Karlsruhe Institute of Technology

Core Augmentation for Subgraphs

Bachelor Thesis of

Philipp Christian Loewner

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Prof. Dr. Dorothea Wagner
	Prof. Dr. Peter Sanders
Advisors:	Tanja Hartmann
	Dr. Ignaz Rutter

Time Period: 06.12.2012 – 05.04.2013

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 8th April 2013

Abstract

The k-core of a graph is obtained by iteratively removing vertices of degree less than k. The *coreness* of a vertex is the highest k for which that vertex is in the k-core.
We study the problem of adding to a graph as few edges as possible to lift a given vertex in that graph to the k-core. We give an algorithm that solves the problem efficiently for fixed k and prove that if k is part of the input, then the problem is $\mathcal{NP}$-complete and cannot be approximated within a factor of $(1-\delta)\log(k^2 \cdot n)$ or $(3-\varepsilon)\log n$ for any $\delta, \varepsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$. Furthermore, we show that for every instance, the optimal solution can be efficiently approximated within a factor of $\frac{k^2+k}{2}$ and we give efficient algorithms that approximate the optimal solution within a factor of k or better, if the k-core of the input graph is not empty or if the degree of every vertex in the input graph is smaller than k. Finally, we give heuristic approaches for solving the problem efficiently and discuss their viability depending on the structure of the input graph.

Deutsche Zusammenfassung

Man erhält den k-core eines Graphen, indem man iterativ alle Knoten mit Grad kleiner als k aus ihm entfernt. Die höhste Zahl k für die ein Knoten im k-core ist nennt man die *coreness* dieses Knotens.
Wir untersuchen das Problem, einen bestimmten Knoten in einem Graphen durch Hinzufügen möglichst weniger Kanten in den k-core zu heben. Wir geben einen Algorithmus an, der dieses Problem für festes k in Polynomialzeit löst. Zusätzlich beweisen wir, dass das Problem für nicht-festes k $\mathcal{NP}$-vollständig ist und dass kein effizienter Algorithmus mit einer relativen Gütegarantie von $(1-\delta)\log(k^2 \cdot n)$ oder $(3-\varepsilon)\log n$ für $\delta, \varepsilon > 0$ existiert, es sei denn, dass $\mathcal{P} = \mathcal{NP}$. Wir zeigen, dass sich jede Instanz des Problems mit einer relativen Gütegarantie von $\frac{k^2+k}{2}$ effizient approximieren lässt und geben Algorithmen an, die das Problem mit einer relativen Gütegarantie von k oder besser lösen, wenn der k-core des Eingabegraphen nicht leer ist oder wenn der Grad eines jeden Knotens im Eingabegraphen kleiner ist als k. Schließlich entwickeln wir effiziente Heuristiken und diskutieren, unter welchen Voraussetzungen an die Struktur des Eingabegraphen sie gute Ergebnisse liefern können.

Contents

1. Introduction

In the field of theoretical computer science, graphs are commonly used to model a wide variety of domains. In particular, they have proven useful in the design and analysis of communication networks as well as in the analysis of social networks.
The k-core of a graph can be obtained by iteratively removing vertices of degree less than k. The coreness of a vertex is the highest k for which the vertex is in the k-core. The concept of cores was originally introduced by Seidman in 1983 in order to find a better way to measure the centrality of people in social networks. Today, cores are still relevant in this regard. For example, Christakis et al. successfully use the coreness of individuals in a social network, among other structural parameters, to predict an outbreak of the flu significantly earlier than with current detection methods for contagious outbreaks [CF10]. For this, they use the fact that people who are more central in a social network are typically infected earlier than random individuals.
In the context of communication networks, Feldmann et al. use cores to show that the topology of the file-sharing network Gnutella differs from randomly generated networks [AFG$^+$06].
The core structure of graphs also has applications in other fields. For example, Wuchty et al. use core decomposition to examine the centrality of different proteins in the protein interaction network of yeast [WA05].

In the context of both social and communication networks, the question arises how the connectivity of a network as a whole, or the connectivity of a single node in the network can be improved by changing the core structure of the network. Görke et al. study this question and show that for $k \geq 3$, the problem of lifting a given set of vertices in a graph to the k-core by adding as few edges as possible is $\mathcal{NP}$-complete, $W[2]$-hard with respect to the solution size and the optimal solution is $\mathcal{NP}$-hard to approximate within a factor of $(1-\varepsilon)\log n$ for any $\varepsilon > 0$. They raise the question whether the problem is polynomial-time computable if only a single vertex in the graph must be lifted to the k-core.
In this thesis, we study that question and conclude that it is indeed polynomial-time computable for fixed k, while it is $\mathcal{NP}$-complete if k is part of the input. In the following, we formalize the problem.

We define $G = (V, E)$ as undirected, simple graph with vertices V and edges E and denote the amount of vertices and edges in G by $n = |V|$ and $m = |E|$, respectively. Furthermore, we call the set $\binom{V}{2} \setminus E$ the *complementary edges* of G. An *augmentation* $F \subseteq \binom{V}{2} \setminus E$ is a subset of complementary edges and the *augmented graph* is denoted by $G+F = (V, E \cup F)$. For a vertex $v \in V$, we denote by the *degree* $\deg_G(v)$ the number of edges in E which are

incident to v. The index serves to avoid ambiguity if we want to compare the degree of v to its degree in a subgraph of G or a graph that results from augmenting G. If it is clear from the context which graph is meant, then we omit the index.
For a number $k \in \mathbb{N}$, the *k-core* of G is obtained by iteratively removing vertices $v \in V$ with $\deg(v) < k$. For a vertex $v \in V$, the *coreness* $\text{coreness}_G(v)$ is defined as the highest number $k \in \mathbb{N}$ for which v is in the k-core. Again, we omit the index if it is clear from the context which graph is meant. Based on these terms, the problem Subset k-CoreAugmentation is defined as follows.

Problem Subset k-CoreAugmentation
Instance: Graph $G = (V, E)$, positive integer k and a set $T \subseteq V$ of terminals
Goal: Find a minimum augmentation F such that all vertices in T are in the k-core in $G + F$

In the context of an instance of Subset k-CoreAugmentation, we call every $t \in T$ a *target vertex*. For an augmentation F, we call every vertex $v \in V \setminus T$ incident to an edge $e \in F$ a *helper vertex*. We define the problem Single Vertex k-CoreAug as follows.

Problem Single Vertex k-CoreAug
Instance: Graph $G = (V, E)$, positive integer k and a target vertex $t \in V$
Goal: Find a minimum augmentation F such that t is in the k-core in $G + F$

There also is an alternative definition of cores: In a graph $G = (V, E)$, a vertex $v \in V$ is in the k-core if and only if it is part of a subgraph $S = (V_S, E_S) \subseteq G$ for which every vertex $u \in V_S$ has degree $\deg_S(u) \geq k$. Clearly, since no vertex in S has degree less than k, no vertex is removed from S during construction of the k-core. Conversely, the k-core only contains vertices with a degree greater than or equal to k and is by definition a subgraph of G.
It follows from this alternative definition that if the k-core of a graph is not empty, then it contains at least $k + 1$ vertices, otherwise there would not be enough possible neighbors for each of them. Furthermore, a vertex $v \in V$ is in the k-core exactly if it has at least k neighbors which are in the k-core as well.

Outline

The remainder of this thesis is structured as follows: In Chapter 2, we prove that Single Vertex k-CoreAug is $\mathcal{NP}$-complete if k is part of the input and can not be efficiently approximated within a factor of $(1 - \delta) \log(k^2 \cdot n)$, unless $\mathcal{P} = \mathcal{NP}$. However, it can be efficiently solved for fixed k. In Chapter 3, we give efficient algorithms that approximate the optimal solution of arbitrary instances of Single Vertex k-CoreAug within a factor of $\frac{k^2+k}{2}$. Furthermore, we give a selection of efficient algorithms that approximate the optimal solution for restricted families of instances within a factor of k or better, depending on the structural parameters of the input graph. In Chapter 4, we develop a set of heuristic algorithms for Single Vertex k-CoreAug and provide reasoning on how fast they are and for which families of instances they should reasonably perform well, although we do not strictly prove performance guarantees for all of them. We conclude in Chapter 5 by summarizing our findings and by posing open questions which may be interesting for future research.

