt und erreicht einen mehr als 4-fachen Speedup gegenüber dem nächstschnellsten exakten Algorithmus. Zudem verallgemeinern wir die Methoden des bisher besten exakten Algorithmus, sowie unsere Anpassungen desselben, auf gewichtete Graphen und erweitern daher die Anwendbarkeit des resultierenden Algorithmus. Für den approximativen Fall betrachten wir einen Lokale-Suche-Algorithmus, für den wir zeigen, dass eine gewisse Einschränkung des Suchraums die Approximationsgarantie erhält und gleichzeitig flächendeckende Speedups ergibt. Zudem motivieren wir die Bedeutung von Algorithmen mit Qualitätsgarantien indem wir beweisen, dass ein häufig verwendeter gieriger Algorithmus beliebig schlechte Ergebnisse liefern kann.

Contents

CHAPTER 1

Introduction

1.1 Motivation

An important task in the analysis of networks is the identification of vertices or groups of vertices which, in some sense, are central. In social networks, an advertiser may want to find influential people to promote their product. In logistics and supply chain networks, companies or governments want to find critical hubs that are worth monitoring or providing additional protection for. Centrality measures quantify importance by assigning a numeric value to a vertex or group of vertices. They are frequently used in the analysis of social networks [1], biological networks [2, 3], electrical power grids [4] and, among many other uses, in facility location problems [5]. 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 paths between other nodes, and degree centrality simply captures the number of neighbors a given node has. As noted by Everett and Borgatti [6], these measures fall short of identifying vertices that are 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 highest individual closeness centrality and groups with high group closeness centrality is relatively small [7]. Naturally, the question arises of how to efficiently find groups of a given size k such that a certain group centrality measure is maximized. For group closeness centrality, in the corresponding 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 $c(S) := (|V| - k)/(\sum_{v \in V} \min_{s \in S}\{\text{dist}(v, s)\})$. This NP-hard [8] problem, which is the subject of this thesis, has been treated in numerous previous works [7, 8, 9, 10, 11, 12]. In particular, our primary focus is on algorithms with quality guarantees, i.e. exact algorithms and algorithms that provide approximation guarantees.

1.2 Our Contribution

For the exact case, we review the state-of-the-art exact algorithm ILPIND by Staus et al. [10] and generalize their proposed search space restriction and Integer Linear Program (ILP) formulation, which are central aspects of the algorithm, to weighted graphs to broaden its applicability. Additionally, we propose two novel techniques to improve the runtime efficiency of their algorithm and argue that they carry over to the weighted case as well. As shown in our experiments, we achieve a more than 4-fold geometric mean speedup over their approach, enabling our algorithm to solve more instances overall. For the approximation case, we consider a local search algorithm proposed by Angriman et al. [11] that guarantees a constant factor approximation. We show that restricting the search space in a certain way preserves the approximation guarantee of the algorithm while reducing its runtime. Overall, we achieve geometric mean speedups of 34% to 56%, increasing with problem size. For 22% of instances, a speedup of at least 100% is achieved. Our experiments also show that the empirical solution quality is largely identical. In addition, we prove that a widely used greedy approach may yield arbitrarily poor results. To our knowledge, we are the first to show this negative result, which emphasizes the significance of exact and approximation algorithms.

1.3 Structure

We begin by reviewing graph-theoretic preliminaries and introducing notation used throughout this thesis in Chapter 2. The same chapter includes the formal definition of group closeness centrality and the problem of group closeness centrality maximization (GCCM). Subsequently, in Chapter 3, we present results from previous related work, with a focus on centrality measures and algorithmic approaches for GCCM. In Chapter 4, we give an overview of known exact algorithms for GCCM. In particular, we focus on the state-of-the-art exact algorithm by Staus et al. [10] and propose new techniques to reduce its runtime. Subsequently, the aforementioned generalizations of their methods and our proposed methods are presented. In Chapter 5, we consider known heuristic approaches for GCCM. We start by proving that a widely considered greedy approach does not approximate GCCM, which motivates the use of approximation algorithms, considered thereafter. To achieve the approximation guarantee, a local search algorithm is considered for which we propose pruning techniques to reduce its runtime while proving that our adaptation preserves the approximation guarantee. Both for the exact and approximation case, the effect of our proposed methods is evaluated experimentally in Chapter 6. We conclude with a discussion in Chapter 7.

CHAPTER 2

Preliminaries

2.1 Graph-Theoretic Notation and Definitions

For a given graph $G = (V, E)$, we denote as $V(G)$ and $E(G)$ the vertex set and edge set of G, respectively. Unless stated otherwise, the graphs we consider are undirected, connected, and unweighted. An edge $\{u, v\} \in E(G)$ is a set of size 2 with $u, v \in V(G)$, $u \neq v$. If $\{u, v\} \in E(G)$, we say that u and v are *adjacent*. We denote the number of vertices as $n(G) := |V(G)|$. Oftentimes, G is clear from the context and we simply write n, V and E instead of $n(G), V(G)$ and $E(G)$, respectively. The *neighborhood* $N(v)$ of a vertex v is the set of vertices which are adjacent to v, i.e. $N(v) = \{u \in V(G) \mid \{v, u\} \in E(G)\}$. The *closed neighborhood* $N[v]$ of vertex v also contains the vertex itself, i.e. $N[v] = N(v) \cup \{v\}$. Furthermore, we define a vertex $v \in V(G)$ to be *dominated* by some other vertex $u \in V(G)$ if $N[v] \subseteq N[u]$. The *degree* of a vertex v, denoted as $\deg(v)$, is the number of adjacent vertices $|N(v)|$. A *path* between vertices $u, v \in V(G)$, also referred to as a u-v path, is a sequence $v_1, \ldots, v_r \in V(G)$ of distinct vertices such that $u = v_1, v = v_r$, and $\{v_i, v_{i+1}\} \in E(G)$ for $i \in \{1, \ldots, r-1\}$. Since we only consider connected graphs, there is a path between every pair of vertices. For vertices u and v, we denote by $\text{dist}(u, v)$ the length, i.e. the number of edges, of a shortest path between u and v. For weighted graphs, $\text{dist}(u, v)$ is the minimum edge weight sum over all u-v paths with respect to some weight function $\omega : E(G) \to \mathbb{R}_{\geq 0}$. Hence, unweighted graphs can be viewed as weighted graphs with $\omega \equiv 1$ without changing the usual notion of $\text{dist}(\cdot, \cdot)$. In some graphs and for some vertices $u, v \in V(G)$, there is a vertex $w \notin \{u, v\}$ which cannot be avoided by any u-v path. Such a vertex w is called a *cut vertex* and clearly, the removal of w implies that G is no longer connected. As the *eccentricity* $\text{ecc}(v)$ of a vertex v, we define the maximum distance $\max_{u \in V(G)} \text{dist}(u, v)$ between v and any other vertex in G. The *diameter* $\text{diam}(G)$ of a graph G is the largest distance realizable in G, i.e. $\text{diam}(G) := \max_{v \in V(G)} \text{ecc}(v)$.

2.2 Group Closeness Centrality

Let G be a graph. The centrality measure considered in this work is called *group closeness centrality* and assigns a value $c(S) \in \mathbb{R}$ to a subset $S \subseteq V(G)$, often called a *(closeness) group*. Refer to Figure 2.1 for an illustration. We follow the definition

$$c(S) := \frac{n - |S|}{f(S)} \qquad \text{with} \qquad f(S) := \sum_{u \in V(G)} \min_{s \in S} \{\text{dist}(u, s)\},$$

where $f(S)$ is called the *(group) farness* of S. Instead of $\min_{s \in S}\{\text{dist}(u, s)\}$, we usually write $\text{dist}(u, S)$. Note that $\text{dist}(s, S) = 0$ for $s \in S$, and therefore some authors sum over $V(G) \setminus S$ in the definition of $f(S)$.

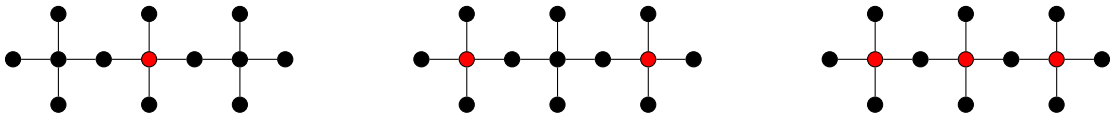


Figure 2.1: Maximum closeness groups (marked red) for $k = 1, 2, 3$ in an example graph.

In this work, we consider the NP-hard problem of finding a group $S \subseteq V(G)$ of size $|S| = k > 0$ with maximum group closeness centrality, or equivalently, minimum group farness. We refer to this problem as *group closeness centrality maximization* and write GCCM for short.

2.3 Submodularity and Supermodularity

A concept frequently appearing throughout this thesis captures that many set functions $f : \mathcal{P}(U) \to \mathbb{R}$ defined on subsets of a *universe* U, that model real-world values, have a diminishing return property. For example, an advertiser might realize that sales do not scale linearly with the amount spent on advertising, because additional advertisements usually target less receptive audiences or reach customers that have already purchased the product[1]. Formally, a set function $f : \mathcal{P}(U) \to \mathbb{R}$ is called *submodular* if

$$f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B) \quad \forall\, A \subseteq B \subseteq U, e \notin B.$$

This definition captures the property of diminishing return in a discrete setting. A related property is called *supermodularity*, and we call a set function $f : \mathcal{P}(U) \to \mathbb{R}$ supermodular, if

$$f(A \cup \{e\}) - f(A) \leq f(B \cup \{e\}) - f(B) \quad \forall A \subseteq B \subseteq U, e \notin B.$$

It is easy to see that for a submodular set function $f(S)$, the additive inverse $-f(S)$ is supermodular, and vice versa. However, the reciprocal $\frac{1}{f(S)}$ of a supermodular set function $f(S)$ is not necessarily submodular, and this fact will become relevant later.

[1] For this example, we assume a customer is less likely to purchase the product twice.

2.4 Integer Linear Programs

Many combinatorial optimization tasks can be expressed as an *Integer Linear Program* (ILP). To understand ILPs, let us first consider *Linear Programs* (LP). An LP can be thought of as an optimization task, which has an *objective function* $f : \mathbb{R}^n \to \mathbb{R}$, that is linear in some vector $x \in \mathbb{R}^n$, i.e. it can be expressed as $f(x) := c^T x = c_1 x_1 + \cdots + c_n x_n$ for some *cost vector* $c \in \mathbb{R}^n$. The objective function can be thought of as a cost or a profit, and therefore one is interested in the minimum or maximum of such a function within some admissible area of $\mathbb{R}^n$. Vectors within this admissible area are part of the *feasible set* F of the LP, which are those vectors that satisfy a set of given *constraints*. For LPs, these constraints must also be linear in x and take the form of either $a_i^T x = b_i$, $a_i^T x \leq b_i$ or $a_i^T x \geq b_i$ with $a_i \in \mathbb{R}^n, b_i \in \mathbb{R}$ for $1 \leq i \leq m$. A solution $x^* \in F$ is called optimum, if $f(x^*) \leq f(x)$ for all $x \in F$ for a minimization task, or $f(x^*) \geq f(x)$ for all $x \in F$ for a maximization task. While (I)LPs can be *infeasible*, i.e. $F = \emptyset$, we always assume feasibility of the ILPs considered in this thesis. Additionally, we always assume the objective value is bounded. Refer to Figure 2.2 for an example of an LP.

$$
\begin{aligned}
\text{maximize} \quad & x_2 \quad \text{subject to} \\
x_1 &\geq 0 \\
x_2 &\geq 0 \\
x_1 + \tfrac{3}{10} x_2 &\leq 3 \\
-\tfrac{2}{3} x_1 + x_2 &\leq 2
\end{aligned}
$$

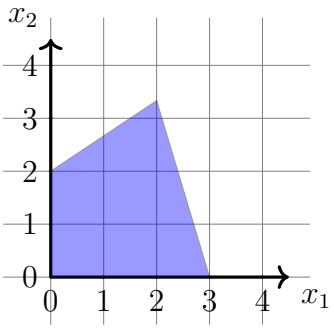


Figure 2.2: Example of an LP. On the left, the LP is given as a linear objective function that ought to be maximized subject to given linear constraints. In the plot on the right, the feasible set is filled blue. Clearly, $x^* = (2, \frac{10}{3})$ is the optimum solution of the LP.

$$
\begin{aligned}
\text{maximize} \quad & x_2 \quad \text{subject to} \\
x_1 &\geq 0 \\
x_2 &\geq 0 \\
x_1 + \tfrac{3}{10} x_2 &\leq 3 \\
-\tfrac{2}{3} x_1 + x_2 &\leq 2 \\
x_1, x_2 &\in \mathbb{Z}
\end{aligned}
$$

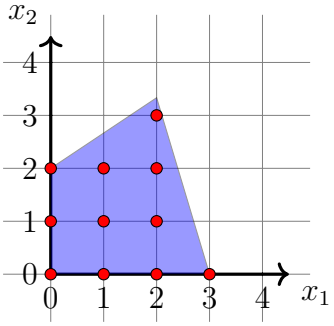


Figure 2.3: The LP from Figure 2.2 with additional integer constraints, hence this is an ILP. The feasible set $F' = F \cap \mathbb{Z}^2$ is marked with red dots. Clearly, $x^* = (2, 3)$ is the optimum solution of the ILP.

For ILPs, additional constraints are introduced. Now, feasible vectors are required to be in $\mathbb{Z}^n$, so only integer coordinates are feasible, i.e. we now consider the feasible set $F' := F \cap \mathbb{Z}^n$. Take for instance the LP from Figure 2.2 with additional integer constraints (making this an ILP) as depicted in Figure 2.3. The LP of Figure 2.2 is called the *linear relaxation* of the ILP of Figure 2.3. Solving the linear relaxation of an ILP that ought to be maximized clearly gives a solution with objective value at least that of the ILP. Therefore, linear relaxations are useful for bounds in branch-and-bound ILP solvers. While LPs can be solved efficiently, solving ILPs can be very hard in the worst case. We will not go into further detail of the algorithmic complexity of the various algorithms for solving LPs and ILPs, just note that LPs can be solved in polynomial time [13, 14] while solving ILPs is NP-hard [15].

CHAPTER 3

Related Work

3.1 Centrality Measures

The history of centrality measures dates back to the mid-20th century [16, 17, 18], to works often motivated by the analysis of social networks. Various centrality measures have been proposed, among them are betweenness centrality, degree centrality, eigenvector centrality, and many more [19]. A widely used measure is *closeness centrality*, which assigns a value of

$$c(v) := \frac{n(G) - 1}{\sum_{u \in V(G)} \text{dist}(u, v)}$$

to a vertex $v \in V(G)$. It was first proposed by Bavelas [16] and later by Beauchamp [20] and Sabidussi [21]. Along with other centrality measures, Everett and Borgatti [6] generalized this notion to groups of vertices. Since varying definitions exist, we define the *group closeness centrality* for a set $S \subseteq V(G)$ to be

$$c(S) := \frac{n(G) - |S|}{\sum_{u \in V(G)} \text{dist}(u, S)},$$

where $\text{dist}(u, S) := \min_{s \in S}\{\text{dist}(u, s)\}$. Other definitions instead use $n(G)$ or 1 as the numerator. Often it is useful to consider the denominator $f(S) := \sum_{u \in V(G)} \text{dist}(u, S)$, called *group farness*. In fact, to maximize $c(S)$ (subject to $|S| = k$ for some $k \in \mathbb{N}$), we may equivalently minimize group farness $f(S)$. Regardless of the exact definition, a set $S \subseteq V(G)$ which maximizes $c(S)$ subject to $|S| = k$ clearly also maximizes group closeness defined with some other numerator. Since this thesis is concerned with finding and approximating such sets, the exact definition of $c(S)$ used is therefore inconsequential in this pursuit.

3.2 Algorithms for Group Closeness Centrality Maximization

Challenging combinatorial problems arise when asking for a group $S \subseteq V(G)$ of size $k \in \mathbb{N}$ with maximum group closeness centrality. Chen et al. [8] considered this problem and claimed NP-hardness by giving a reduction from the k-median problem[2]. 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* [10], while others refer to it as *group closeness maximization* [7]. We refer to the problem as *group closeness centrality maximization* (GCCM), to clearly differentiate the problem from the function that ought to be maximized. As briefly mentioned, we may equivalently minimize group farness $f(S)$ subject to $|S| = k$, i.e. ask for a set $S^* = \operatorname{argmin}_{S \subseteq V(G), |S|=k} f(S)$. Also note that group farness is supermodular [7].

To address the hardness of GCCM, Chen et al. [8] proposed a simple greedy heuristic, which we call GREEDY, and a sampling algorithm for large graphs. Essentially, GREEDY first starts with the empty set $S_0 = \emptyset$ and iteratively adds the vertex that yields the highest marginal gain in group closeness centrality. More precisely, when S_i is the current set and $i < k$, then GREEDY selects the vertex $v \in V(G) \setminus S_i$ with highest marginal gain $c(S_i \cup \{v\}) - c(S_i)$ to obtain $S_{i+1} := S_i \cup \{v\}$. In their work, Chen et al. [8] claimed that GREEDY has an approximation ratio of $(1 - 1/e)$, i.e. they claimed that for groups S_k produced by GREEDY, it holds that $c(S_k) \geq (1 - 1/e)c(S^*)$, where S^* denotes an optimum group. In their proof, they (implicitly) assumed that group closeness centrality is submodular, which led them to use a result due to Nemhauser et al. [23] for non-negative monotone submodular functions from which the approximation guarantee arises. Bergamini et al. [24] proposed an improved greedy algorithm called GREEDY++, which computes the same solution as GREEDY in much lower time by, among other techniques, exploiting the supermodularity of group farness. However, in a corrected version [7], they later pointed out that the submodularity assumption in Chen et al. [8] was invalid by providing a counter-example, and therefore showed that the approximation claim for GREEDY and GREEDY++ cannot be supported by the proof strategy used in Chen et al. [8]. Consequently, the question of whether GCCM could be approximated was considered unsettled [11]. Nonetheless, Bergamini et al. [7] still observed very high quality solutions in experiments. Angriman et al. [9] proposed two local search algorithms, LOCALSWAPS and GROWSHRINK, both of which are based on performing vertex exchanges to improve the given group. For both LOCALSWAPS and GROWSHRINK, no approxima-

[2] Actually, Chen et al. [8] describe a reduction from the Euclidean k-*means* clustering problem and cite [22]. As is, we believe this reduction to be invalid, likely due to a confusion with the k-median problem, as Euclidean k-means clustering is NP-hard even for $k = 2$ [22], while GCCM is solvable in polynomial time for fixed k. However, a reduction from the k-median problem, similar to the one outlined by Chen et al. [8], is valid. Also note that other authors [10] observed NP-hardness by relating GCCM to DOMINATING SET.

tion guarantee is known [9]. In a later work, Angriman et al. [11] proposed local search algorithms with approximation guarantees, hence settling the approximability question of GCCM. For instance, their algorithm GREEDY-LS-C starts with the solution computed by GREEDY/GREEDY++ and subsequently refines it with a specific local search technique. Essentially, this local search is based on performing swaps between the solution and the remaining vertices which increase the objective function, until no swap can improve the solution. This swap-based local search algorithm was considered and analyzed by Arya et al. [25] and originally intended for the related k-median problem. In their work, Angriman et al. [11] argue that this algorithm, which produces groups S with $c(S) \geq \frac{1}{5}c(S^*)$, can be used for GCCM.

The first exact algorithm for GCCM was given in form of an ILP by Bergamini et al. [7] to experimentally evaluate the solution quality of heuristic algorithms. To obtain heuristic solutions, and exact solutions when run until termination, Rajbhandari et al. [12] proposed an anytime algorithm called PRESTO. Staus et al. [10] proposed exact algorithms that utilize ILP solvers and branch-and-bound algorithms that exploit the supermodularity of group farness to prune the search space. Among these, their algorithm ILPIND is currently the state-of-the-art exact algorithm for GCCM. Essentially, ILPIND solves an ILP model which might not have enough variables and constraints to represent the problem. Crucially, once the ILP has been solved, it can be efficiently checked if the variables and constraints at hand are sufficient. If not, variables and constraints are subsequently added and the ILP is solved again. These steps are repeated until the model is sufficient to represent the problem and thus to obtain a valid solution. Staus et al. [10] also showed that a certain subset of the vertices is guaranteed to contain an optimum group. Essentially, this set is constructed such that, for each $v \in V(G)$, it contains v itself or some vertex which dominates v. Since an optimum solution can be found within this subset of $V(G)$, their algorithms construct this subset, to which the search is then restricted.

Since group farness $f(S)$ is supermodular, the problem can also be viewed as submodular maximization of $-f(S)$. Note that the result in Nemhauser et al. [23] applies to *non-negative* submodular monotone functions, hence this result cannot be applied here to obtain an approximation algorithm of $-f(S)$. Still, exact algorithms for generic cardinality constrained submodular function maximization such as [26] can be used to maximize $-f(S)$ subject to $|S| = k$ for some $k \in \mathbb{N}$, and therefore to solve GCCM.

Other notable related works include Zhao et al. [27], in which a notion of group closeness centrality, which estimates the original definition, is considered. Unlike group closeness centrality, this estimate is submodular. Therefore, their results do not generally carry over to the widely used definition of group closeness centrality considered in this work. Li et al. [28] consider current flow group closeness centrality, for which a different notion of distances is used. For this problem, they propose approximation algorithms. However, these algorithms are not approximation algorithms in the usual sense [11], that guarantee the objective function value to be bounded by a constant multiple of the optimum objective value.

CHAPTER 4

Exact Algorithms

The prevailing methods for finding a group of size k with globally maximum group closeness centrality tend to fall into two categories. On one hand, the problem can be solved by branch-and-bound algorithms which use the supermodularity of group farness to prune the search space. On the other hand, the problem can be formulated as an ILP and thus be solved by one of the many available ILP-solvers. Bergamini et al. [7] propose a simple ILP formulation to experimentally evaluate the quality of solutions produced by certain heuristic algorithms. Staus et al. [10] propose several exact algorithms. Among these are both branch-and-bound algorithms and algorithms that make use of ILP solvers. Unlike Bergamini et al. [7], their algorithm ILPIND solves several ILPs, one in each iteration of the algorithm. Essentially, ILPIND attempts to use as few decision variables as possible to decrease solution time. Doing so, it risks that the ILP model may, in a certain sense, not be sufficient to compute an optimum solution. However, an insufficient ILP model can be detected once the ILP has been solved. In that case, more variables and constraints are added iteratively, until the model is eventually sufficient. We elaborate further on their method in Section 4.2. As part of our own work, we propose techniques to both reduce the number of iterations and variables in ILPIND, in Section 4.3. Note that both the work by Staus et al. [10] and our subsequent refinements considered in Section 4.3 apply to unweighted graphs. Therefore, we generalize a search space reduction proposed by Staus et al. [10] and an ILP formulation, both of which are crucial to ILPIND, to weighted graphs in Section 4.4, in which we also argue that our refinements carry over to the weighted case.

As previously mentioned, Rajbhandari et al. [12] propose an algorithm which finds exact solutions if run until termination. However, we were unable to obtain an implementation and their benchmarks indicate that for the exact case, their algorithm requires orders of magnitude more runtime than the exact algorithms proposed by Staus et al. [10]. Therefore, we will not be considering their approach here.

4.1 A simple ILP

We now present the ILP formulation due to Bergamini et al. [7]. Their formulation not only illustrates that computationally, GCCM is usually formulated as minimization of group farness, but also serves as a good introductory example before considering more advanced approaches. In the formulation, there is a variable y_u for each $u \in V(G)$ that indicates whether u is in the computed solution S^*, i.e. $y_u = 1$ if $u \in S^*$ and $y_u = 0$ otherwise. Additionally, there are variables $x_{u,v}$, $u, v \in V(G)$ which indicate that u selects v as the closest vertex in S^*. The full ILP formulation reads as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{u \in V(G)} \sum_{v \in V(G)} \text{dist}(u,v) \cdot x_{u,v} \quad \text{subject to} \\
& k = \sum_{u \in V(G)} y_u && (4.1) \\
& \forall u \in V(G): \\
& 1 = \sum_{v \in V(G)} x_{u,v} && (4.2) \\
& x_{u,v} \leq y_v \quad \forall v \in V(G) && (4.3) \\
& y_u \in \{0,1\} \\
& x_{u,v} \in \{0,1\} \quad \forall v \in V(G)
\end{aligned}$$

For some $u, v \in V(G)$, assume $x_{u,v} = 1$, indicating that u selects v as the closest vertex in S^*. Then, v must be in S^* and therefore it must be the case that $y_v = 1$, as ensured by constraints (4.3). Constraint (4.1) ensures that the group has size k. Constraints (4.2) make sure that each vertex is assigned to exactly one vertex, i.e. selects one vertex which represents the closest vertex from the solution. The objective naturally corresponds to group farness. Note that the number of variables and constraints is quadratic in $n(G)$.

4.2 State of the Art

To our knowledge, Staus et al. [10] are the first to primarily consider exact algorithms for GCCM. Their two most efficient algorithms are called DvInd and IlpInd. Out of the two, IlpInd is to be regarded as superior. This assessment is well-supported by their own benchmarks [10], in which IlpInd outperforms DvInd for the majority of instances. The high-level idea of IlpInd is to drastically reduce the number of decision variables and constraints used in the ILP model and solve the problem in iterations. In the following paragraphs, we explore in detail how this is done.

Roughly speaking, the algorithm solves an ILP model which, in a certain sense later defined, might not be sufficient to represent the problem. Crucially, the model can be detected as being insufficient once the ILP has been solved. Variables and constraints are

subsequently added and the ILP is solved again. These steps are repeated until the model is eventually sufficient. It is not immediately clear how this technique affects runtime. After all, we are guaranteed to find an optimum group with a *single* optimization of the ILP given by Bergamini et al. [7]. However, the ILP models in ILPIND usually have a significantly smaller number of variables and constraints, leading to reduced runtime despite having to solve several such ILPs [10]. We now give an overview of the techniques used in ILPIND.

First, let us get familiar with the ILP formulation used in ILPIND. For each $v \in V(G)$, there are binary decision variables $x_{v,i}$ with $i \in \{0, \dots, d(v)\}$, which indicate the distance of v to the closest vertex in the solution S^*. More precisely, $x_{v,i} = 1$ indicates $\text{dist}(v, S^*) = i$ for $i < d(v)$ and $\text{dist}(v, S^*) \geq i$ for $i = d(v)$, where $d(v) \in \mathbb{N}$ is a value that depends on v. Essentially, $d(v)$ is the maximum distance from v to the closest vertex in the solution, which can be represented by the ILP. For example, we might have a vertex $v \in V(G)$ with $\text{dist}(v, S^*) = 4$ for some optimum solution S^*. Then, if $d(v) < 4$, the ILP cannot represent $\text{dist}(v, S^*) = 4$, since no variable $x_{v,4}$ exists. Such an ILP model would therefore not be sufficient. The idea of Staus et al. [10] is, essentially, to start with $d(v) = 2$ for all $v \in V(G)$ and solve the ILP model. If in the solution, $x_{v,d(v)} = 1$ for any $v \in V(G)$ with $d(v) < \text{ecc}(v)$, then $d(v)$ is insufficiently large for v (since possibly $d(v, S^*) > d(v)$) and the increment $d(v) = d(v) + 1$ is performed for all such vertices, followed by re-optimization of the ILP model. This is done iteratively until the model is eventually sufficient, i.e. for all vertices $v \in V(G)$ we get $x_{v,d(v)} = 0$ or $d(v) \geq \text{ecc}(v)$. Note that since $d(v) \geq \text{ecc}(v)$ is sufficient, this iterative process terminates. From the sufficient, optimized model, S^* can be obtained as $S^* = \{v \in V(G) \mid x_{v,0} = 1\}$. The full formulation reads as follows:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{v \in V(G)} \sum_{i \in \{0,\dots,d(v)\}} i \cdot x_{v,i} \quad \text{subject to} \\
& k = \sum_{v \in V(G)} x_{v,0} \\
& \forall v \in V(G): \\
& 1 = \sum_{i \in \{0,\dots,d(v)\}} x_{v,i} \\
(\star) \quad & \left\{ \quad x_{v,i} \leq \sum_{\substack{w \in V(G), \\ \text{dist}(v,w)=i}} x_{w,0} \quad \forall i \in \{0, \dots, d(v)-1\} \right. \\
& \qquad x_{v,i} \in \{0,1\} \quad \forall i \in \{0, \dots, d(v)\}
\end{aligned}
\tag{4.4}
$$

The constraints are mostly analogous to the constraints used in the formulation by Bergamini et al. [7]. Note that for $v \in V(G)$ and $i < d(v)$, constraints $(\star)$ ensure that the assignment $x_{v,i} = 1$ requires the existence of a vertex $w \in V(G)$ at distance $\text{dist}(v, w) = i$ that is in the solution, i.e. $x_{w,0} = 1$. However, $x_{v,d(v)}$ is not constrained in this sense, as $x_{v,d(v)} = 1$ ought to indicate $\text{dist}(v, S^*) \geq d(v)$.