2. Complexity

In this chapter, we take up the question posed by Görke et al. and inspect how the complexity of Subset k-CoreAugmentation changes once we only require a single vertex in a graph to be lifted to the k-core. The following theorem states the most obvious result - restricting the number of target vertices does not invalidate the checking algorithm for possible solutions. However, the influence of the number of target vertices on the complexity is more subtle than just that, and will be discussed in more detail after this theorem.

Theorem 2.1. Single Vertex k-CoreAug *is in* $\mathcal{NP}$.

Proof. Subset k-CoreAugmentation is $\mathcal{NP}$-complete [GGR12]. While the general result is not directly applicable to Single Vertex k-CoreAug, we can employ the same checking algorithm to determine if a solution is valid for any instance of Single Vertex k-CoreAug; the only difference is that the required coreness only needs to be checked for one vertex. As the checking algorithm for Subset k-CoreAugmentation runs in polynomial time, it immediately follows that Single Vertex k-CoreAug $\in \mathcal{NP}$. □

We still have to discuss whether Single Vertex k-CoreAug can be efficiently solved. At a first glance, Subset k-CoreAugmentation clearly seems easier if the number of target vertices is reduced to a constant number, even more so if we only require one vertex to be lifted to the k-core. On the other hand, the degree of a vertex is also an upper bound on its coreness, so any algorithm that lifts one target vertex into the k-core must potentially lift a whole subset $S \subseteq V$ of helper vertices into the k-core as well.
While Görke et al. give a reduction to prove that Subset k-CoreAugmentation is $\mathcal{NP}$-complete for a fixed $k \geq 3$ but variable number of target vertices, we change their reduction to create instances with only one target vertex. However, in exchange, we have to let our reduction adjust k accordingly. In fact, we show in Section 2.3 that Single Vertex k-CoreAug can be solved polynomially, if k is fixed.

Our reduction heavily relies on an interesting property of cliques in relation to Core-Augmentation: In the following, let $C = (V_C, E_C)$ be a clique with $c = |V_C|$ vertices and let $v \in V_C$ arbitrary. Since vertices in V_C are all pairwise connected, v has coreness$(v) = c - 1$.
By removing a single, arbitrary edge $e \in E_C$, we can reduce the coreness of every $v \in V_C$ to coreness$(v) = c - 2$. Consequently, an augmentation could increase the coreness of every $v \in V_C$ back to coreness$(v) = c - 1$ by re-adding e.

This property of C is useful even in more general cases, where C is only a subgraph, as long as certain conditions are met. We use this frequently later in this chapter, so we define cliques where exactly one edge is missing as *clusters*.

Definition 2.2. *A* c-cluster *is a clique* $C = (V_C, E_C)$ *with* $|V_C| = c$ *where exactly one edge* $\{u, v\} \in E_C$ *is missing. We call* u *and* v *the* end vertices *of* C *and denote them by* $\text{end}_1(C)$ *and* $\text{end}_2(C)$*, respectively.*
We call $\text{inner}(C) := V_C \setminus \{\text{end}_1(C), \text{end}_2(C)\}$ *the* inner vertices *of* C *and designate one inner vertex* $w \in \text{inner}(C)$ *as representative* $\text{rep}(C) := w$.

For a c-cluster $C = (V_C, E_C)$, we call $e = \{\text{end}_1(C), \text{end}_2(C)\} \in \binom{V_C}{2} \setminus E_C$ the *closing edge*. We say that an augmentation F *closes* C if $e \in F$. Figure 2.1 shows a 3-cluster, a 4-cluster and a 5-cluster.

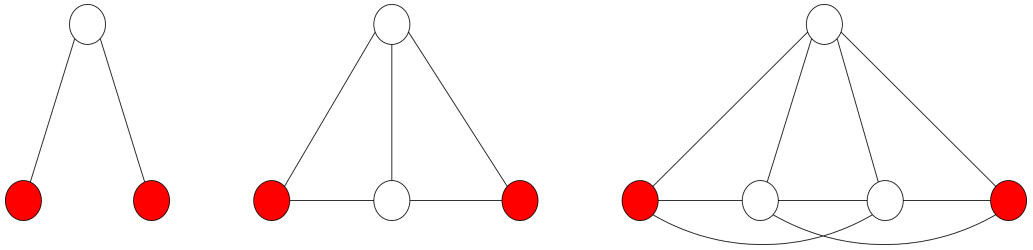

Figure 2.1: left: 3-cluster; middle: 4-cluster; right: 5-cluster. End vertices filled red.

Later in this chapter, we construct instances of Single Vertex k-CoreAug in order to give a reduction from DominatingSet. To be able to put the reduction and accompanying proofs in easier terms, we first define two kinds of frequently used structures which consist of clusters.

Definition 2.3. *A* cluster-overlap $\mathcal{C}$ *of size* $\text{size}(\mathcal{C})$ *is the result of* merging $\text{size}(\mathcal{C})$ c*-clusters* $C_1, \ldots, C_{\text{size}(\mathcal{C})}$ *for* $c \in \mathbb{N}$*. We merge the clusters by identifying their representatives with each other and we denote* $\text{center}(\mathcal{C}) = \text{rep}(C_1) = \ldots = \text{rep}(C_{\text{size}(\mathcal{C})})$.

Figure 2.2 shows three 5-clusters and how they are combined into an overlap of size 3.

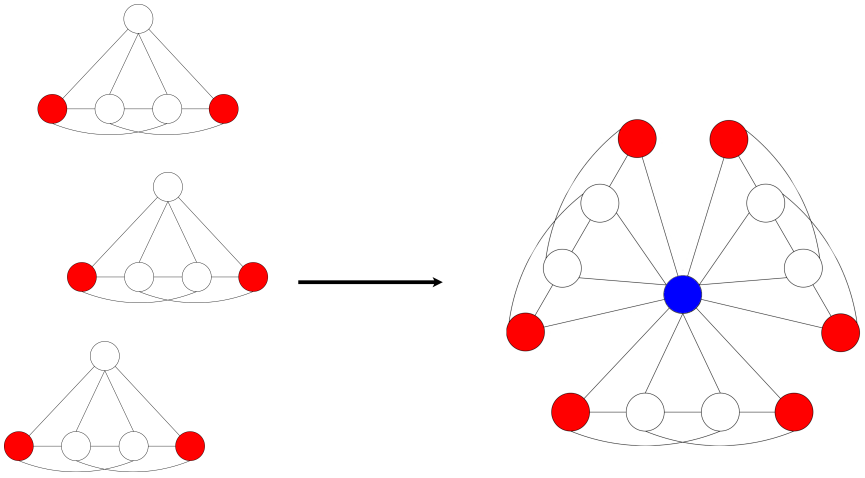

Figure 2.2: Cluster-overlap consisting of three 5-clusters. End vertices filled red, center filled blue.

Definition 2.4. *A* cluster-chain $\mathcal{C}$ *of length* $\text{length}(\mathcal{C})$ *consists of* $\text{length}(\mathcal{C})$ c*-clusters* ($c \in \mathbb{N}$) *denoted by* $C_1, \ldots, C_{\text{length}(\mathcal{C})}$ *and the additional edges* $\{\text{end}_2(C_i), \text{end}_1(C_{i+1})\}$ *for*

$1 \leq i \leq \text{length}(\mathcal{C}) - 1$.
We denote by $\text{end}_1(\mathcal{C}) = \text{end}_1(C_1)$ *and* $\text{end}_2(\mathcal{C}) = \text{end}_2(C_{\text{length}(\mathcal{C})})$ *the* end vertices *of* $\mathcal{C}$. *The* inner vertices *of* $\mathcal{C}$ *are denoted by* $\text{inner}(\mathcal{C}) = (V_1 \cup \ldots \cup V_{\text{length}(\mathcal{C})}) \setminus \text{end}(\mathcal{C})$.