For high level pseudocode of the ILPIND algorithm refer to Algorithm 1. For a proof of correctness, we refer the reader to the work of Staus et al. [10].

Algorithm 1 ILPIND

```
1: function SOLVE(G: GRAPH, k : ℕ)
2:     for v ∈ V(G) do
3:         d(v) ← 2
4:     while true do
5:         solve ILP (4.4) using {d(v)}_{v∈V(G)}
6:         ilpSufficient ← true
7:         for v ∈ V(G) do
8:             if not SUFFICIENT(v) then
9:                 ilpSufficient = false
10:                d(v) ← d(v) + 1
11:        if ilpSufficient then
12:            return {v ∈ V(G) | x_{v,0} = 1}
13:
14: function SUFFICIENT(v : VERTEX)
15:     return x_{v,d(v)} = 0 ∨ d(v) ≥ ecc(v)
```

Another technique used by Staus et al. [10] is based on their observation that only a subset of the search space has to be explored. More precisely, it is permissible to only consider groups in $D \subseteq V(G)$, where for each vertex $v \in V(G)$, $v \in D$ or some vertex $w \in D$ exists which dominates v. Oversimplifying, this allows a vertex in an optimum solution which is not in D to be replaced by the vertex in D, by which it is dominated. The set D is called *sufficient* by Staus et al. [10] and, as outlined, is guaranteed to contain an optimum group $S^* \subseteq D$. One subtlety to note here is that when k is large enough, it might exceed the size of D. In that case, we can add arbitrary vertices from $V(G) \setminus D$ to D until D has size k and obtain an optimum solution S^* without the need of solving any ILPs. We therefore assume, in the remainder of the text, that $k \leq |D|$. In their work, Staus et al. [10] consider a more general notion of domination and sufficient sets. However, we only use the aforementioned notion sufficient sets and we therefore simply call such sets *sufficient*. Similarly, we use a single notion of domination and we therefore simply say that a vertex $u \in V(G)$ is dominated by some other vertex $v \in V(G)$ if $N[u] \subseteq N[v]$[3]. As a consequence of being able to find an optimum group in D, in the ILP formulation, the variable $x_{v,0}$ is not necessary for $v \in V(G) \setminus D$ and may be replaced by zero. For a detailed exposition and proof of correctness, see [10].

[3] Staus et al. [10] use a slightly more permissive notion of domination, i.e. their constructed sufficient sets can be smaller. We found the practical difference to be negligible and use the simpler aforementioned, slightly more restrictive definition of domination. Later, we show that this definition generalizes nicely to the weighted case.

4.3 Novel Techniques

In this section, we first propose a technique which aims to reduce the number of iterations needed in the iterative approach of Staus et al. [10]. Essentially, this is achieved by adjusting the initial values $\{d(v)\}_{v\in V(G)}$ to carefully chosen values based on high quality heuristic solutions. Additionally, we propose a second technique which allows most real-world graphs to be reduced in size while maintaining the structure of the original problem. This reduction can allow for the removal of some variables in the ILP formulation while preserving the "character" of the original problem by adjusting weights in the objective function.

4.3.1 Reducing the Number of Iterations

Recall that in the ILPIND algorithm by Staus et al. [10], $d(v)$ is initialized to $d(v) = 2$ for all $v \in V(G)$. 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 model. After the model is optimized with the current $\{d(v)\}_{v\in V(G)}$, we may find that $x_{v,d(v)} = 1$ for some vertices v in the solution, 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, $d(v)$ needs to be incremented to make sure that a possible solution S^* with $\text{dist}(v, S^*) > d(v)$ can be represented by the ILP model. As a result, Staus et al. [10] obtain an iterative algorithm which solves an ILP model in each iteration and checks if the values $\{d(v)\}_{v\in V(G)}$ are sufficient in this sense. If not, they are incremented and the algorithm proceeds with the next iteration. Otherwise, the solution of the ILP model corresponds to a globally optimum solution and the algorithm terminates.

We argue that oftentimes, we can be quite certain that the initial $\{d(v)\}_{v\in V(G)}$ values used by Staus et al. [10] are insufficient. Even more, we might in fact have a good estimate of $\{\text{dist}(v, S^*)\}_{v\in V(G)}$. Our idea is to use this estimate to derive an initial estimate of $\{d(v)\}_{v\in V(G)}$ that is much closer to the final, sufficient values. Therefore, these values need not be incremented as many times and the iteration count is reduced. To obtain these values, the idea is to initially estimate

$$d(v) \approx \text{dist}(v, S^*) + 1. \tag{4.5}$$

Indeed, if our estimate $\{\text{dist}(v, S^*)\}_{v\in V(G)}$ was exact, these values permit a sufficient solution of the ILP model. Of course, we cannot efficiently supply an exact estimate of $\{\text{dist}(v, S^*)\}_{v\in V(G)}$. However, we can compute a high quality heuristic solution $\widetilde{S}$ and use $\{\text{dist}(v, \widetilde{S})\}_{v\in V(G)}$ as an estimate. Then, analogously to (4.5), we use

$$d(v) = \max\{2, \text{dist}(v, \widetilde{S}) + 1\}.$$

as an initial assignment of $\{d(v)\}_{v\in V(G)}$. Taking the maximum of 2 and $\text{dist}(v, S^*) + 1$ ensures our initial $\{d(v)\}_{v\in V(G)}$ values are at least as high as those used in ILPIND and

has shown to be beneficial for performance in preliminary experiments. To compute the heuristic solution $\widetilde{S}$, we use the GS-LS-C algorithm by Angriman et al. [11], which is guaranteed to supply a constant factor approximation of the exact solution as briefly mentioned in Chapter 3. More precisely, this algorithm is guaranteed to produce a solution with group farness of at most $5 \cdot f(S^*)$. For the practical effect of this technique and in particular its influence on the number of iterations in our algorithm, see Chapter 6. Also, note that GREEDY/GREEDY++ gives an exact solution for $k = 1$. Therefore, GREEDY++ is run for $k = 1$ and its solution is returned directly, eliminating the need to do local search or to solve any ILPs.

4.3.2 Data Reductions

Recall that Staus et al. [10] construct a set $D \subseteq V(G)$ which is guaranteed to contain a globally optimum solution $S^* \subseteq D$. For the ILP model, this has the desirable consequence that their algorithm may omit creating the variable $x_{v,0}$ for $v \in V(G) \setminus D$. However, for these $v \in V(G) \setminus D$, the variables $x_{v,1}, \ldots, x_{v,d(v)}$ are still created. In some situations, this seems unnecessary. Take for instance a vertex v with $\deg(v) = 1$, which is dominated by its neighbor w, we thus have $w \in D$, $v \in V(G) \setminus D$. We then know that the shortest path from v to S^* will necessarily go through w. In the optimum solution, we therefore have $\text{dist}(v, S^*) = \text{dist}(w, S^*) + 1$. Hence, *all* variables $x_{v,0}, \ldots, x_{v,d(v)}$ associated with v can be removed in the ILP model, and the costs associated with the variables of w are adjusted as follows:

(i) We associate a cost of 1 with $x_{w,0} = 1$, which corresponds to $w \in S^*$. We therefore need to account for the *hidden* cost of its removed neighbor v, which amounts to $d(v, S^*) = d(v, w) = 1$.

(ii) We associate a cost of $2i + 1$ with $x_{w,i} = 1$, $i = 1, \ldots, d(v)$. Here, the assignment $x_{w,i} = 1$ corresponds to $d(w, S^*) = i$ and we need to additionally account for the *hidden* cost of its removed neighbor v, which amounts to $d(v, S^*) = d(w, S^*) + 1 = i + 1$. Therefore, we associate a total cost of $2i + 1$ with the assignment $x_{w,i} = 1$.

This distinction merely serves an explanatory purpose and can be condensed into a single cost adjustment as both cases are encapsulated by associating a cost of $2i+1$ with $x_{w,i} = 1$ for all $i \in \{0, \ldots, d(v)\}$.

In general, a vertex $w \in D$ might have several, say $\alpha \in \mathbb{N}$, degree 1 neighbors which are not in D. Then, in analogy to the special case above, we may remove all these neighbors and associate a cost of $\alpha \cdot (i + 1) + i$ with $x_{w,i} = 1$ for all $i \in \{0, \ldots, d(v)\}$. As discussed later, we find that more cases permit the removal of a vertex v together with the adjustment of the cost associated with the variables of a neighbor w. In these cases, we say that w *absorbs* v and denote as $\alpha(w)$ the number of neighbors which w absorbs. For an absorbed vertex v, we denote as $\rho(v)$ the vertex, by which v was absorbed. We denote the total set of absorbed vertices as A. The ILP formulation by Staus et al. [10] needs to be adjusted

correspondingly and now reads as follows:

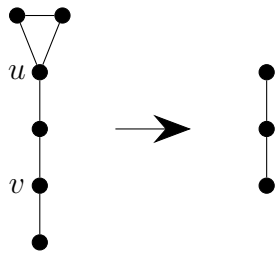
$$
\begin{aligned}
\text{minimize} \quad & \sum_{v \in V(G) \setminus A} \; \sum_{i \in \{0,\ldots,d(v)\}} c_{v,i} \cdot x_{v,i} \quad \text{subject to} \\
& k = \sum_{v \in V(G) \setminus A} x_{v,0} \\
& \forall v \in V(G) \setminus A : \\
& 1 = \sum_{i \in \{0,\ldots,d(v)\}} x_{v,i} \\
& x_{v,i} \leq \sum_{\substack{w \in V(G) \setminus A, \\ \text{dist}(v,w)=i}} x_{w,0} \quad \forall i \in \{0, \ldots, d(v) - 1\} \\
& x_{v,i} \in \{0,1\} \quad \forall i \in \{0, \ldots, d(v)\}
\end{aligned}
\tag{4.6}
$$

with costs $c_{v,i} = \alpha(v) \cdot (i+1) + i$. As mentioned, there are more cases which permit a vertex $v \notin D$ to be removed (absorbed). More generally, v can be absorbed by some neighbor $w \in D$ if the following conditions are met:

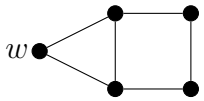
(i) $v \in V(G) \setminus D$ is dominated by some cut vertex $w \in D$

(ii) all vertices in the component of v in $G - w$ are in $V(G) \setminus D$ and also dominated by w

If both conditions are met, the vertices $u_1, \ldots, u_l$ in the component of v in $G - w$, including v, are excluded from D and dominated by $w \in D$. Hence, there is an optimum solution $S^* \subseteq D$, which does not contain any of the $u_1, \ldots, u_l$. Their respective shortest paths to S^* must therefore go through the cut vertex w and our reasoning from above applies. Refer to Figure 4.1 for an illustration of the reduction rules.



(a) u and v are cut vertices satisfying the conditions to absorb the vertices they dominate. After the reductions, we have $\alpha(u) = 2$ and $\alpha(v) = 1$



(b) Since w is dominated by two vertices which are not cut vertices, it is not clear which vertex should absorb w.

Figure 4.1: The reduction rules applied to an example graph (Subfigure 4.1a) and an example of a vertex that is dominated, but cannot be reduced (Subfigure 4.1b). Hence, for a sufficient set $D \subseteq V(G)$ we get $A \subseteq V(G) \setminus D$, but not necessarily $A = V(G) \setminus D$.

For high-level pseudocode of our algorithm GROVER[4], see Algorithm 2. For a rigorous treatment of the correctness of the reduction, refer to Theorem 1 and its proof.

Algorithm 2 GROVER

```
1:  function SOLVE(G: GRAPH, k : ℕ)
2:      compute heuristic solution S̃
3:      if k = 1 then
4:          return S̃ // heuristic is exact for k = 1
5:      compute A and {α(v)}_{v∈V(G)\A}
6:      for v ∈ V(G) \ A do
7:          d(v) ← max{2, dist(v, S̃) + 1}
8:      while true do
9:          solve ILP (4.6) using {d(v)}_{v∈V(G)\A} and {α(v)}_{v∈V(G)\A}
10:         ilpSufficient ← true
11:         for v ∈ V(G) \ A do
12:             if not SUFFICIENT(v) then
13:                 ilpSufficient ← false
14:                 d(v) ← d(v) + 1
15:         if ilpSufficient then
16:             return {v ∈ V(G) \ A | x_{v,0} = 1}
17:
18: function SUFFICIENT(v : VERTEX)
19:     return x_{v,d(v)} = 0 ∨ d(v) ≥ ecc(v)
```

Theorem 1. *The given reduction is correct, i.e. from the sufficient, optimized ILP* (4.6)*, an optimum solution S^* can be obtained as $S^* = \{v \in V(G) \setminus A \mid x_{v,0} = 1\}$.*

Proof. To show the claim, we leverage the proof of correctness for ILPIND by Staus et al. [10] by constructing an optimum solution of the ILP (4.4) used in ILPIND from the sufficient, reduced ILP (4.6), while ensuring that the sufficiency of the model is preserved.

Let $\{\widetilde{x}_{v,i} \mid 0 \leq i \leq d(v)\}_{v \in V(G) \setminus A}$ be the variables of the reduced model with values $\{d(v)\}_{v \in V(G) \setminus A}$ 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 $v \in V(G) \setminus A$, we have $\widetilde{x}_{v,d(v)} = 0$ or both $\widetilde{x}_{v,d(v)} = 1$ and $\text{ecc}(v) \leq d(v)$.

Consider the original ILP model (4.4) used in ILPIND, for which we introduce variables $\{x_{v,i} \mid 0 \leq i \leq d(v)\}_{v \in V(G)}$, with the same $d(v)$ as the reduced model for $v \in V(G) \setminus A$ and

[4] **Gro**up closeness centrality maximization sol**ver**

$d(v) = d(\rho(v)) + 1$ for $v \in A$, where $\rho(v)$ denotes the vertex which absorbed v. We assign

$$x_{v,i} = \begin{cases} 0 & v \in A, i = 0 \\ \widetilde{x}_{\rho(v),i-1} & v \in A, i \geq 1 \\ \widetilde{x}_{v,i} & v \in V(G) \setminus A \end{cases} \qquad \text{for } 0 \leq i \leq d(v). \tag{4.7}$$

It is simple to verify that this construction yields a feasible and sufficient solution of the original ILP model. It remains to show that this variable assignment corresponds to an optimum solution. First note that by the reduction rules, the identity

$$\sum_{v \in V(G) \setminus A} \sum_{i \in \{0,\dots,d(v)\}} \alpha(v) \cdot (i+1) \cdot \widetilde{x}_{v,i} = \sum_{v \in A} \sum_{i \in \{0,\dots,d(\rho(v))\}} (i+1) \cdot \widetilde{x}_{\rho(v),i} \tag{4.8}$$

holds. Roughly speaking, this identity expresses that the hidden 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 (4.6), and f_* of the original ILP (4.4) used in ILPIND, we then get

$$\begin{aligned} \widetilde{f}_* &= \sum_{v \in V(G) \setminus A} \sum_{i \in \{0,\dots,d(v)\}} c_{v,i} \cdot \widetilde{x}_{v,i} \\ &= \sum_{v \in V(G) \setminus A} \sum_{i \in \{0,\dots,d(v)\}} (\alpha(v) \cdot (i+1) + i) \cdot \widetilde{x}_{v,i} \\ &= \sum_{v \in V(G) \setminus A} \sum_{i \in \{0,\dots,d(v)\}} i \cdot \widetilde{x}_{v,i} + \sum_{v \in V(G) \setminus A} \sum_{i \in \{0,\dots,d(v)\}} \alpha(v) \cdot (i+1) \cdot \widetilde{x}_{v,i} \\ &\overset{(4.8)}{=} \sum_{v \in V(G) \setminus A} \sum_{i \in \{0,\dots,d(v)\}} i \cdot \widetilde{x}_{v,i} + \sum_{v \in A} \sum_{i \in \{0,\dots,d(\rho(v))\}} (i+1) \cdot \widetilde{x}_{\rho(v),i} \\ &\overset{(\star)}{=} \sum_{v \in V(G) \setminus A} \sum_{i \in \{0,\dots,d(v)\}} i \cdot x_{v,i} + \sum_{v \in A} \sum_{i \in \{0,\dots,d(v)\}} i \cdot x_{v,i} \\ &= \sum_{v \in V(G)} \sum_{i \in \{0,\dots,d(v)\}} i \cdot x_{v,i} \\ &= f_*, \end{aligned} \tag{4.9}$$

where $(\star)$ follows by applying (4.7) together with an index shift. Hence, the constructed solution of the original ILP (4.4) used in ILPIND has the same objective value $f_* = \widetilde{f}_*$ as the solution of the reduced ILP (4.6). 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 $\{\hat{x}_{v,i} \mid 0 \leq i \leq d(v)\}_{v \in V(G)}$. We argue that from this *better* solution, we can construct a *better* solution of the reduced model, contradicting its

optimality. Indeed, using $\widetilde{x}_{v,i} = \hat{x}_{v,i}$ for all $v \in V(G) \setminus A$ and $0 \leq i \leq d(v)$ as an assignment for the reduced ILP, we obtain a feasible solution of the reduced model. Applying the same rearrangements as in (4.9), we conclude that this solution has objective value $f' < \widetilde{f}_*$, a contradiction to the optimality of the solution of the reduced ILP. □

4.4 Generalization to Weighted Graphs

We now propose generalizations to weighted graphs of the techniques used in ILPIND and GROVER. First, we show that the notion of a sufficient set as used in Section 4.2 can be generalized to weighted graphs. We then present an adapted ILP formulation that accommodates for weighted graphs.

4.4.1 Generalization of Sufficient Sets

Let D be a sufficient set in the sense of Section 4.2. For all $u \in V(G)$, it holds that $u \in D$ or there exists some $v \in D$ which dominates u, i.e. $N[u] \subseteq N[v]$. In search of an optimum group S^*, we may then only consider groups in D, as we are guaranteed to find an optimum group $S^* \subseteq D$ [10]. Consider the following notion of domination, which can be applied to weighted graphs:

$$u \preceq v :\Leftrightarrow \text{dist}(u, v) \leq \omega(G) \wedge \text{dist}(x, v) \leq \text{dist}(x, u)\ \forall x \in V(G) \setminus \{u, v\} \tag{4.10}$$

where $\omega(G) := \min_{e \in E(G)} \omega(e)$ is the smallest edge weight in G. This new notion of dominance is equivalent to the original notion for unweighted graphs as shown in Lemma 1.

Lemma 1. *For unweighted graphs, which we may consider as weighted graphs with unit weights, it holds that*

$$\forall u, v \in V(G) : N[u] \subseteq N[v] \Leftrightarrow u \preceq v$$

Proof. Let G be a unit-weighted graph, so $\omega(G) = 1$. The claim trivially holds for $u = v$, so further assume $u \neq v$. For $(\Rightarrow)$, assume $N[u] \subseteq N[v]$. Then clearly, $\text{dist}(u, v) \leq 1$ since $u \in N[u]$ and thus $u \in N[v]$. Let $x \in V(G) \setminus \{u, v\}$. Consider a shortest path $P = x \ldots u'u$ from x to u. Since $x \neq u$, P has length at least 1. If x and u are adjacent, then we identify $x = u'$. Note that $u' \in N[u] \subseteq N[v]$, so either $u' = v$ and there is a x-v path $P' = x \ldots v$ or $u' \in N(v)$ and there is a x-v path $P' = x \ldots u'v$. Either way, it is easy to see that P' is no longer than P and therefore $\text{dist}(x, v) \leq \text{dist}(x, u)$. For $(\Leftarrow)$, assume $u \preceq v$. Let $x \in N[u]$. If $x = u$, then $\text{dist}(x, v) = \text{dist}(u, v) \leq \omega(G) = 1$ and therefore $x \in N[v]$. If $x = v$, then clearly $x \in N[v]$. Otherwise, $x \in N[u] \setminus \{u, v\}$, and then $u \preceq v$ implies $\text{dist}(x, v) \leq \text{dist}(x, u) \leq 1$, where the last inequality is implied by $x \in N[u]$. From $\text{dist}(x, v) \leq 1$, it directly follows that $x \in N[v]$. Due to the arbitrary choice of x we get $N[u] \subseteq N[v]$, as desired. □

This result naturally raises the question whether a sufficient set D can be constructed for weighted graphs with this generalized notion of domination with the same desirable property $S^* \subseteq D$ for some optimum group S^*. This question can be answered affirmatively, as asserted by Theorem 2. Note that even though (4.10) is equivalent to the previously considered notion of domination for unweighted graphs, it is very restrictive for weighted graphs with heterogeneous weights. We later illustrate how this restriction can be weakened and use the aforementioned definition to simplify the following exposition.

Theorem 2. *Let G be a weighted graph, let $D \subseteq V(G)$ be a sufficient set with respect to the notion $\preceq$, i.e. for all $u \in V(G)$ it holds that $u \in D$ or there exists some $v \in D$ with $u \preceq v$. Then, for all $S \subseteq V(G)$ with $|S| = k \leq |D|$, there is a set $\widetilde{S} \subseteq D$ with $f(\widetilde{S}) \leq f(S)$.*

Proof. Let G be a weighted graph and let $D \subseteq V(G)$ be a sufficient set w.r.t. the notion $\preceq$. Let $S \subseteq V(G)$ with $|S| = k \leq |D|$. We prove the claim by showing that each $u \in S \setminus D$ can be replaced by some vertex in D without increasing group farness. Clearly, if $S \subseteq D$, we are done. Otherwise there is a vertex $u \in S \setminus D$ and by construction therefore a vertex $v \in D$ with $u \preceq v$. We now consider a case distinction.

Case 1. Assume $v \notin S$. Consider $S' := (S \setminus \{u\}) \cup \{v\}$, which has one more vertex in D than S. We argue that $f(S') \leq f(S)$ by considering the change of the shortest path distances for $x \in V(G)$. Let $x \in V(G) \setminus \{u, v\}$ be arbitrary. Since u is the only vertex removed, we only need to consider $x \in V(G) \setminus \{u, v\}$, for which the shortest path to S ended in u. For those x, we get $\text{dist}(x, S') \leq \text{dist}(x, v) \leq \text{dist}(x, u) = \text{dist}(x, S)$, since $u \preceq v$. For $x = v$, we get $\text{dist}(x, S') = \text{dist}(v, S') = 0$. So clearly, the shortest path distance for v decreased by at least $\omega(G)$. Therefore, we can even afford to increase the shortest path distance for u by at most that amount. Indeed, because $u \preceq v$, we get $\text{dist}(u, S') \leq \text{dist}(u, v) \leq \omega(G)$, which completes this case.

Case 2. Assume $v \in S$. Consider $S' := (S \setminus \{u\}) \cup \{w\}$ for an arbitrary $w \in D \setminus S$, which has one more vertex in D than S. The claim $f(S') \leq f(S)$ follows analogously to the above case, but we instead argue about the decrease of the shortest path distance for w. □

This result has the delightful consequence that even for weighted graphs, we may find a smaller set $D \subseteq V(G)$ within which we are guaranteed to find some optimum solution S^*. As mentioned, depending on the graph at hand, $\text{dist}(u, v) \leq \omega(G)$ in (4.10) can be a very strong restriction. However, this condition can be weakened without compromising the claim of Theorem 2. Instead of using $\text{dist}(u, v) \leq \omega(G)$ in (4.10), we may define

$$u \preceq v :\Leftrightarrow \text{dist}(u, v) \leq \omega(v, k) \wedge \text{dist}(x, v) \leq \text{dist}(x, u)\ \forall x \in V(G) \setminus \{u, v\}, \tag{4.11}$$

where $\omega(v, k) := \min\{\omega(v), \omega_k(D)\}$ with $\omega(v) := \min_{\{v,w\} \in E(G)} \omega(\{v, w\})$ being the smallest edge weight at v, and $\omega_k(D) := \max\{\min_{x \in D'} \omega(x) \mid D' \subseteq D, |D'| = k\}$ being the minimum of the k largest $\omega(x)$ out of the $x \in D$. Only note that in Case 2

of the proof, $w \in D$ needs to be selected as to maximize $\omega(w)$. However, since this notion of domination uses D, which depends on the notion of domination, the construction of D requires careful consideration because of the introduction of this circular dependency. A simple way to construct *a* set D, that is sufficient w.r.t. (4.11) consists in sorting $\{v_1, \ldots, v_n\} = V(G)$ s.t. $\omega(v_1) \leq \cdots \leq \omega(v_n)$ and fixing $\{v_{n-k+1}, \ldots, v_n\} \subseteq D$, since this guarantees $\omega_k(D) = \omega(v_{n-k+1})$, resolving the circularity. Altogether, it remains a subject of further research how to efficiently construct small sufficient sets for the weighted case and how effective this restriction of the search space is for real-world weighted graphs.

4.4.2 A Generalized ILP Formulation

Recall the formulation due to Bergamini et al. [7]:

$$\begin{aligned}
\text{minimize} \quad & \sum_{u \in V(G)} \sum_{v \in V(G)} \text{dist}(u, v) \cdot x_{u,v} \quad \text{subject to} \\
& k = \sum_{u \in V(G)} y_u \\
\forall u \in V(G) : & \\
& 1 = \sum_{v \in V(G)} x_{u,v} \\
& x_{u,v} \leq y_v \quad \forall v \in V(G) \\
& y_u \in \{0, 1\} \\
& x_{u,v} \in \{0, 1\} \quad \forall v \in V(G)
\end{aligned}$$

For weighted graphs, we may simply compute the all-pairs shortest path distances and use these in the objective function as before. Therefore, the formulation by Bergamini et al. [7] generalizes to weighted graphs without further complications. However, now consider the formulation due to Staus et al. [10] used in the iterations of IlpInd:

$$\begin{aligned}
\text{minimize} \quad & \sum_{v \in V(G)} \sum_{i \in \{0, \ldots, d(v)\}} i \cdot x_{v,i} \quad \text{subject to} \\
& k = \sum_{v \in V(G)} x_{v,0} \\
\forall v \in V(G) : & \\
& 1 = \sum_{i \in \{0, \ldots, d(v)\}} x_{v,i} \\
& x_{v,i} \leq \sum_{\substack{w \in V(G), \\ \text{dist}(v,w) = i}} x_{w,0} \quad \forall i \in \{0, \ldots, d(v) - 1\} \\
& x_{v,i} \in \{0, 1\} \quad \forall i \in \{0, \ldots, d(v)\}
\end{aligned}$$

Here, we have decision variables $x_{v,0}, \ldots, x_{v,d(v)}$ for $v \in V(G)$, for which $x_{v,i} = 1$ indicates that $v \in V(G)$ is at distance i from the computed optimum group S^*. Naturally, fractional distances cannot be expressed this way. Even for graphs with weights in the positive integers, complications arise when weights are large, as many unnecessary variables may be created. Therefore, we propose an adapted formulation which can be used instead to accommodate for weighted graphs. We define for all $v \in V(G)$:

$$\{\text{dist}(u, v) \mid u \in V(G)\} =: \{d_{v,0}, \ldots, d_{v,r}\} \text{ with } d_{v,0} < \cdots < d_{v,r}.$$

The adapted ILP formulation then reads as follows:

$$\text{minimize} \quad \sum_{v \in V(G)} \sum_{i \in \{0,\ldots,d(v)\}} d_{v,i} \cdot x_{v,i} \quad \text{subject to} \tag{4.12}$$

$$k = \sum_{v \in V(G)} x_{v,0} \tag{4.13}$$

$$\forall v \in V(G):$$

$$1 = \sum_{i \in \{0,\ldots,d(v)\}} x_{v,i} \tag{4.14}$$

$$x_{v,i} \leq \sum_{\substack{w \in V(G), \\ \text{dist}(v,w) = d_{v,i}}} x_{w,0} \quad \forall i \in \{0, \ldots, d(v) - 1\} \tag{4.15}$$

$$x_{v,i} \in \{0, 1\} \quad \forall i \in \{0, \ldots, d(v)\}$$

Here, a new meaning is associated with the decision variables. The assignment $x_{v,i} = 1$ indicates that v is at distance $d_{v,i}$ from the computed optimum group. Since $d_{v,0} = 0$, the same meaning as before is associated with $x_{v,0} = 1$, namely $v \in S^*$. Therefore, constraint (4.13) remains identical. Similarly, constraints (4.14) still indicate that each vertex is at a particular distance from the closest vertex within the optimum solution. Only constraints (4.15) change slightly. Now, the assignment $x_{v,i} = 1$ requires the existence of a vertex $w \in V(G)$ at distance $\text{dist}(v, w) = d_{v,i}$ which is in the solution, i.e. $x_{w,0} = 1$. This adapted ILP can be used in place of the previous ILP in an iterative algorithm like ILPIND or GROVER. However, the sufficiency condition needs to be adapted slightly. For such a formulation to be sufficient, we now require $x_{v,d(v)} = 0$ or $d_{v,d(v)} \geq \text{ecc}(v)$ for all $v \in V(G)$. Again, a heuristic can be used to compute an initial estimate of $\{d(v)\}_{v \in V(G)}$ as outlined in Section 4.3.1. Also, as shown in Section 4.4.1, we can compute a sufficient set $D \subseteq V(G)$ for the weighted case as well. The reduction rules we propose in Section 4.3.2 carry over to the weighted case with the weighted notion of domination and sufficient sets with costs

$$c_{v,i} = d_{v,i} + \sum_{\substack{u \in A, \\ \rho(u) = v}} (\text{dist}(u, v) + d_{v,i})$$

in place of $d_{v,i}$ in (4.12), analogously to 4.3.2. We therefore conclude that our techniques generalize to the weighted case. It remains a subject of further research to evaluate the practical effects of the proposed techniques for the weighted case.