For a cluster-chain $\mathcal{C}$, we call $e = \{\text{end}_1(\mathcal{C}), \text{end}_2(\mathcal{C})\}$ the *closing edge*. We say that an augmentation F *closes* $\mathcal{C}$ if $e \in F$. Figure 2.3 shows three 5-clusters and how they are combined to obtain a cluster-chain of length 3.

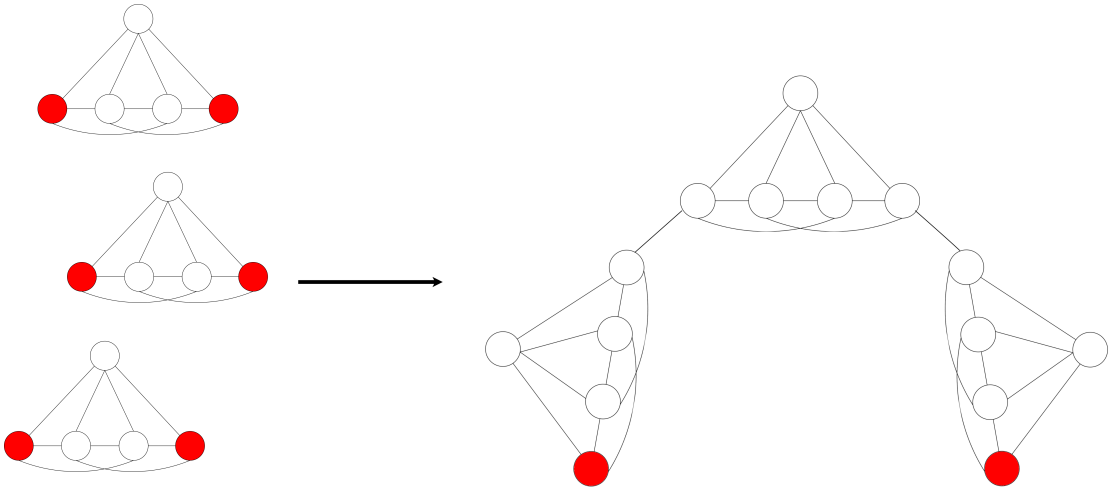

Figure 2.3: cluster-chain consisting of three 5-clusters. End vertices of the chain are filled red.

After defining these basic structures, we are now ready to give a reduction from another $\mathcal{NP}$-complete problem to Single Vertex k-CoreAug. We show that for our reduction, we can polynomially transform solutions for one problem into at most equally-sized solutions for the other, thus proving that Single Vertex k-CoreAug is $\mathcal{NP}$-complete if k is part of the input.

2.1 $\mathcal{NP}$-completeness

Our reduction transforms instances of DominatingSet to Single Vertex k-CoreAug. The problem DominatingSet is defined as follows.

Problem DominatingSet
Instance: Graph $G = (V, E)$.
Goal: Find a minimum set of vertices $S \subseteq V$ such that every $v \in V$ is in S or has a neighbor in S.

It is known that DominatingSet is $\mathcal{NP}$-complete [GJ79]. In the following, let $I = (G = (V, E), d)$ be an instance of the decision problem whether there is a dominating set S in G that contains at most d vertices.
We construct a corresponding instance $I' = (G' = (V', E'), t, k, d)$ of the decision problem whether there is an augmentation of size at most d that lifts the vertex t in G' to the k-core. Since our reduction is derived from that employed by Görke et al. [GGR12] in their proof that Subset k-CoreAugmentation is $\mathcal{NP}$-complete for $k \geq 3$, we give a quick outline of their reduction and state the changes that were necessary to adapt it to Single Vertex k-CoreAug. This is followed by a detailled description of our reduction.
Görke et al. create a *marking gadget* and a *checking gadget* for each vertex $v \in V$. The marking gadget is a cluster-chain of length $deg(v) + 1$ consisting of 4-clusters, where one of the clusters is associated with v. The other clusters are each associated with one of v's

neighbors in G. For each vertex $u \in V$, Görke et al. overlap every cluster associated with u and call the center of the resulting cluster-overlap the checking gadget for u. They require every checking gadget to be lifted to the 3-core. The idea is that each end vertex of a cluster-chain only has degree 2, while every inner vertex has degree at least 3. Thus, by closing the cluster-chain for a vertex $v \in V$, an augmentation can lift every inner vertex of that cluster-chain to the 3-core. This holds in particular for the checking gadgets of v and all its neighbors in G. Hence, closing a cluster-chain of a vertex $v \in V$ is equivalent to adding v to a subset $S \subseteq V$ for the original instance I of DominatingSet.
The basic idea behind our reduction is to add another vertex t to G' and to connect it to other vertices in G' such that for some $k \in \mathbb{N}$, t is in the k-core exactly if every checking gadget is. Obviously, choosing $k = 3$ and connecting t to every checking gadget does not work, since for $|V| \geq 3$, an augmentation could simply choose three checking gadgets and connnect them to a clique, thus solving our constructed instance with a constant number of edges. Our solution for this problem is to choose k depending on the number of vertices in V, adjust the number of vertices in the clusters accordingly and add a *diffusion gadget* between t and the checking gadgets. This ensures that if an augmentation does not lift every checking gadget to the k-core, then the number of required edges explodes in such a way that we can trivially find an augmentation of equal size that lifts every checking gadget to the k-core.

After stating the general idea of our reduction, we now provide detailled construction instructions. Without loss of generality, let $|V| \geq 4$ and even. This is possible because we can add an arbitrary number of isolated vertices to V and then invoke our reduction. Since those vertices have to be in every dominating set, we will simply remove them from the dominating set that we calculate from our solution for I' without explicitly stating it in the remainder.

In the first step of our reduction, we choose $k = 2 \cdot |V| + 2$. Then we create the marking gadget $\mathcal{C}_v$ for every vertex $v \in V$, which is a cluster-chain with $\text{length}(\mathcal{C}_v) = \deg(v) + 1$ that consists of $(k+1)$-clusters. We associate each vertex $u \in V$ that is adjacent to v with one cluster in $\mathcal{C}_v$ and denote that cluster by C_u^v. We associate the remaining cluster in $\mathcal{C}_v$ with v and denote it by C_v^v.

In the second step, we modify the graph to obtain the checking gadget for each vertex $v \in V$. For this, we overlap each cluster that is associated with v, yielding a cluster-overlap $\mathcal{J}_v$ with $\text{size}(\mathcal{J}_v) = \deg(v) + 1$.
Note that $\mathcal{J}_v$ consists of the clusters C_v^u from the marking gadgets of every vertex $u \in V$ that is adjacent to v, as well as the cluster C_v^v from v's own marking gadget.
We call $\text{center}(\mathcal{J}_v)$ the checking gadget for v.

In the third step, we create the diffusion gadget. This is a $(|V|+1)$-regular subgraph of G' consisting of $k - 2 = 2 \cdot |V|$ vertices. We create this subgraph by first creating the vertices and assigning each vertex a unique number from 1 to $2 \cdot |V|$.
We then add each edge $\{u, v\}$ where u is a vertex in the diffusion gadget that has an even number and v has an odd number, giving each vertex in the diffusion gadget degree $|V|$. Last, we partition both the evenly numbered and the oddly numbered vertices in pairs such that every vertex is contained in exactly one pair and add edges between the vertices that we just paired up. This is possible because $|V|$ is even and the diffusion gadget contains $2 \cdot |V|$ vertices, yielding an even number of both evenly and oddly numbered vertices. We connect each vertex in the diffusion gadget to each checking gadget.

In the last step, we add the target vertex t and connect it to every vertex in the diffusion gadget. Additionally, we add a $(k+1)$-clique to the graph and connect two arbitrary vertices from the clique to t. We require t to be lifted to the k-core.

Figures 2.4 through 2.6 depict example transformations on different input graphs and in different levels of detail.