CHAPTER 5

Heuristic Algorithms

Due to the NP-hardness of finding a group $S^* \subseteq V(G)$ of cardinality k with globally maximum group closeness centrality, it is advisable to consider employing heuristic algorithms whenever optimality is not paramount. Chen et al. [8] considered a simple greedy algorithm, which we call GREEDY. Essentially, GREEDY starts with the empty set and iteratively adds the vertex which improves group closeness centrality the most, until the set has k elements. See Algorithm 3 for high-level pseudocode.

Algorithm 3 GREEDY

```
1: function GREEDY(G: GRAPH, k : ℕ)
2:     S_0 ← ∅
3:     for i = 1, ..., k do
4:         pick s ∈ V(G) \ S_{i−1}, s.t. c(S_{i−1} ∪ {s}) is maximized
5:         S_i ← S_{i−1} ∪ {s}
6:     return S_k
```

The supermodularity of group farness $f(S)$ led Chen et al. [8] to believe that group closeness centrality $c(S)$ is submodular. For a monotone, non-negative submodular set function, a well-known result by Nemhauser et al. [23] states that an algorithm like Algorithm 3 yields a $(1 - 1/e)$-approximation, i.e. $c(S_k) \geq (1 - 1/e)c(S^*)$. However, group closeness centrality is not submodular, and this error in the work of Chen et al. [8] was pointed out by Bergamini et al. [7]. An approximation claim can therefore not be supported by this proof strategy.

In light of these facts, it is unclear if GREEDY approximates GCCM up to a constant factor. After all, Bergamini et al. [7] observe very high *empirical* approximation factors, never lower than 0.97. One might therefore still hope to show that GREEDY approximates GCCM, perhaps by using a different result to the one presented by Nemhauser et al. [23]. However, in Section 5.1, we show that we cannot hope for such a result by proving that

GREEDY may return sets with group closeness centrality worse than that of the global optimum by any arbitrary factor. To our knowledge, we are the first to present such an argument. In light of this result, other heuristic algorithms which give approximation guarantees or exact algorithms have to be considered for applications in which solution quality is of great importance. Since it was unclear if GCCM can be approximated at all after the approximation claim in Chen et al. [8] was shown not to hold, renewed interest was directed towards the topic. Angriman et al. [11] settled the approximability of GCCM by using an argument due to Arya et al. [25] for the k-median problem. We discuss their work in Section 5.2. We also present techniques to speed up this approximation algorithm. Note that Angriman et al. [11] apply the local search algorithm to the results of several heuristic algorithms. We consider the GREEDY-LS-C algorithm of [11], which performs local search on the result of GREEDY/GREEDY++. Our results also apply to the other presented variant GS-LS-C, which performs local search on the solution of a heuristic algorithm called GROWSHRINK [9].

5.1 GREEDY Does Not Approximate

We now argue that the simple greedy algorithm GREEDY as presented by Chen et al. [8] does not approximate GCCM (and therefore, neither does GREEDY++). Bergamini et al. [7] already showed that the approximation claim by Chen et al. [8] does not hold. To our knowledge, it has however not been shown that GREEDY does not yield any kind of approximation. We now show that this is the case, i.e. GREEDY might produce solutions with objective values which are worse than the optimum by any arbitrary factor.

Consider a graph G_r with some parameter $r \in \mathbb{N}$ of the following form. G_r contains a path P with end vertices e_1 and e_2. P has an odd number of vertices (precisely $2r - 1$) and thus has a central vertex c. Attached to e_1 and e_2 are r^2 leaves (degree 1 vertices) each.

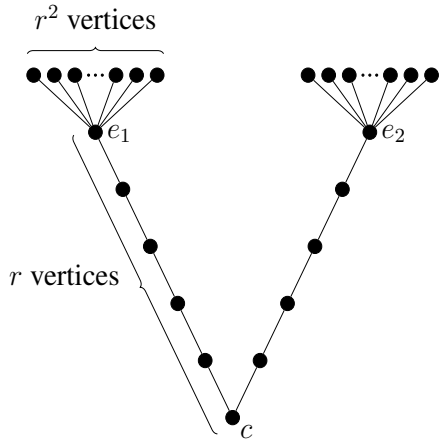


Figure 5.1: Illustration of G_r

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 $f(S) = \sum_{v \in V(G)} \text{dist}(v, S)$ denotes the group farness of S. Consider the following lemma:

Lemma 2.

$$\frac{f(S_r^G)}{f(S_r^*)} \to \infty \text{ for } r \to \infty. \tag{5.1}$$

Proof. First, let us consider the solution S_r^G produced by GREEDY. We have $S_0 = \emptyset$ and, due to symmetry, $S_1 = \{c\}$. Additionally in S_2, we may have either e_1 or e_2, and due to symmetry, we set $S_2 = S_r^G = \{c, e_2\}$ without loss of generality. Now, the r^2 leaves attached to e_1 incur a cost of r to $f(S_r^G)$, each. Therefore, we get $f(S_r^G) \geq r^3$. On the other hand, consider the solution $S_e := \{e_1, e_2\}$. Clearly, $f(S_r^*) \leq f(S_e)$[5]. 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

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

as claimed. □

Clearly, Lemma 2 implies that in the expression $f(S_r^G) = \alpha \cdot f(S_r^*)$, the factor α may get arbitrarily large. Conversely, for group closeness centrality we get

$$c(S_r^G) = \frac{\lambda}{f(S_r^G)} = \frac{\lambda}{\alpha \cdot f(S_r^*)} = \frac{1}{\alpha} c(S_r^*)$$

with $\lambda \in \{1, n(G) - k, n(G)\}$, depending on the definition of group closeness centrality used. Note that we can construct similar graphs for which GREEDY produces arbitrarily poor results for any $k > 2$ by attaching additional *flowers*[6] to c.

This result has important implications for applications in which high quality is important. For those cases, GREEDY cannot give any guarantees, yet high solution quality is observed on real-world graphs [7]. The local search algorithms in Angriman et al. [9] have no known approximation guarantee and hence applications with high quality requirements cannot make any such assumptions. The algorithm proposed by Rajbhandari et al. [12] provides better solutions with more runtime and exact solution after termination. However, as noted before, the runtime required for termination is larger than required by the exact algorithms by Staus et al. [10] and as far as we are aware, their algorithm does not give a constant factor approximation guarantee when stopped before termination. In conclusion, the local search algorithms in Angriman et al. [11] are the only efficient algorithms we are aware of that have been considered and implemented for GCCM which offer approximation guarantees. We consider their work further in the next section.

[5] Indeed, we could argue $S_r^* = S_e$, but we do not need to.

[6] Refers to a connected component of $G_r - c$, which visually resembles a flower.

5.2 An Approximation Algorithm

In their work, Angriman et al. [11] use an argument from Arya et al. [25] for a local search algorithm for the k-median problem. Essentially, this algorithm starts with a set $S \subseteq V(G)$ of k vertices. As long as it finds a swap of a pair of vertices $s \in S$, $o \notin S$ that improves the objective function, it performs that swap, i.e. it sets $S := (S \setminus \{s\}) \cup \{o\}$. Arya et al. [25] show that after this local search procedure terminates, the final set S has at most 5 times the cost of a globally optimum set S^*. This result directly translates to GCCM, as argued by Angriman et al. [11]. Therefore, the set S^* produced by this procedure has at most 5 times the group farness of the globally optimum set, or at least $\frac{1}{5}$ the group closeness centrality of the optimum. In Section 5.2.1, we give the proof of the approximation claim due to Arya et al. [25], adapted to GCCM. In Section 5.2.2, we subsequently argue that this proof allows for a careful restriction of the search space without compromising the approximation result.

5.2.1 Proof of Correctness

Since Arya et al. [25] proved this claim for the more general k-median problem, we think it can be helpful to revisit their proof[7], adapted to our special case. First consider the following Lemma.

Lemma 3. *Let $A = \{a_1, \dots, a_n\}$ be a set and let $\{p_i\}_{1 \le i \le k}$ be a partition of A. Then there exists a bijection $\pi : A \to A$ that satisfies:*

$$\text{For all } i \in \{1, \dots, k\}, \text{ we have } |p_i| \le \tfrac{1}{2}|A| \Rightarrow \pi(p_i) \cap p_i = \emptyset. \tag{5.2}$$

Proof. To see why such a bijection exists, first assume that there is a *big* partition p_i with $m := |p_i| > \frac{1}{2}|A|$ for some $i \in \{1, \dots, k\}$, say $p_1 = \{a_1, \dots, a_m\}$. Clearly, at most one such partition can exist. We construct π by mapping the elements in $A \setminus p_1$ to $\{a_1, \dots, a_{n-m}\} \subseteq p_1$ bijectively. Additionally, $\{a_1, \dots, a_{n-m}\} \subseteq p_1$ are mapped to $A \setminus p_1$ bijectively, and the remaining elements in p_1 are mapped onto themselves. Now we consider the case that no big partition p_i with $|p_i| > \frac{1}{2}|A|$ exists. We assume $|p_1| =: m = \max_{1 \le i \le k} |p_i|$ and order the elements of A sequentially by partition, i.e.

$$\underbrace{a_1, \dots, a_m}_{p_1}, \underbrace{a_{m+1}, \dots, a_{m+|p_2|}}_{p_2}, \dots$$

such that $a_1, \dots, a_m \in p_1$, and similarly order the elements in the remaining partitions consecutively within their respective partition. We construct π in two steps. In the first step, we set $\pi(a_i) = a_{i+m}$ for $i = 1, \dots, n-m$. By the choice of m, no element maps to an element within the same partition. Note that no element is yet mapped to p_1 and it therefore remains to define $\pi^{-1}(a_1), \dots, \pi^{-1}(a_m)$. This is achieved by setting $\pi^{-1}(a_i) = a_{n-m+i}$ for $i = 1, \dots, m$. Clearly, $a_{n-m+1}, \dots, a_n$ are not in p_1, as by assumption $2m \le n$. It is therefore easy to verify that π satisfies (5.2). □

[7] Note that there is a more recent version [29]. Here, we closely follow the proof of the original version [25].

With help of Lemma 3, we now show the approximation claim. The proof given here is largely identical to the proof for the k-median problem as presented in Arya et al. [25]. We adapt some notation to make it consistent with notation used for GCCM and elaborate further on some aspects of the proof.

Theorem 3. *The set S produced by the local search algorithm satisfies the inequality $f(S) \leq 5 \cdot f(S^*)$, where S^* denotes a globally optimum solution.*

Proof. First, we introduce some notation, most of which is identical to Arya et al. [25]. We write $S - s + o$ to denote $(S \setminus \{s\}) \cup \{o\}$. We denote a swap $S := S - s + o$ by $\langle s, o \rangle$. In a solution S, every vertex $v \in V(G)$ has a (not necessarily unique) vertex $s \in S$, to which the distance $\text{dist}(v, s)$ is minimized. We say that v is *assigned* to s and if there are several such vertices in S, v is arbitrarily assigned to a single one. For every $s \in S$, we denote by $A_S(s)$ the set of vertices which are assigned to s in the solution S. Similarly, for a globally optimum solution S^*, we denote by $A_{S^*}(s)$ the set of vertices assigned to s in the solution S^*. Clearly, the sets $\{A_S(s)\}_{s \in S}$ and $\{A_{S^*}(s)\}_{s \in S^*}$ represent partitions of $V(G)$. After the local search procedure terminates, we get

$$f(S - s + o) \geq f(S) \text{ for all } s \in S, o \notin S, \tag{5.3}$$

otherwise we would have $f(S - s + o) < f(S)$ for some $s \in S$, $o \notin S$ and the swap $\langle s, o \rangle$ would have been performed. Even more, we can assert that $f(S - s + o) \geq f(S)$ for any $s \in S$, $o \in S^*$. For each vertex $o \in S^*$, we partition the vertices $A_{S^*}(o)$ which are assigned to o in the solution S^* into subsets $p_{o,s} := A_{S^*}(o) \cap A_S(s)$ for $s \in S$. Consider for some $o \in S^*$ a bijection $\pi_o : A_{S^*}(o) \to A_{S^*}(o)$ which satisfies:

$$\text{For all } s \in S, \text{ we have } |p_{o,s}| \leq \tfrac{1}{2}|A_{S^*}(o)| \Rightarrow \pi_o(p_{o,s}) \cap p_{o,s} = \emptyset. \tag{5.4}$$

Such a bijection exists for all $o \in S^*$ as shown in Lemma 3. Since the expression $\pi_o(v)$ is only defined for $v \in A_{S^*}(o)$, the exact bijection can be determined by its argument and we simply write $\pi(v)$ in the rest of the proof. By taking the union over all $o \in S^*$, π can naturally be understood as a bijection on $V(G)$.

Like Arya et al. [25], we say that a node $o \in S^*$ is *captured* by a node $s \in S$ if more than half the vertices assigned to o are assigned to s, i.e. $|A_{S^*}(o) \cap A_S(s)| > \frac{1}{2}|A_{S^*}(o)|$. Note that node $o \in S^*$ is captured by at most one $s \in S$, while a node $s \in S^*$ may capture none, one or several $o \in S^*$. Additionally, we call a node $s \in S$ *bad* if it captures one or more nodes in S^*, and *good* otherwise. To prove the claim, the idea is to consider k hypothetical swaps and bound the increase in farness due to these swaps.

Whenever a bad node $s \in S$ captures exactly one node $o \in S^*$, we consider the swap $\langle s, o \rangle$. Say l nodes in S (and thus l nodes in S^*) are not considered in such swaps. The l nodes in S therefore capture either none (thus, these are *good* nodes) or at least two nodes in S^*, each. Since each $o \in S^*$ is captured by at most one $s \in S$, there must be at least $l/2$ *good* nodes in S. Having considered $k - l$ swaps already, we now consider the remaining l

swaps. For each of the remaining l nodes in S^*, a swap with a good node is considered s.t. each good node is considered in at most two swaps. As there are at least $l/2$ good nodes, it is easy to see that this can be done. Note that the k swaps satisfy the following properties.

(i) Each $o \in S^*$ is swapped-in exactly once.

(ii) Each $s \in S$ is swapped-out at most twice.

(iii) If the swap $\langle s, o\rangle$ is considered, then s does not capture any $o' \neq o$.

We now analyze these k hypothetical swaps by considering an arbitrary swap $\langle s, o\rangle$ out of the k considered swaps. To show the claim, the idea is to upper bound the increase in farness due to this swap by upper bounding the shortest path distances to $S - s + o$. Clearly, it suffices to only upper bound the increase in farness for nodes in $A_S(s) \cup A_{S^*}(o)$ as the shortest path distances for all other nodes can only decrease. For $v \in A_{S^*}(o)$, we have $\text{dist}(v, S - s + o) \leq \text{dist}(v, o) = \text{dist}(v, S^*)$. Consider a node $v' \in A_S(s) \cap A_{S^*}(o')$ for $o \neq o' \in S^*$. Since by (iii) the node s does not capture o', we have $|A_S(s) \cap A_{S^*}(o')| \leq \frac{1}{2}|A_{S^*}(o')|$ and therefore by the choice of π we get $\pi(v') \notin A_S(s)$. Let $\pi(v') \in A_S(s')$, $s \neq s' \in S$. We get

$$\begin{aligned}\text{dist}(v', S - s + o) &\leq \text{dist}(v', s') \\ &\leq \text{dist}(v', o') + \text{dist}(o', \pi(v')) + \text{dist}(\pi(v'), s') \\ &\leq \text{dist}(v', S^*) + \text{dist}(\pi(v'), S^*) + \text{dist}(\pi(v'), S),\end{aligned} \tag{5.5}$$

since $s' \in S - s + o$ and due to the triangle inequality. Furthermore, due to (5.3) we have $f(S - s + o) - f(S) \geq 0$ and hence for the increase in farness due to the swap $\langle s, o\rangle$ we get

$$\sum_{v \in A_{S^*}(o)} \underbrace{(\text{dist}(v, S^*)}_{\geq\, \text{dist}(v, S-s+o)} - \text{dist}(v, S)) \tag{5.6}$$

$$+ \sum_{v \in A_S(s) \setminus A_{S^*}(o)} (\underbrace{\text{dist}(v, S^*) + \text{dist}(\pi(v), S^*) + \text{dist}(\pi(v), S)}_{\geq\, \text{dist}(v, S-s+o) \text{ by (5.5)}} - \text{dist}(v, S)) \geq 0. \tag{5.7}$$

Since by (i) each $o \in S^*$ is swapped-in exactly once, (5.6) sums to exactly $f(S^*) - f(S)$ over all k considered swaps.

For the second sum, recall that by (ii), each $s \in S$ is swapped-out at most twice. Additionally, for all $v \in V(G)$ we get $\text{dist}(v, S) \leq \text{dist}(v, S^*) + \text{dist}(\pi(v), S^*) + \text{dist}(\pi(v), S)$ using the triangle inequality. Therefore, the second sum added over all k considered swaps is at most $2\sum_{v \in V(G)}(\text{dist}(v, S^*) + \text{dist}(\pi(v), S^*) + \text{dist}(\pi(v), S) - \text{dist}(v, S))$. Since π is bijective on $V(G)$, it holds that $\sum_{v \in V(G)} \text{dist}(v, S^*) = \sum_{v \in V(G)} \text{dist}(\pi(v), S^*) = f(S^*)$ and $\sum_{v \in V(G)} \text{dist}(\pi(v), S) - \text{dist}(v, S) = 0$. Hence we get

$$2 \sum_{v \in V(G)} (\text{dist}(v, S^*) + \text{dist}(\pi(v), S^*) + \text{dist}(\pi(v), S) - \text{dist}(v, S)) = 4f(S^*).$$

Combining the two substitutions, we get $f(S^*) - f(S) + 4f(S^*) \geq 0$ and by rearrangement, the claim follows. □

5.2.2 A Pruned Local Search Algorithm

It should be noted that in the proof of Theorem 3, S^* can be *any* globally optimum group. Crucially, we may therefore restrict the local search to a subset of $V(G)$ which is guaranteed to contain some optimum solution S^*. Doing so, the important inequality $f(S - s + o) \geq f(S)$ for any $s \in S$, $o \in S^*$ clearly still holds after the local search procedure terminates. From an algorithm engineering perspective, this gives rise to an interesting opportunity to reduce the runtime of the local search algorithm: Given a set $D \subseteq V(G)$ which contains a globally optimum solution, we may restrict the local search to this set D without compromising the approximation guarantee. Conveniently, the construction of such a set D has been considered by Staus et al. [10] for the unweighted case and subsequently by us in Section 4.4.1, generalized to the weighted case. We therefore propose the use of the same method here to restrict the search space of the local search algorithm and thus to reduce its runtime. Note that in their work, Angriman et al. [11] already proposed a simple restriction of the search space by not considering swaps with degree 1 vertices. However, while their approach works for unweighted graphs, it is invalid for the weighted case. For a simple example of a failure of this reduction rule for a weighted graph, see Figure A.1 in the appendix. Refer to Algorithm 4 for high-level pseudocode of the pruned local search algorithm.

Algorithm 4 LS-PRUNED

```
1: function LS-PRUNED(G: GRAPH, k : ℕ, S : initial solution)
2:     D ← COMPUTESUFFICIENTSET(G)
3:     while ∃ swap⟨s, o⟩, s ∈ S, o ∈ D s.t. SWAPADMISSIBLE(⟨s, o⟩) do
4:         S ← S − s + o
5:     return S
6:
7: function SWAPADMISSIBLE(⟨s, o⟩ : swap of s ∈ S, o ∈ D)
8:     return f(S − s + o) ≤ (1 − ϵ/Q) f(S)
```

Like the authors of [25], in the proof of Theorem 3, we made the assumption that no swap improves the objective function value, i.e. $f(S - s + o) \geq f(S)$ for all $s \in S$, $o \notin S$ after termination. To ensure polynomial runtime, Arya et al. [25] propose an alternative stopping condition as shown in Algorithm 4, by which we obtain a $5/(1-\epsilon)$-approximation algorithm, i.e. $f(S) \leq \frac{5}{1-\epsilon} f(S^*)$ after the local search procedure. The constant Q denotes the number of possible swaps, i.e. $Q = k(n - k)$ for our problem [11]. We usually refer to the algorithm as a 5-approximation algorithm (or $\frac{1}{5}$-approximation for group closeness centrality), which is true for the local search algorithm that performs swaps until no swap improves the objective. However, note that in implementations like the one due to Angriman et al. [11] in NetworKit [30], usually the modified stopping condition in Algorithm 4 is used to obtain a polynomial runtime guarantee.

CHAPTER 6

Experimental Evaluation

We now provide experimental evaluation of our techniques as proposed in Section 4.3 and Section 5.2.2. Our experiments were performed on an Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz with 16/32 cores/threads, 92 GiB RAM, running Ubuntu 24.04.3. We use Java OpenJDK 21.0.8 for the implementation by Staus et al. [10] and C++20 with CMake 4.1.2 and GCC version 15.1.0 for our implementations. For both the implementation of ILPIND due to Staus et al. [10] and our implementation, we use Gurobi 12.0.1 (https://www.gurobi.com)[8] to solve the ILPs. The exact case is considered in Section 6.1, where we analyze the effect of our techniques as presented in Section 4.3 and compare our results to the state-of-the-art exact algorithm by Staus et al. [10]. The heuristic case is considered in Section 6.2, in which we consider an adapted variant of a local search algorithm proposed by Angriman et al. [11]. As presented in Section 5.2.2, we essentially add a pruning step to restrict the search space of the algorithm while retaining its approximation guarantee. We evaluate the practical effect of this method by benchmarking the runtime of the original and modified local search algorithm. In addition, we analyze the effect on the empirical solution quality.

6.1 Exact Algorithms

Like Staus et al. [10], we perform our experiments on a single thread. As instances for our experiments, we use graphs from the experiments by Staus et al. [10] with the same values of $k \in \{1, \ldots, 20\}$ they used. The graphs were taken from the Network Data Repository [31]. Staus et al. [10] used some graphs which we were unable to obtain, in total we were able to obtain 27 out of the 30 graphs used by Staus et al. [10]. For each problem instance, i.e. some graph together with a value of $k \in \{1, \ldots, 20\}$, each algorithm was given 10 minutes for completion.

[8] Originally, the implementation by Staus et al. [10] used an older version of Gurobi and Java OpenJDK. We modified its dependencies to use the aforementioned newer versions.

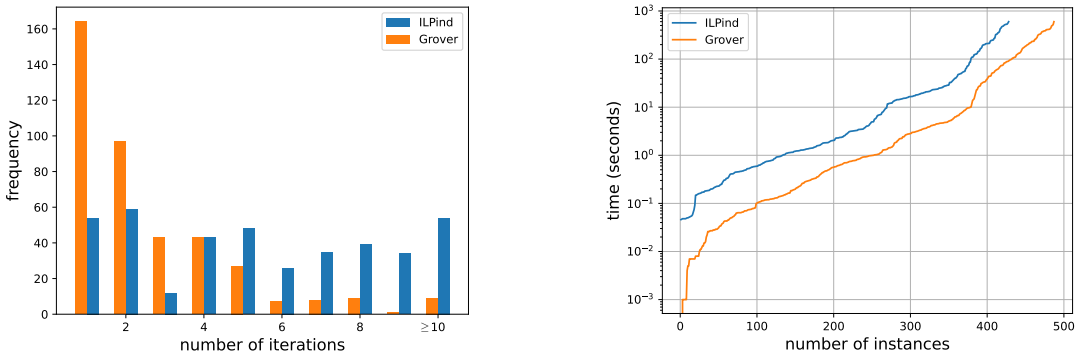
Instance			Mean Runtime (seconds)		Mean Speedup	Instances Solved	
name	$\|V\|$	$\|E\|$	ILPIND	GROVER	GROVER	ILPIND	GROVER
contiguous-usa	49	107	0.05	**0.01**	**3.76**	**20**	**20**
arenas-jazz	198	2742	0.18	**0.04**	**4.88**	**20**	**20**
ca-netscience	379	914	0.23	**0.07**	**3.29**	**20**	**20**
robot24c1_mat5	404	14260	0.75	**0.36**	**3.21**	**20**	**20**
reptilia-tortoise-network-fi	496	984	1.55	**0.50**	**3.11**	**20**	**20**
econ-beause	507	39427	0.49	**0.07**	**6.55**	**20**	**20**
bio-diseasome	516	1188	0.48	**0.13**	**3.55**	**20**	**20**
soc-wiki-Vote	889	2914	1.50	**0.17**	**8.26**	**20**	**20**
ca-CSphd	1025	1043	2.98	**0.35**	**7.58**	**20**	**20**
econ-mahindas	1258	7513	2.74	**0.65**	**4.07**	**20**	**20**
bio-yeast	1458	1948	8.83	**2.50**	**4.39**	**20**	**20**
comsol	1500	48119	1.44	**1.30**	**1.41**	**20**	**20**
heart2	2339	340229	1.58	**0.88**	**1.67**	**20**	**20**
econ-orani678	2529	86768	3.34	**0.87**	**3.92**	**20**	**20**
inf-openflights	2905	15645	15.62	**2.79**	**6.07**	**20**	**20**
ca-GrQc	4158	13422	229.70	**77.17**	**3.26**	**20**	**20**
inf-power	4941	6594	186.40	**58.77**	**3.08**	16	**19**
ca-Erdos992	4991	7428	43.00	**13.55**	**3.40**	**20**	**20**
soc-advogato	5054	39374	24.53	**5.12**	**4.73**	**20**	**20**
bio-dmela	7393	25569	538.91	**140.98**	**4.04**	2	**14**
ia-escorts-dynamic	10106	39016	479.59	**74.09**	**8.07**	10	**20**
ca-HepPh	11204	117619	—	—	—	0	**7**
soc-anybeat	12645	49132	33.40	**6.45**	**4.82**	**20**	**20**
econ-poli-large	15575	17468	20.62	**3.94**	**5.13**	**20**	**20**
ca-AstroPh	17903	196972	—	—	—	0	**1**
ca-CondMat	21363	91286	—	—	—	0	**6**
ca-cit-HepTh	22721	2444642	—	—	—	0	**19**

Table 6.1: Results of our experiments. For each graph, we run both algorithms with $k = 1, \ldots, 20$ with a time limit of 10 minutes for each instance (G, k). For a fair comparison of the runtime and speedup, $k = 1$, for which GROVER simply runs GREEDY++, is excluded from the means. For the *Mean Runtime (seconds)* columns, the *arithmetic mean* runtime over the instances which both algorithms solved is displayed, the symbol — denotes that this intersection is empty. For the *Mean Speedup* column, the *geometric mean* of the speedups of GROVER over ILPIND for instances solved by both algorithms is shown, hence the runtime ratio is not necessarily identical to the speedup, for more detail refer to Section A.2 in the appendix. For each cell, best results are bolded.

Table 6.1 summarizes our results. The two rightmost columns show the number of instances solved in the given amount of time for each graph by the respective algorithm and show that our algorithm GROVER[9] is able to solve more instances. The 4th and 5th

[9]For our source code, see https://github.com/JakobTernes/Grover.

columns show the arithmetic mean runtimes of both algorithms and reveal that GROVER consistently outperforms ILPIND. The 6th column reports the geometric mean speedup of GROVER over ILPIND. Our results show that GROVER achieves significant speedups over the state-of-the-art algorithm ILPIND. Crucially, a substantial number of large instances which cannot be solved by ILPIND within the given time can be solved by GROVER, showing its superior scalability. Overall, GROVER achieves a more than 4-fold geometric mean speedup over ILPIND.



(a) Plot showing how often (y-axis) ILPIND and GROVER needed x number of iterations.

(b) Number of instances solved (x-axis) within a given time (logarithmic y-axis).

Figure 6.1: Plots of data from our experiments, showing lower iteration count and superior performance of our algorithm GROVER. Overall, GROVER takes noticeably less iterations than ILPIND and takes significantly less time to solve the same number of instances.