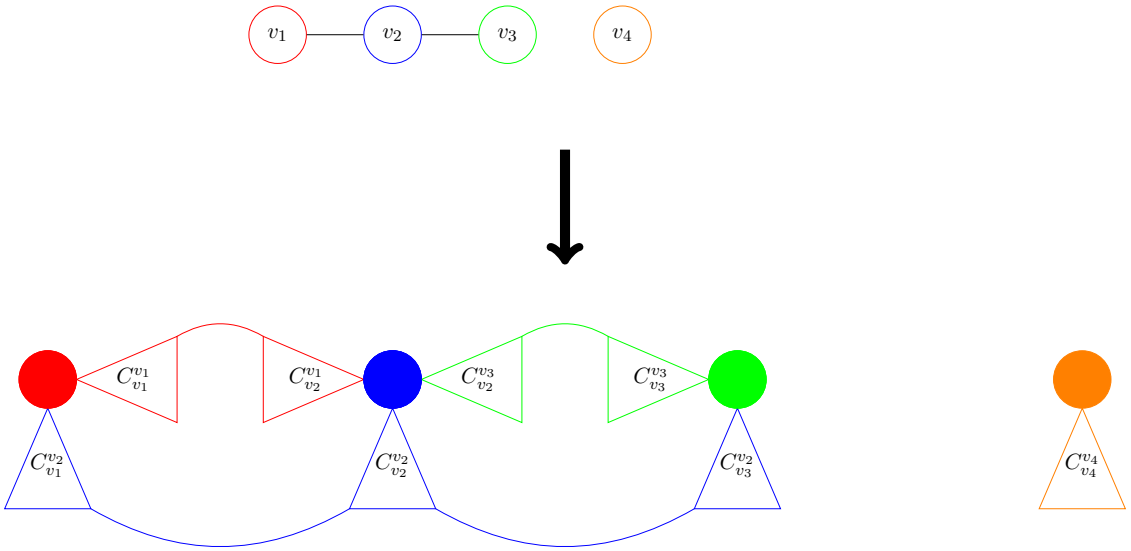

Figure 2.4: Example reduction for DominatingSet $\propto$ Single Vertex k-CoreAug, showing only marking and checking gadgets. Top: G. Bottom: G'. Marking gadgets are drawn in the same color as the corresponding vertex in G, checking gadgets are filled with that color.

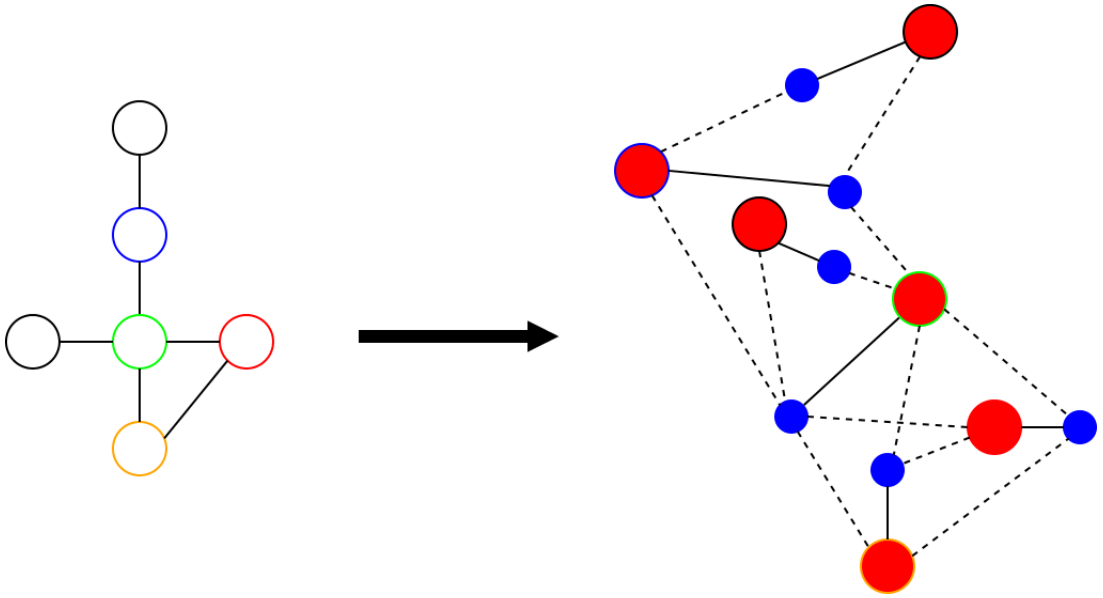

Figure 2.5: Example reduction for DominatingSet $\propto$ Single Vertex k-CoreAug, showing only marking and checking gadgets. Left: G. Right: G'. Marking gadgets and checking gadgets are depicted as small blue and big red dots, respectively. For every vertex $v \in V$, the checking gadget center($\mathcal{J}_v$) is connected to the marking gadget $\mathcal{C}_v$ with a solid line and to the marking gadget $\mathcal{C}_u$ for every neighbor u of v with a dashed line.

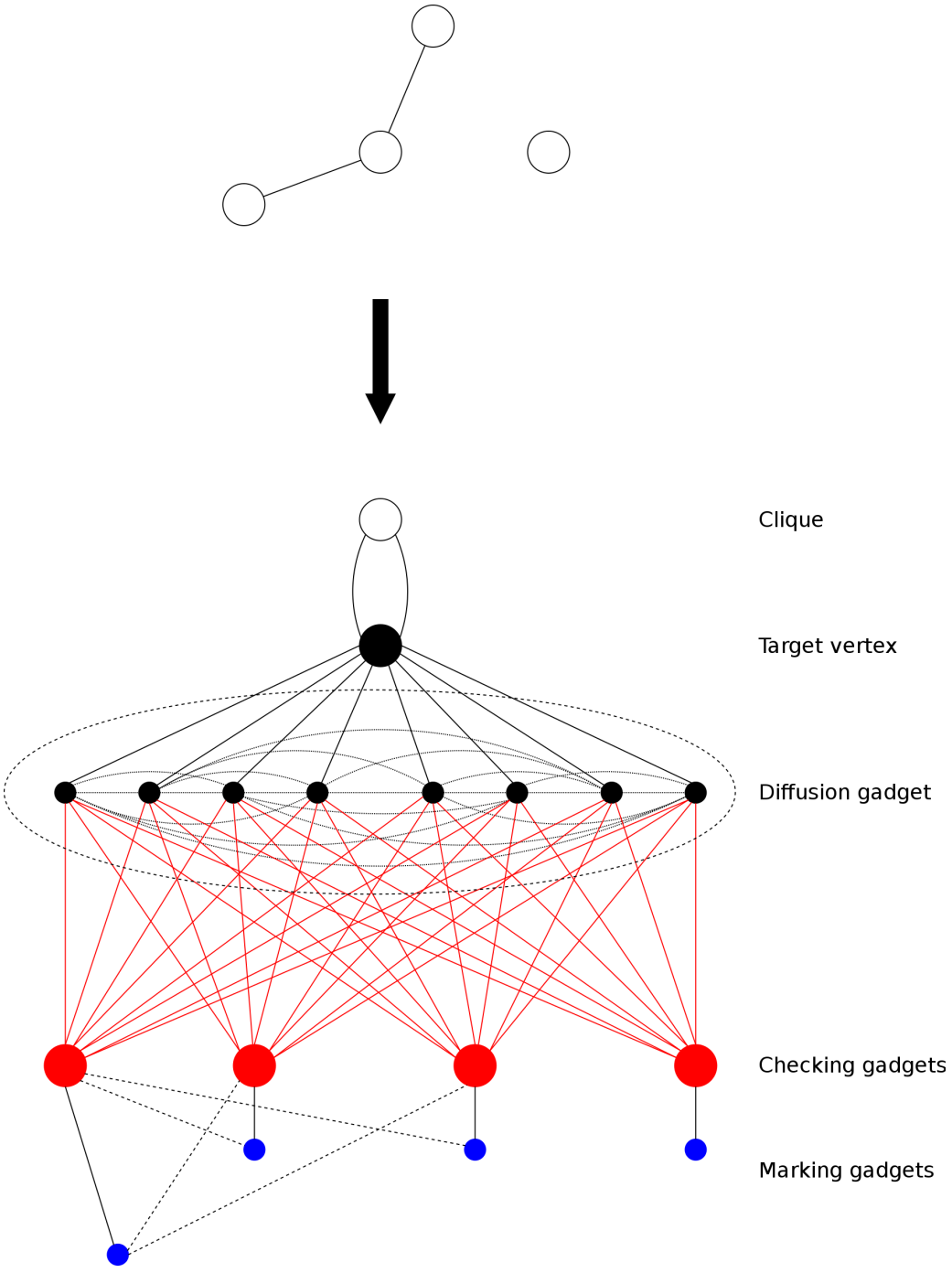

Figure 2.6: Example reduction for DominatingSet $\propto$ Single Vertex k-CoreAug. Top: G. Bottom: G'. Marking gadgets and checking gadgets are depicted as small blue and big red dots, respectively. For each vertex $v \in V$, the checking gadget center($\mathcal{J}_v$) is connected to the marking gadget $\mathcal{C}_v$ with a solid line and to the marking gadget $\mathcal{C}_u$ for every neighbor u of v with a dashed line. Small black vertices represent the diffusion gadget, big black and big white vertex represent the target vertex and the $(k+1)$-clique, respectively.

Runtime

Creating a $(k+1)$-cluster or a $(k+1)$-clique takes time $\mathcal{O}(k^2)$ for vertices and edges. There are $\mathcal{O}(\deg(v))$ $(k+1)$-clusters created for every vertex $v \in V$, where $\deg(v)$ itself is bounded by $|V|$, resulting in a running time of

$$\mathcal{O}(k^2 \cdot \sum_{v \in V} \deg(v)) \subseteq \mathcal{O}(|V|^2 \cdot |V|^2) \subseteq \mathcal{O}(|V|^4)$$

for creating the marking and checking gadgets. Our algorithm for creating the diffusion gadget clearly runs in polynomial time, and so does creating the $(k+1)$-clique and linking the gadgets. Hence the running time for the whole reduction is polynomial in $|V|$.

Correctness

We show that for every valid dominating set S that solves I, we can compute an augmentation F of smaller or equal size that solves I' and vice versa.

First, let S be a dominating set that solves I. We construct an augmentation F by closing the marking gadget $\mathcal{C}_v$ in G' for every $v \in S$.
First of all, we note that F contains exactly $|S| \leq d$ edges, so it has valid solution size with respect to I'. The following Lemma implies that in addition to having valid size, F also lifts t to the k-core:

Lemma 2.5. *Closing a cluster or marking gadget in the instances constructed by our reduction lifts every vertex in that cluster or marking gadget to the k-core.*

Proof. Counting only internal edges in a cluster, we observe that every inner vertex has degree k. The two end vertices both have degree $k-1$, so adding an edge between them increases both their degrees to k. Obviously, after closing the cluster, it is a subgraph of G' where every vertex has degree at least k and is thus in the k-core.
The argument for marking gadgets is very similar: Counting only internal edges, we observe that every inner vertex in every cluster has degree k. Since the clusters are additionally connected to a cluster-chain, every end vertex of every cluster has degree k except for the two end vertices of the whole cluster-chain, which have degree $k-1$. Closing the cluster-chain increases the degrees of both its end vertices to k, thus making the whole chain a subgraph where every vertex has degree at least k. It is thus in the k-core. □

We have to prove that F lifts t to the k-core in $G' + F$. By construction, the marking gadget $\mathcal{C}_v$ contains the checking gadget for every vertex $u \in V$ that is adjacent to v as well as the checking gadget for v itself. Since every vertex $w \in V$ is dominated by S, the checking gadget of w must be in at least one closed marking gadget in the augmented graph and is thus in the k-core.
For every vertex v in the diffusion gadget we have $\deg(v) = 2 \cdot |V| + 2 = k$, because they all have degree $|V| + 1$ from internal edges in the gadget and they are connected to each of the $|V|$ checking gadgets and the target vertex t. We have already shown that each checking gadget is in the k-core.
The target vertex t is connected to every one of the $2 \cdot |V|$ vertices in the diffusion gadget as well as two vertices from the $(k+1)$-clique and thus has $\deg(t) = 2 \cdot |V| + 2 = k$. Every vertex in the $(k+1)$-clique has degree k as well.
Thus, the closed marking gadgets, the diffusion gadget, t and the $(k+1)$-clique form a subgraph of G' in which every vertex has degree at least k and is thus in the k-core.
In conclusion, F has valid size and lifts t to the k-core in $G' + F$, so F solves I'.

Conversely, let F be an augmentation that lifts t to the k-core. We show how to obtain a dominating set S for G of smaller or equal size.

Initially, consider the special case where F only closes marking gadgets in G'. In this case, we can simply reverse the construction above by choosing the set S as the set of every vertex $v \in V$ whose marking gadget $\mathcal{C}_v$ is closed in the augmented graph $G' + F$. Note that if even a single checking gadget was not in the k-core in $G' + F$, then all of the vertices in the diffusion gadget would be left with at most $k-1$ adjacent vertices. So in order

for t to be in the k-core, F would either have to contain additional edges connecting the vertices in the diffusion gadget to new neighbors, lifting them to the k-core, or it would have to contain an edge connecting t to a new neighbor in the k-core for each vertex in the diffusion gadget that was not in the k-core in $G' + F$. But F only closes marking gadgets, so every checking gadget is in the k-core after all.
By construction, the marking gadget of a vertex $v \in V$ contains the checking gadget for v as well as the checking gadgets of all vertices adjacent to v in G. Since every checking gadget is in the k-core in $G' + F$, every vertex in V must either be in S or adjacent to a vertex in S, and S hence dominates G. Additionally, we have exactly one vertex in S for every closed marking gadget, which yields $|S| = |F|$.

Of course, not every augmentation F is structured as presumed above. In the following, we show that, nonetheless, every augmentation F that solves I' can be modified without adding any edges such that it still solves I' and only closes marking gadgets. This makes it possible to construct a dominating set S that solves I for *every* possible augmentation F that solves I'.
We identify two classes of augmentations F that can solve I': Those where not every checking gadget is in the k-core in $G' + F$ and those where every checking gadget is.

First, let r be the number of checking gadgets that are not in the k-core in $G' + F$ and assume $r \geq 1$. We count the smallest possible number of edges that F must contain in order to lift t to the k-core.
We denote by s the number of vertices in the diffusion gadget that F lifts to the k-core. Consider that t only has degree k in the unaugmented graph G', but is in the k-core in $G' + F$. Thus, for every vertex u in the diffusion gadget that is not in the k-core, F must replace the edge $\{t, u\} \in E$ by another edge $\{t, v\} \in F$ such that v actually is in the k-core. Since t has two adjacent vertices that already are in the k-core in the unaugmented graph G', the augmentation F must satisfy $k - s - 2$ incidences in order for t to be in the k-core in $G' + F$.
In addition to that, observe that for every checking gadget that is not in the k-core in $G'+F$, every vertex in the diffusion gadget misses one adjacent vertex in the k-core. Since we have r checking gadgets that are not in the k-core, F needs to satisfy $r \cdot s$ incidences to lift s vertices in the diffusion gadget to the k-core.
One edge can satisfy two incidences, which yields the formula

$$|F| \geq \frac{r \cdot s}{2} + \frac{k - 2 - s}{2} = \frac{r \cdot s + k - 2 - s}{2} = \frac{(r - 1) \cdot s + k - 2}{2}.$$

In order to find the minimum number of edges that F needs to contain, we need to minimize its size over the two variables r and s. Since we have $r \geq 1$ and $s \geq 0$, we obviously get a minimum if $r = 1$ or $s = 0$. Both yield

$$|F| \geq \frac{k - 2}{2} = \frac{2 \cdot |V| + 2 - 2}{2} = |V|.$$

But then, we can simply discard F and introduce the augmentation F_2, which closes every marking gadget. Every checking gadget is in at least one marking gadget, so as already proven above, every checking gadget is in the k-core. But then, every vertex in the diffusion gadget is in the k-core as well, and so is t. Thus, F_2 solves I' with $|F_2| \leq |F|$ and F_2 only closes marking gadgets. Then we can simply employ the algorithm above on F_2 to calculate a dominating set of equal size that solves I.