Figure 6.1 contains plots showing the effect of our techniques. As intended, GROVER needs less iterations than ILPIND as can be seen in Subfigure 6.1a. Together with our reduction technique, this allows GROVER to solve considerably more instances than ILPIND in the same amount of time as shown in Subfigure 6.1b. For example, for 400 instances GROVER terminated in less than 39 seconds, while ILPIND required around 210 seconds for the same number of instances. Our experiments also show, as perhaps expected, that the runtime overhead in GROVER due to computing the heuristic solution with GS-LS-C is largely negligible. Out of the total runtime over all instances with $k \geq 2$, only 3.2% of the runtime is spent computing the heuristic solution. For comparison, 93.7% of the runtime is spent on solving ILPs. The majority of the remaining time is spent on computing shortest path distances used to construct the ILPs.

Figure 6.2 shows runtimes for example graphs for varying values of k. The runtime for $k = 1$ is visibly small for GROVER since GREEDY++ is run in this case. Occasional spikes in the runtime of GROVER can be observed due to varying quality of the initial estimate of $\{d(v)\}_{v \in V(G)}$. For some instances, a slight trend of convergence of the runtimes of GROVER and ILPIND for larger values of k can be observed. Presumably, this is explained by the estimate $d(v) = 2$ for all $v \in V(G)$ used in ILPIND being closer to sufficient val-

ues for larger values of k. Interestingly, runtime seems to remain constant or even tend to decrease with larger values of k, despite the simple brute-force approach for GCCM, that enumerates all solutions and returns the one with optimum objective value, scaling with $n^{k+O(1)}$.

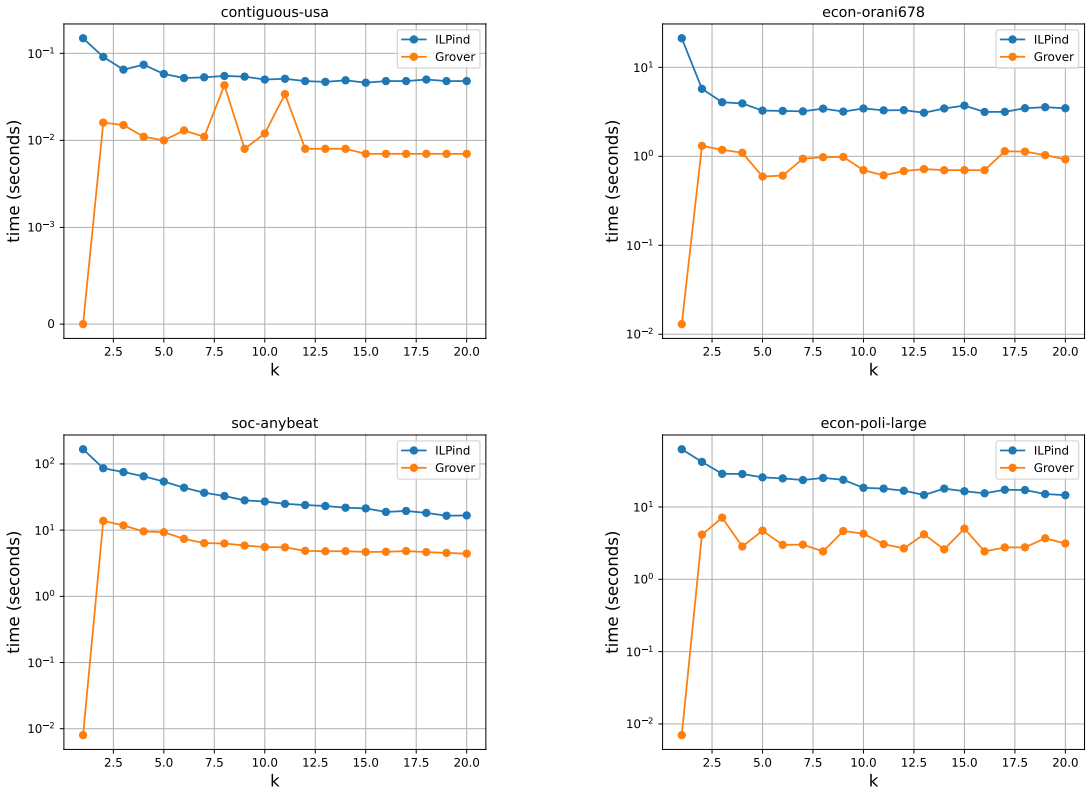


Figure 6.2: Plots of runtimes (logarithmic y-axis) for four example graphs from the experiments for varying values of k (x-axis).

6.2 Heuristic Algorithms

To evaluate the effect of our pruning technique, we compare our modified local search algorithm to the local search algorithm of Angriman et al. [11] as implemented in the open source network analysis toolkit NetworKit [30]. For our experiments, we use the same unweighted graphs as in Section 6.1. Recall that the local search algorithm takes a group, which it refines, as input. Since the approximation guarantee does not depend on the input group, many variants of the local search algorithm are conceivable. Angriman et al. [11] consider two variants, one which takes the result of GREEDY/GREEDY++ as input (GREEDY-LS-C) and one which takes the result of a heuristic called GROWSHRINK as input (GS-LS-C). We performed our experiments for both

variants and observed very similar speedups. Here, we present the results with GREEDY-LS-C and write LS for short to refer to this algorithm, and LS-PRUNED to refer to our pruned variant.

Instance	k = 5		k = 10		k = 50		k = 100	
	LS	LS-PRUNED	LS	LS-PRUNED	LS	LS-PRUNED	LS	LS-PRUNED
arenas-jazz	0.002	**0.001**	**0.002**	**0.002**	0.008	**0.006**	0.011	**0.009**
ca-netscience	0.002	**0.001**	0.004	**0.002**	0.013	**0.004**	0.022	**0.006**
robot24c1_mat5	0.014	**0.013**	0.021	**0.019**	0.035	**0.032**	0.055	**0.051**
reptilia-tortoise-network-fi	0.018	**0.013**	0.039	**0.028**	0.089	**0.062**	0.11	**0.075**
econ-beause	0.01	**0.008**	0.016	**0.01**	0.066	**0.034**	0.125	**0.064**
bio-diseasome	0.008	**0.004**	0.021	**0.009**	0.026	**0.01**	0.035	**0.013**
soc-wiki-Vote	0.013	**0.011**	0.024	**0.02**	0.052	**0.048**	0.096	**0.088**
ca-CSphd	**0.027**	**0.027**	0.016	**0.014**	0.043	**0.04**	0.086	**0.081**
econ-mahindas	0.055	**0.05**	0.097	**0.094**	**0.344**	0.399	0.324	**0.317**
bio-yeast	0.023	**0.019**	0.033	**0.027**	0.302	**0.292**	0.273	**0.248**
comsol	0.064	**0.027**	0.045	**0.017**	0.208	**0.06**	0.396	**0.112**
heart2	0.191	**0.161**	0.198	**0.154**	0.765	**0.352**	1.443	**0.595**
econ-orani678	**0.053**	0.076	**0.095**	0.12	**0.439**	0.468	**0.833**	0.884
inf-openflights	0.068	**0.049**	0.284	**0.188**	0.824	**0.503**	1.561	**0.81**
ca-GrQc	0.378	**0.212**	0.295	**0.152**	2.873	**1.297**	5.005	**2.088**
inf-power	1.191	**0.976**	1.246	**1.198**	**9.862**	9.939	24.037	**22.298**
ca-Erdos992	0.108	**0.085**	0.119	**0.097**	1.004	**0.771**	1.661	**1.2**
soc-advogato	0.155	**0.137**	0.286	**0.247**	1.05	**0.866**	1.912	**1.636**
bio-dmela	0.47	**0.469**	0.929	**0.923**	**2.424**	2.432	9.067	**8.9**
ia-escorts-dynamic	**0.752**	0.757	**1.047**	1.116	**4.7**	5.201	**10.647**	11.698
ca-HepPh	2.434	**1.435**	2.53	**1.43**	33.984	**17.336**	34.57	**15.819**
soc-anybeat	0.458	**0.339**	0.831	**0.586**	4.036	**2.777**	8.029	**5.504**
econ-poli-large	0.419	**0.322**	0.639	**0.487**	3.265	**2.335**	7.793	**5.864**
ca-AstroPh	6.286	**3.998**	7.405	**4.239**	52.882	**30.644**	173.315	**83.112**
ca-CondMat	5.509	**2.676**	7.494	**3.357**	54.092	**21.798**	98.347	**41.802**
ca-cit-HepTh	6.199	**4.971**	11.016	**7.902**	53.646	**35.431**	129.401	**77.326**
Speedup over LS	**× 1.34**		**× 1.39**		**× 1.49**		**× 1.56**	

Table 6.2: Runtimes of the local search algorithm (LS) by Angriman et al. [11] as implemented in NetworKit [30] and the pruned local search algorithm (LS-PRUNED) in seconds. We use the same graphs as in Section 6.1 (except for those with $n \leq 100$) and values of $k \in \{5, 10, 50, 100\}$ as in the benchmarks by Angriman et al. [11]. Due to runtime variance, experiments were conducted ten times. The arithmetic mean of those times is reported in the table. The last row reports the geometric mean speedup of LS-PRUNED over LS for the given value of k. Evidently, the speedup increases with larger k.

To implement LS-PRUNED, we modified the implementation of Angriman et al. [11] by adding a pruning step as described in Section 5.2.2. As argued, this modification retains the approximation guarantee while potentially restricting the search space. Table 6.2 shows the execution times for both algorithms in our experiments. The experiments were conducted in a single thread due to the current sequential implementation of the pruning step. We note however, that the pruning is in principle highly parallelizable. Also note that the

measured times include the pruning step for LS-PRUNED, they do not however include the time required by either algorithm to run GREEDY++ before the local search. Our results show that the pruning is highly effective, speedups are achieved for the vast majority of instances. For 39 out of the 104 reported instances (37.5% of instances), LS-PRUNED achieved a speedup by at least a factor of 1.5, and for 23 instances (22% of instances), even a speedup by at least a factor of 2 was achieved. The objective function values achieved by both algorithms are remarkably close with identical scores in over 93% of the instances and only slight differences in the remaining cases, for more detail refer to Section A.3 in the appendix. As noted briefly in Section 4.4.1, it remains a subject of further study to experimentally evaluate the effectiveness of our pruning technique applied to weighted graphs.

CHAPTER 7

Discussion

We presented current state-of-the-art approaches for group closeness centrality maximization (GCCM), both for the exact and the approximation case. For the presented algorithms, we proposed new techniques to reduce their runtime. In the exact case, our techniques yield a geometric mean 4-fold speedup. Additionally, we generalized the methods of the state-of-the-art exact algorithm and our new techniques to weighted graphs, thereby broadening the applicability of the resulting exact algorithm. For the approximation case, we showed that our adaptation of the considered algorithm preserves its approximation guarantee while at the same time reducing its runtime in experiments. While Bergamini et al. [7] pointed out that the approximation proof of the simple greedy algorithm as outlined by Chen et al. [8] is invalid, we additionally showed the even stronger result that the approximation ratio can, in fact, be arbitrarily bad.

Future work could investigate if the size of the set $D \subseteq V(G)$, which is guaranteed to contain a globally optimum solution S^*, can be reduced in size. A further reduction of the size of D would presumably yield speedups for both our exact algorithm GROVER and our pruned local search algorithm. Another interesting question is whether the proposed techniques could be adapted and generalized as to be applicable to more general problems like the k-median problem. For instance, a similar search space reduction might be possible, or perhaps the iterative technique proposed by Staus et al. [10] together with our iteration reduction and data reduction techniques could be used to obtain an exact algorithm for more general problems. As previously mentioned, it also remains to evaluate the effectiveness of the restriction of the search space (both for the exact and approximation case) and the adapted ILP from Section 4.4 for real-world weighted graphs, as our experiments treated the unweighted case.

Bibliography

[1] Jeffrey Johnson, Steve Borgatti, and Martin Everett. *Analyzing Social Networks*. Jan. 2013. ISBN: 978-1446247419.

[2] Martijn P Van den Heuvel and Olaf Sporns. "Network hubs in the human brain." In: *Trends in cognitive sciences* 17.12 (2013), pp. 683–696.

[3] Eric Chea and Dennis R. Livesay. "How accurate and statistically robust are catalytic site predictions based on closeness centrality?" In: *BMC Bioinform.* 8 (2007).

[4] Isaiah G. Adebayo and Yanxia Sun. In: *International Journal of Emerging Electric Power Systems* 21.3 (2020), p. 20200013.

[5] Dirk Koschützki et al. "Centrality Indices." In: *Network Analysis: Methodological Foundations*. Ed. by Ulrik Brandes and Thomas Erlebach. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 16–61. ISBN: 978-3-540-31955-9.

[6] Martin G Everett and Stephen P Borgatti. "The centrality of groups and classes." In: *The Journal of mathematical sociology* 23.3 (1999), pp. 181–201.

[7] Elisabetta Bergamini, Tanya Gonser, and Henning Meyerhenke. "Scaling up Group Closeness Maximization." In: *CoRR* abs/1710.01144 (2019).

[8] Chen Chen, Wei Wang, and Xiaoyang Wang. "Efficient Maximum Closeness Centrality Group Identification." In: *Databases Theory and Applications - 27th Australasian Database Conference, ADC 2016, Sydney, NSW, Australia, September 28-29, 2016, Proceedings*. Ed. by Muhammad Aamir Cheema, Wenjie Zhang, and Lijun Chang. Vol. 9877. Lecture Notes in Computer Science. Springer, 2016, pp. 43–55.

[9] Eugenio Angriman, Alexander van der Grinten, and Henning Meyerhenke. "Local Search for Group Closeness Maximization on Big Graphs." In: *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*. Ed. by Chaitanya K. Baru et al. IEEE, 2019, pp. 711–720.

[10] Luca Pascal Staus et al. "Exact Algorithms for Group Closeness Centrality." In: *SIAM Conference on Applied and Computational Discrete Algorithms, ACDA 2023, Seattle, WA, USA, May 31 - June 2, 2023*. Ed. by Jonathan W. Berry et al. SIAM, 2023, pp. 1–12.

[11] Eugenio Angriman et al. “Group-Harmonic and Group-Closeness Maximization - Approximation and Engineering.” In: *Proceedings of the 23rd Symposium on Algorithm Engineering and Experiments, ALENEX 2021, Virtual Conference, January 10-11, 2021*. Ed. by Martin Farach-Colton and Sabine Storandt. SIAM, 2021, pp. 154–168.

[12] Baibhav Rajbhandari et al. “PRESTO: Fast and Effective Group Closeness Maximization.” In: *IEEE Trans. Knowl. Data Eng.* 35.6 (2023), pp. 6209–6223.