The only remaining cases are augmentations that lift every checking gadget to the k-core and contain edges that do not close marking gadgets. Again, we show how these augmentations can be transformed to smaller or equally-sized augmentations that lift at least the same vertices to the k-core and only close marking gadgets. The remainder is

very similar to the proof that Görke et al. already use for their transformation, with only few additions to account for the different structure of our graph.

For sake of simplicity, we temporarily allow the augmentation to contain loops and parallel edges. Furthermore, we assume that each vertex incident to edges in F is in the k-core, or otherwise we could remove the edge without changing the k-core, yielding a smaller solution.
Note that, since every checking gadget is in the k-core, it must have k neighbors that are in the k-core as well. Every vertex in the diffusion gadget is in the k-core and connected to every checking gadget, but there are only $k-2$ vertices in the diffusion gadget. Hence, every checking gadget needs two adjacent vertices in the k-core apart from the vertices in the diffusion gadget.
Thus, F must either contain two edges incident to every checking gadget, or, for checking gadgets where this is not the case, F must contain at least one edge incident to those checking gadgets and lift at least one of their neighbors (apart from vertices in the diffusion gadget) to the k-core. For each checking gadget for which none of the above is true, F must lift at least two of its neighbors to the k-core. In the following, we count the number of edges incident to vertices in a marking gadget that F needs to contain in order for even one vertex in that marking gadget to be in the k-core.

Every checking gadget is the center of an overlap of $(k+1)$-clusters. For an arbitrary checking gadget, we choose an arbitrary one of these clusters and count the number of edges incident to vertices (apart from the checking gadget) in that cluster that F needs to contain in order to lift even one of those vertices to the k-core. We denote by $r \in \mathbb{N}$ the number of vertices (apart from the checking gadget) in the cluster that is in the k-core in $G' + F$ and by $i \in \mathbb{N}$ the number of edges incident to vertices in the cluster that F needs to contain. We need a total number of $k \cdot r$ edges incident to the r vertices in order for them to be in the k-core. Since the checking gadget is in the k-core due to our preconditions and it is connected to every one of our selected vertices, we can subtract r from the total number of needed incidences. Furthermore, there are edges between all possible selected vertices except the end vertices of the cluster. First, we focus only on inner vertices of the cluster. We cover cases where its end vertices are lifted to the k-core later on. Since we do not lift end vertices and since the checking gadget does not count towards r in the first place, we get the restriction $r \leq k-2$ and we have $r^2 - r$ incidences by edges between inner vertices in the cluster. This yields the following value for i depending on r.

$$i \geq r \cdot k - r - (r^2 - r) = r \cdot k - r^2 = r \cdot (k - r)$$

This yields a minimum for both $r = 0$ and $r = k$, both of which are against our preconditions. Since the function above is a negative parabola that contains $(0, 0)$, we obtain minimal value such that r matches our preconditions for $r = 1$, which yields $i \geq k - 1$. But since we have $k = 2 \cdot |V| + 2$ and $|V| \geq 4$ by construction, this yields $i \geq 9$. Thus, if F lifts inner vertices of a cluster apart from the checking gadget to the k-core, then it must contain at least 9 incidences to vertices in that cluster.
We have not yet covered cases where F lifts end vertices of the cluster to the k-core. First, assume that F lifts exactly one end vertex of the cluster to the k-core and let end_1 be that vertex. Since we allow only one end vertex to be lifted to the k-core and that end vertex is still connected to every inner vertex of the cluster as well as to the checking gadget, the formula above still applies. The only difference is that if end_1 is not an end vertex of the marking gadget that contains the cluster, then it is connected to an end vertex of another cluster and thus possibly needs one less incidence, if the connected end vertex is in the k-core. This yields $i \geq r \cdot (k - r) - 1$ or $i \geq r \cdot (k - r)$, depending on the coreness of the connected end vertex. Clearly, the first formula yields a smaller value for every r. Thus, in the domain of allowed values for r we get minimal value for i for $r = 1$ or $r = k - 1$,

both of which yield a value of $i \geq k-2$. Thus, F must contain at least $i \geq 8$ incidences to vertices in the cluster.
The only remaining case is that F lifts both end vertices of the cluster to the k-core. We have already proven that even if F does not contain additional edges which are incident to inner vertices from the cluster apart from the end vertices, this will lift every vertex in the cluster to the k-core. We examine how F can lift the end vertices of the cluster to the k-core. If the marking gadget that contains the cluster only consists of a single cluster, then both end vertices of the cluster have degree $k-1$. Thus, F needs to satisfy one additional incidence for each end vertex and we have $i \geq 2$. Otherwise, if exactly one of the end vertices of the cluster is an end vertex of the marking gadget that contains the cluster, then that end vertex has degree $k-1$ and F needs to satisfy one additional incidence in order for that vertex to be in the k-core. The other end vertex is connected to an end vertex of another cluster. If F does not lift the connected end vertex to the k-core, then the end vertex in our cluster needs one additional neighbor in the k-core, and F needs to satisfy a total amount of $i \geq 2$ incidences, one for each end vertex. If F lifts the connected end vertex to the k-core, then F lifts at least one vertex in another cluster to the k-core. If neither of the end vertices of the cluster are end vertices of the marking gadget that contains the cluster, then we can apply the argument above to both end vertices - that is, F needs to satisfy one additional incidence for each end vertex, if it does not lift the respective connected end vertex to the k-core. But if F lifts a connected end vertex from another cluster to the k-core, then it clearly lifts at least one vertex from that cluster to the k-core. Thus, this counting argument can be applied recursively to the other clusters for which F lifts vertices to the k-core.
We have just proven that if any vertex in a marking gadget (except for checking gadgets) is in the k-core, then F needs to satisfy at least two incidences for vertices in that marking gadget.

For every marking gadget $\mathcal{C}$ that contains vertices in the k-core which are not checking gadgets, let $I \subseteq F$ be the edges of F that are incident to vertices in $\mathcal{C}$. We choose an arbitrary edge $\{u, v\} \in I$ where u is in $\mathcal{C}$ and replace it by $\{end_1(\mathcal{C}), v\}$. We replace every other edge $\{x, y\}$ where x is in $\mathcal{C}$ by $\{end_2(\mathcal{C}), y\}$. If a replacement creates a loop on one of the end vertices, then we replace the loop by the closing edge of $\mathcal{C}$. After these replacements, every vertex in $\mathcal{C}$ is in the k-core since both of $\mathcal{C}$'s end vertices are. This holds in particular for every checking gadget that $\mathcal{C}$ contains. Additionally, no incidences outside of $\mathcal{C}$ are changed. Thus, t remains in the k-core.
For every checking gadget d that has incident edges in F, let $I \subseteq F$ contain exactly those edges. If we have $|I| = 1$, then at least one of the marking gadgets that contains d must be in the k-core after performing the replacement above, because for the checking gadget to be in the k-core with only one edge incident to it in F, at least one of the neighbors of that checking gadget apart from vertices in the diffusion gadget must be in the k-core. But then, the replacement performed above has changed F such that the whole marking gadget containing that vertex is in the k-core. Let $\mathcal{C}$ be an arbitrary one of the marking gadgets that contain d and are in the k-core and let $\{\{d, v\}\} = I$. We replace the edge by $\{end_1(\mathcal{C}), v\}$.
If we have $|I| \geq 2$, then choose an arbitrary $\{d, v\} \in I$. We choose an arbitrary marking gadget $\mathcal{C}$ that contains d and replace our edge by $\{end_1(\mathcal{C}), v\}$. Every other edge $\{d, y\} \in I$ we replace by $\{end_2(\mathcal{C}), y\}$. Again, we replace any loops that may occur on end vertices by the closing edge of $\mathcal{C}$ and again, every vertex in $\mathcal{C}$ is in the k-core after performing the replacements. We have not changed any incidences outside of d, so again this can not influence the rest of the graph.
We call the new solution that results from exchanging edges for all marking and checking gadgets this way F_1. In this changed solution F_1, either no vertices in a marking gadget are incident to edges of F_1 or both end vertices are incident to edges of F_1. Hence, either

a whole marking gadget is in the k-core or it is not in the k-core at all. Additionally, F_1 contains no edges that are incident to inner vertices (including checking gadgets) of marking gadgets.