[13] Leonid Genrikhovich Khachiyan. “A polynomial algorithm in linear programming.” In: *Doklady Akademii Nauk*. Vol. 244. 5. Russian Academy of Sciences. 1979, pp. 1093–1096.

[14] “A new polynomial-time algorithm for linear programming.” In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. 1984, pp. 302–311.

[15] Richard M. Karp. “Reducibility Among Combinatorial Problems.” In: *Proceedings of a symposium on the Complexity of Computer Computations*. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103.

[16] Alex Bavelas. “A mathematical model for group structures.” In: *Applied Anthropology* 7.3 (1948), pp. 16–30. ISSN: 00932914.

[17] Leo Katz. “A new status index derived from sociometric analysis.” In: *Psychometrika* 18.1 (1953), pp. 39–43.

[18] Marvin E Shaw. “Group structure and the behavior of individuals in small groups.” In: *The Journal of psychology* 38.1 (1954), pp. 139–149.

[19] Mark Newman. *Networks*. Oxford University Press, Inc., 2018. ISBN: 9780198805090.

[20] Murray A. Beauchamp. “An improved index of centrality.” In: *Behavioral Science* 10.2 (1965), pp. 161–163.

[21] Gert Sabidussi. “The centrality index of a graph.” In: *Psychometrika* 31.4 (1966), pp. 581–603.

[22] Daniel Aloise et al. “NP-hardness of Euclidean sum-of-squares clustering.” In: *Mach. Learn.* 75.2 (2009), pp. 245–248.

[23] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. “An analysis of approximations for maximizing submodular set functions - I.” In: *Math. Program.* 14.1 (1978), pp. 265–294.

[24] Elisabetta Bergamini, Tanya Gonser, and Henning Meyerhenke. “Scaling up Group Closeness Maximization.” In: *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018, New Orleans, LA, USA, January 7-8, 2018*. Ed. by Rasmus Pagh and Suresh Venkatasubramanian. SIAM, 2018, pp. 209–222.

[25] Vijay Arya et al. “Local search heuristic for k-median and facility location problems.” In: *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. 2001, pp. 21–29.

[26] Henning Martin Woydt, Christian Komusiewicz, and Frank Sommer. “SubModST: A Fast Generic Solver for Submodular Maximization with Size Constraints.” In: *32nd Annual European Symposium on Algorithms, ESA 2024, Royal Holloway, London, United Kingdom, September 2-4, 2024*. Ed. by Timothy M. Chan et al. Vol. 308. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 102:1–102:18.

[27] Junzhou Zhao et al. “Measuring and maximizing group closeness centrality over disk-resident graphs.” In: *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*. Ed. by Chin-Wan Chung et al. ACM, 2014, pp. 689–694.

[28] Huan Li et al. “Current Flow Group Closeness Centrality for Complex Networks?” In: *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. Ed. by Ling Liu et al. ACM, 2019, pp. 961–971.

[29] Vijay Arya et al. “Local Search Heuristics for k-Median and Facility Location Problems.” In: *SIAM J. Comput.* 33.3 (2004), pp. 544–562.

[30] Eugenio Angriman et al. “Algorithms for Large-Scale Network Analysis and the NetworKit Toolkit.” In: *Algorithms for Big Data - DFG Priority Program 1736*. Ed. by Hannah Bast et al. Vol. 13201. Lecture Notes in Computer Science. Springer, 2022, pp. 3–20.

[31] Ryan A. Rossi and Nesreen K. Ahmed. “The Network Data Repository with Interactive Graph Analytics and Visualization.” In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, 2015, pp. 4292–4293.

APPENDIX A

Appendix

A.1 Degree 1 Vertices in the Weighted Case

Figure A.1 illustrates that, in weighted graphs, degree 1 vertices may not in general be removed from the search space without affecting the approximation guarantee of the local search algorithm considered in Section 5.2. The numbers annotated to the edges denote edge weights. For this example, the reduction of degree 1 vertices would lead to the three

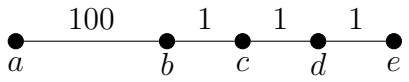


Figure A.1: Example graph that illustrates why degree 1 vertices may not in general be removed from the search space of the local search algorithm.

central vertices $\{b, c, d\}$ remaining in the search space. Suppose $k = 2$. If the initial solution S passed to the local search algorithm does not include a, for instance $S = \{b, c\}$, then the best solution the local search can find is $\{b, d\}$ with group farness $f(\{b, d\}) = 102$. However, the optimum solution $S^* = \{a, c\}$ has group farness $f(S^*) = 4$. Clearly, the example and initial sets can be constructed as to evoke arbitrarily poor solutions returned by the local search algorithm, contradicting the approximation claim. Note that our reduction rule as proposed in Section 4.4.1 correctly identifies that a cannot be reduced, since $\text{dist}(a, b) \leq \omega(G) = 1$ is not satisfied.

A.2 Benchmark Metrics

In our experiments, we frequently take the mean over several values, usually these values represent output data from different runs on the same instance or related instances. For example, in Section 6.1, we take the arithmetic mean of runtimes. As mentioned, the case $k = 1$ is excluded for a fair comparison of the two algorithms, since GROVER is able to solve these instances rather quickly due to simply running GREEDY++. Some instances which were solved by GROVER were not solved by ILPIND. Therefore, for each graph G used in the experiments, we take the subset $I(G) \subseteq \{(G, 2), \ldots, (G, 20)\}$ of maximum size with instances solved in time by both algorithms and take the arithmetic mean runtime for each algorithm. Formally, denote by $R_{\text{ILPIND}}((G, k))$ and $R_{\text{GROVER}}((G, k))$ the runtime of ILPIND and GROVER for instance $(G, k) \in I(G)$, respectively. Then, the arithmetic mean runtimes reported in Table 6.1 for each graph G are obtained as

$$\frac{1}{|I(G)|} \sum_{i \in I(G)} R_{\text{ILPIND}}(i) \qquad \text{and} \qquad \frac{1}{|I(G)|} \sum_{i \in I(G)} R_{\text{GROVER}}(i),$$

for ILPIND and GROVER, respectively. Similarly, the geometric mean speedups are obtained as

$$\left(\prod_{i \in I(G)} \frac{R_{\text{ILPIND}}(i)}{R_{\text{GROVER}}(i)} \right)^{\frac{1}{|I(G)|}}.$$

Similarly, the runtimes reported in Table 6.2 are arithmetic means, but now taken over ten runs for the same instance (G, k) due to high runtime variance observed in preliminary experiments. All algorithms were run until completion. Let $R_{\text{LS}}((G, k))$, $R_{\text{LS-PRUNED}}((G, k))$ be the arithmetic mean runtime over the ten runs for the instance (G, k). The reported speedup $S(k)$ for a fixed value of k, in analogy to the exact case, are obtained by taking the geometric mean of the speedups, i.e.

$$S(k) := \left(\prod_{G \in \mathcal{G}} \frac{R_{\text{LS}}((G, k))}{R_{\text{LS-PRUNED}}((G, k))} \right)^{\frac{1}{|\mathcal{G}|}},$$

where $\mathcal{G}$ denotes the set of all graphs used in the experiments.

A.3 Empirical Local Search Scores

Refer to Table A.1 for the empirical scores, i.e. the farness of the solution computed by LS and LS-PRUNED. Overall, the solution quality is largely identical. The algorithms achieve identical scores in 97 out of the 104 reported instances. For 2 instances, LS achieves a better score while LS-PRUNED achieves a better score for 5 instances.

Instance	k = 5		k = 10		k = 50		k = 100	
	LS	LS-PRUNED	LS	LS-PRUNED	LS	LS-PRUNED	LS	LS-PRUNED
arenas-jazz	**213**	**213**	**194**	**194**	**148**	**148**	**98**	**98**
ca-netscience	**779**	**779**	**641**	**641**	**335**	**335**	**279**	**279**
robot24c1_mat5	**433**	**433**	**398**	**398**	**354**	**354**	**304**	**304**
reptilia-tortoise-network-fi	**1653**	**1653**	**1266**	**1266**	**619**	**619**	444	**442**
econ-beause	**502**	**502**	**497**	**497**	**457**	**457**	**407**	**407**
bio-diseasome	**1320**	**1320**	1073	**1068**	**540**	**540**	**416**	**416**
soc-wiki-Vote	**1707**	**1707**	**1494**	**1494**	**1099**	**1099**	**914**	**914**
ca-CSphd	**4239**	**4239**	**2982**	**2982**	**1567**	**1567**	**1175**	**1175**
econ-mahindas	**2164**	**2164**	**1943**	**1943**	**1619**	**1619**	**1390**	**1390**
bio-yeast	**4875**	**4875**	**4217**	**4217**	**2559**	**2559**	**1968**	**1968**
comsol	**2140**	**2140**	**1646**	**1646**	**1450**	**1450**	**1400**	**1400**
heart2	**2647**	**2647**	**2329**	**2329**	**2289**	**2289**	**2239**	**2239**
econ-orani678	**2697**	**2697**	**2672**	**2672**	**2572**	**2572**	**2472**	**2472**
inf-openflights	**6113**	**6113**	**5523**	**5523**	**4122**	**4122**	**3523**	**3523**
ca-GrQc	**13226**	**13226**	**11887**	**11887**	**8572**	8575	7047	**7046**
inf-power	**34567**	**34567**	**28118**	**28118**	17531	**17525**	**13846**	**13846**
ca-Erdos992	**14294**	**14294**	**12757**	**12757**	**9077**	**9077**	**7410**	**7410**
soc-advogato	**8529**	**8529**	**8032**	**8032**	**6785**	**6785**	**6237**	**6237**
bio-dmela	**18296**	**18296**	**16749**	**16749**	**13524**	**13524**	**12049**	**12049**
ia-escorts-dynamic	**23629**	**23629**	**21873**	**21873**	**17802**	**17802**	**15936**	**15936**
ca-HepPh	**29333**	**29333**	**27279**	**27279**	**21824**	21825	**19462**	**19462**
soc-anybeat	**19121**	**19121**	**17838**	**17838**	**15934**	**15934**	**15137**	**15137**
econ-poli-large	**48013**	**48013**	**43211**	**43211**	**30836**	**30836**	**26025**	**26025**
ca-AstroPh	**43445**	**43445**	**40736**	**40736**	**33546**	**33546**	30132	**30131**
ca-CondMat	**62955**	**62955**	**58294**	**58294**	**48157**	**48157**	**43342**	**43342**
ca-cit-HepTh	**32359**	**32359**	**30724**	**30724**	**27209**	**27209**	**25812**	**25812**

Table A.1: Group farness values from our experiments with the local search algorithm (LS) applied to the result of GREEDY/GREEDY++, which corresponds to GREEDY-LS-C by Angriman et al. [11], and our pruned variant (LS-PRUNED). Out of the 10 runs, the lowest score is reported in the table.

For the local search variant which uses the result of GROWSHRINK as the input group, which corresponds to GS-LS-C in Angriman et al. [11], we observe slightly higher speedups and slightly worse solution quality for the pruned variant. For $k = 5, 10, 50, 100$, we observe mean speedups by a factor of $1.34, 1.41, 1.65, 1.61$, respectively. At the same

time, we observe slightly worse solution quality in the pruned variant as shown in Table A.2. Still, the solution quality is largely identical.

Instance	k = 5		k = 10		k = 50		k = 100	
	LS	LS-PRUNED	LS	LS-PRUNED	LS	LS-PRUNED	LS	LS-PRUNED
arenas-jazz	**213**	**213**	**191**	192	**148**	**148**	**98**	**98**
ca-netscience	**779**	**779**	**636**	**636**	**334**	**334**	**279**	**279**
robot24c1_mat5	**431**	433	**396**	**396**	**354**	**354**	**304**	**304**
reptilia-tortoise-network-fi	**1653**	**1653**	**1266**	**1266**	**615**	617	**437**	**437**
econ-beause	**502**	**502**	**497**	**497**	**457**	**457**	**407**	**407**
bio-diseasome	**1311**	**1311**	**1072**	**1072**	**538**	**538**	**416**	**416**
soc-wiki-Vote	**1707**	**1707**	**1494**	**1494**	**1097**	**1097**	**916**	**916**
ca-CSphd	**4239**	**4239**	**2982**	**2982**	**1567**	**1567**	**1175**	**1175**
econ-mahindas	**2164**	**2164**	**1940**	1941	**1618**	1619	**1392**	1394
bio-yeast	**4875**	**4875**	**4213**	**4213**	**2559**	**2559**	1967	**1966**
comsol	**2140**	**2140**	**1646**	**1646**	**1450**	**1450**	**1400**	**1400**
heart2	**2586**	**2586**	**2329**	**2329**	**2289**	**2289**	**2239**	**2239**
econ-orani678	**2697**	**2697**	**2672**	**2672**	**2571**	**2571**	2474	**2472**
inf-openflights	**6113**	6130	**5523**	**5523**	**4122**	**4122**	**3523**	**3523**
ca-GrQc	**13226**	**13226**	**11877**	**11877**	8539	**8535**	**7039**	**7039**
inf-power	**34567**	**34567**	**28118**	**28118**	17500	**17481**	**13843**	13846
ca-Erdos992	**14294**	**14294**	**12757**	**12757**	**9066**	**9066**	7408	**7407**
soc-advogato	**8529**	**8529**	**8032**	**8032**	**6785**	**6785**	**6234**	**6234**
bio-dmela	**18296**	**18296**	**16749**	**16749**	**13524**	**13524**	12049	**12047**
ia-escorts-dynamic	**23629**	**23629**	**21873**	**21873**	**17798**	**17798**	**15931**	15932
ca-HepPh	**29283**	**29283**	**27154**	27166	**21803**	21818	**19433**	19442
soc-anybeat	**19121**	**19121**	**17838**	**17838**	**15934**	**15934**	**15136**	**15136**
econ-poli-large	**48013**	**48013**	**42880**	**42880**	**30832**	**30832**	**26025**	**26025**
ca-AstroPh	**43445**	**43445**	**40736**	**40736**	33574	**33546**	**30095**	30096
ca-CondMat	**62924**	**62924**	**58294**	**58294**	**48147**	48150	43340	**43337**
ca-cit-HepTh	**32359**	**32359**	**30724**	**30724**	**27200**	**27200**	**25798**	25801

Table A.2: Group farness from our experiments. Here, GROWSHRINK is used to compute the initial solution instead of GREEDY++, i.e. LS now corresponds to GS-LS-C in [11] and LS-PRUNED corresponds to our pruned variant of that algorithm. We observe more instances (23) with differing solution quality, but still identical solution quality for the great majority of instances. We also observe higher absolute differences of up to 28 for ca-AstroPh with $k = 50$.