We have already proven that lifting each checking gadget to the k-core will lift t as well as every vertex in the diffusion gadget to the k-core, while every vertex in the $(k+1)$-clique is in the k-core even without an augmentation being applied. Thus, we can safely remove any incidences from these vertices without changing the k-core of the augmented graph. For this, we remove every edge from F_1 that is not incident to at least one end vertex of a marking gadget. Furthermore, let $I \subseteq F_1$ be the edges in F_1 that are incident to exactly one end vertex of a marking gadget. After the replacement performed above, every edge $e \in I$ is incident to one end vertex of a marking gadget and one vertex that is either t, or part of the diffusion gadget or $(k+1)$-clique. We replace e by the closing edge of the marking gadget. This does not affect the k-core since we can safely remove any incidences from any of the vertices that e may be connected to apart from the end vertex, and we lift every vertex in the marking gadget to the k-core by closing it.
We call F_2 the new solution that results from removing and replacing the edges in F_1 with the procedure described above. Additionally to the properties of F_1, F_2 lifts at least the same vertices to the k-core and only contains edges between end vertices of marking gadgets.

We now change the augmentation F_2 such that it only closes marking gadgets. For this, let $\mathcal{C}$ be a marking gadget that is not already closed by F_2, but for which every vertex is in the k-core. Then, due to the construction above, for both end vertices of $\mathcal{C}$ there is at least one incident edge in F_2. We arbitrarily choose the edges $\{end_1(\mathcal{C}), u\}$ and $\{end_2(\mathcal{C}), v\}$ from F_2 and replace them by the edges $\{end_1(\mathcal{C}), end_2(\mathcal{C})\}$ and $\{u, v\}$. If we have $u = v$ and this creates a loop $\{u, u\}$, then we replace the loop by the closing edge of the marking gadget that contains u. Obviously, this closes $\mathcal{C}$ and leaves the k-core unchanged.
We execute the procedure above as long as there are unclosed marking gadgets in $G' + F_2$ for which every vertex is in the k-core and call the augmentation obtained by the procedure above F_3.

Finally, we remove the temporarily allowed parallel edges from our augmentation. We do not need to replace loops, since we already replaced them by closing edges for marking gadgets right away. Observe that $G' + F_3$ only contains either closing edges for marking gadgets or edges between end vertices of different marking gadgets that are already closed. Thus, we simply remove all but one closing edge for each closed marking gadget as well as all edges between end vertices of different marking gadgets. We call the augmentation obtained by this procedure F_4.
Clearly, $|F_4| \leq |F|$ and every checking gadget is in the k-core in $G' + F_4$. Additionally, F_4 only contains closing edges for marking gadgets. Thus, we can obtain a dominating set S that solves I by the procedure described above. Together with the fact that this can obviously be performed in polynomial time, this proves the following theorem:

Theorem 2.6. Single Vertex k-CoreAug *is $\mathcal{NP}$-complete, if k is part of the input.*

2.2 Non-Approximability

We now examine how our results can be interpreted in terms of efficient approximation algorithms for Single Vertex k-CoreAug. The following theorem is based on our reduction from the previous section and covers the impossibility to efficiently approximate Single Vertex k-CoreAug within a certain factor of the optimal solution.
However, our reduction creates a very specific family of instances of Single Vertex k-CoreAug and using it, we cannot prove that there are no families of instances that can be

approximated efficiently. In fact, we further examine possible approximation algorithms in Chapter 2 and show that there are instances of Single Vertex k-CoreAug for which the optimal solution can be efficiently approximated, although we do not prove a relative performance guarantee close to this factor.

Theorem 2.7. *There is no efficient algorithm that approximates the best possible result of* Single Vertex k-CoreAug *within a factor of*

- $(1-\delta)\log(k^2 \cdot n)$
- $(3-\varepsilon)\log n$

for any $\delta, \varepsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$.

Proof. DominatingSet can not be efficiently approximated within a factor of $(1-\varepsilon)\log n$ for any $\varepsilon > 0$ [BYM84, Fei98].
If an approximation within the given factor was possible for Single Vertex k-CoreAug, then an approximation algorithm for DominatingSet could simply use our reduction to obtain an instance I' of Single Vertex k-CoreAug for any instance I of DominatingSet. For each vertex $v \in V$, I' contains one $(k+1)$-cluster (the component C_v^v). The clusters created for the vertices each contain $k+1$ vertices as well as $\frac{k^2+k}{2} - 1$ edges.
For each edge $\{u, v\} \in E$, I' contains two $(k+1)$-clusters C_u^v and C_v^u as well as two edges connecting these clusters to their respective marking gadgets. For each of those clusters, we count k vertices (since they overlap with the cluster created for the vertices u and v by one vertex) as well as $\frac{k^2+k}{2}$ edges.
So far, for marking and checking gadgets, we have $|V| \cdot (k+1) + 2 \cdot |E| \cdot k$ vertices as well as $|V| \cdot (\frac{k^2+k}{2} - 1) + 2 \cdot |E| \cdot \frac{k^2+k}{2}$ edges. We continue by counting the vertices and edges that are created by our reduction independent from the structure of G.
The instance I' contains a $(k+1)$-clique (with $k+1$ vertices and $\frac{k^2+k}{2}$ egdes) and the diffusion gadget, which contains $k - 2 = 2 \cdot |V|$ vertices and $(|V|+1) \cdot |V|$ edges. The diffusion gadget is connected to the target vertex with $2 \cdot |V|$ edges and to the checking gadgets with $2 \cdot |V|^2$ edges. Finally, we have two edges connecting the target vertex to two vertices from the $(k+1)$-clique.

We see a total increment in the number of vertices in G' by a factor that is linear in k, as well as an increment in the number of edges by a factor that is quadratic in k. This results in an overall increment of the input size by a factor in $\Theta(k^2)$.
Since, for a valid solution of I', there exists a solution for I of equal size, an approximation algorithm for DominatingSet can calculate an approximation within the factor $(1-\varepsilon)\log n$ for I from any solution that approximated I' within a factor of $(1-\varepsilon)\log(k^2 \cdot n)$. This contradicts the non-approximability of DominatingSet.

For the second part, note that

$$(1-\varepsilon)\log(k^2 \cdot n) \leq (1-\varepsilon)\log(n^3) = 3 \cdot (1-\varepsilon)\log(n)$$

which amounts to $(3-\varepsilon)\log(n)$ for some other $\varepsilon > 0$. □

2.3 Fixed parameter k

In this thesis, we do not give any conclusions about whether Single Vertex k-CoreAug is fixed parameter tractable. However, if we set k fixed, then we can solve Single Vertex k-CoreAug optimally in polynomial time, where the degree of the polynom depends on k. We formulate a more precise notion of this in the following theorem.

Theorem 2.8. *An instance $I = (G = (V, E), k, t)$ of* Single Vertex k-CoreAug *can be solved optimally in a runtime of*

- $\mathrm{OPT} \cdot |V|^{2\,\mathrm{OPT}} \cdot f(G)$
- $\frac{k^2+k}{2} \cdot |V|^{k^2+k} \cdot f(G)$

where $f(G)$ denotes the (polynomial) running time for computing the k-core of G and possible augmented graphs $G + F$.

Proof. We solve I by systematically trying augmentations of increasing size. We perform a number of iterations and denote by $i \geq 1$ the number of the current iteration. In every iteration, we try every possible augmentation of size exactly i and calculate the k-core of every resulting augmented graph. We return the first augmentation for which t is in the k-core. The returned augmentation is clearly minimal.

The maximum number of possible edges between vertices in G is

$$|E| \leq \frac{(|V|-1)^2 + (|V|-1)}{2} = \frac{|V|^2 - 2 \cdot |V| + |V| - 1}{2} = \frac{|V|^2 - |V|}{2}$$

In every iteration, we choose every possible combination of i edges from the complementary edges of G, the number of which is limited by the same upper bound as $|E|$. This results in a total number of $\binom{(|V|^2 - |V|)/2}{i}$ combinations checked in the ith iteration. For m iterations, we obtain a total number of

$$\begin{aligned}
& \sum_{i=1}^{m} \binom{(|V|^2 - |V|)/2}{i} \\
= & \sum_{i=1}^{m} \frac{(\frac{|V|^2-|V|}{2})!}{i! \cdot (\frac{|V|^2-|V|}{2} - i)!} \\
\leq & \sum_{i=1}^{m} \frac{(\frac{|V|^2-|V|}{2})!}{(\frac{|V|^2-|V|}{2} - i)!} \\
\leq & \sum_{i=1}^{m} (\frac{|V|^2 - |V|}{2})^i \\
\leq & \sum_{i=1}^{m} (\frac{|V|^2 - |V|}{2})^m \\
\leq & \sum_{i=1}^{m} (|V|^2 - |V|)^m \\
\leq & \; m \cdot (|V|^2 - |V|)^m \\
\leq & \; m \cdot |V|^{2m}
\end{aligned}$$

combinations of edges checked.
The theorem becomes clear considering that we need to perform OPT iterations to find a minimal augmentation. Additionally, a minimal augmentation contains at most $\mathrm{OPT} \leq \frac{k^2+k}{2}$ edges, since otherwise we could obtain a smaller augmentation by choosing t as well as k other vertices in G and augmenting G such that the chosen vertices form a $(k+1)$-clique. □

3. Approximation

In Section 2.2 we prove that, in general, Single Vertex k-CoreAug can not be efficiently approximated within a factor that is logarithmic in k and n, unless $\mathcal{P} = \mathcal{NP}$. In this chapter, we prove that every instance of Single Vertex k-CoreAug can be efficiently approximated within a factor of $\frac{k^2+k}{2}$. Additionally, we give algorithms that can approximate some instances of Single Vertex k-CoreAug efficiently within a factor of k or better, depending on the structure of the input graph. Unfortunately, we are not able to give an algorithm that can efficiently approximate every instance of Single Vertex k-CoreAug within a factor of k.
The relative performance guarantee of all algorithms discussed in this chapter depends on k or on structural parameters of the input graph and is independent from its size. This is quite good, since we can assume $k << n$ in realistic instances.

Theorem 3.1. *An instance $I = (G = (V, E), k, t)$ of* Single Vertex k-CoreAug *can be efficiently approximated within a factor of $\frac{k^2+k}{2}$.*

Proof. We choose an arbitrary set $S \subseteq V \setminus t$ of k vertices and select F such that $S \cup \{t\}$ is a $(k+1)$-clique in $G + F$. Since a $(k+1)$-clique contains exactly $\frac{k^2+k}{2}$ edges and the optimal solution contains at least one edge, the statement follows immediately. □

This algorithm is quite naive, but yields better results if the graph provides special structural properties. In particular, if each vertex $v \in V$ has $\deg(v) < k$ (note that this implies that the k-core of the input graph is empty), then we can calculate the relative performance guarantee more precisely. For this, we define the maximum degree among the vertices in G as $d_{\max} := \max\{\deg(v) | v \in V\}$. In particular, the following theorem implies that the smaller $d_{\max}$ relative to k, the closer we are to a relative performance guarantee of 1.

Theorem 3.2. *An instance $I = (G = (V, E), k, t)$ of* Single Vertex k-CoreAug *can be efficiently approximated within a factor of $\frac{k}{k-d_{\max}} \leq k$, if $d_{\max} < k$.*

Proof. We employ the same algorithm as in Theorem 3.1 and denote by F the augmentation which it returns. The augmentation F connects $k+1$ vertices such that they form a clique in $G+F$. This clique contains exactly $\frac{k^2+k}{2}$ edges, so this is an upper bound on the size of F.
In the following, let F' be the optimal solution and $\text{OPT} := |F'|$. Clearly, F' must contain

so many edges that there are $k+1$ vertices in the augmented graph that have degree at least k. But every vertex in G has at most degree $d_{\max}$ and thus needs $k - d_{\max}$ additional incidences. Since every edge in F' can satisfy two incidences, it must contain at least $\text{OPT} \geq \frac{(k-d_{\max})\cdot(k+1)}{2}$ edges.
Additionally, note that we have $d_{\max} < k$ and that $d_{\max}$ only allows discrete values, thus $d_{\max} \leq (k-1)$. Considering all of the above, we obtain the formula

$$\frac{|F|}{\text{OPT}} \leq \frac{\frac{k\cdot(k+1)}{2}}{\frac{(k-d_{\max})(k+1)}{2}} = \frac{k}{k-d_{\max}} \leq \frac{k}{k-(k-1)} = k.$$

□

We now examine instances where the k-core is not empty in the unaugmented graph. Note that this is not complementary to the previous theorem.
For a vertex $v \in V$, we denote by $\text{neigh}^{\text{k}}(v) = \#\{\{v,u\} \in E | \, \text{coreness}(u) \geq k\}$ the number of vertices in G adjacent to v that are in the k-core. This definition allows us to state the size of the augmentations produced by the approximation algorithm which is used in the following two theorems. Additionally, the following lemma allows us to prove a lower bound on the size of the optimal solution even when there are vertices in the input graph of degree higher than k. Without this lemma, we would only be able to assume $\text{OPT} \geq 1$ in the following two theorems, thus allowing for less precision in our performance guarantees. In addition, we use this lemma frequently in Chapter 4, where we examine heuristic approaches for solving Single Vertex k-CoreAug.

Lemma 3.3. *An augmentation F can increase the coreness of every vertex $v \in V$ by at most $|F|$.*

Proof. As every augmented graph again can be used as input graph, it is sufficient to prove that an augmentation F of size $|F| = 1$ can only increase the coreness of a vertex in a graph $G = (V, E)$ by at most 1. We show this by proving that removing a single edge from G can not lower the coreness of any vertex in G by more than 1. We choose an arbitrary vertex $x \in V$ with $\text{coreness}_G(x) =: k$ and remove an arbitrary edge $e = \{u, v\} \in E$. We denote by $G' := (V, E \setminus \{e\})$ the graph resulting from removing e from G and calculate $\text{coreness}_{G'}(x)$ in the modified graph. We denote by S and S' the k-core of G and G', respectively. There are three cases to be considered:

- If neither u nor v is in S, then e is not incident to any vertex in S. Hence, removing e leaves the degree of every vertex in S unchanged. We thus obtain $S' = S$, x is in S' and $\text{coreness}_{G'}(x) = \text{coreness}_G(x)$.
- Either u or v is in S. Without loss of generality, let $u \in S$. Since $v \notin S$, the edge e is not in S either. Hence the degree of every vertex in S remains unaffected by removing e and we have $S' = S$. Again, we get $\text{coreness}_{G'}(x) = \text{coreness}_G(x)$.
- If both u and v are in S, then we have $\deg_{G'}(u) = \deg_G(u) - 1 \geq k - 1$ and $\deg_{G'}(v) = \deg_G(v) - 1 \geq k - 1$ while the degree of every other vertex remains unchanged. In particular, every other vertex that is in S still has degree greater than, or equal to k after removing e. Hence, every vertex that is in S remains in the $(k-1)$-core of G'. Thus, we obtain $\text{coreness}_{G'}(x) \geq \text{coreness}_G(x) - 1$.

□

We now use the previous lemma to prove the following two theorems.

Theorem 3.4. *An instance $I = (G = (V, E), k, t)$ of* Single Vertex k-CoreAug *can be efficiently approximated within a factor of $\frac{k-\text{neigh}^k(t)}{k-\text{coreness}(t)} \leq k - \text{neigh}^k(t) \leq k$ if the k-core of G is not empty.*

Proof. First, we compute the k-core of G. If t is already in the k-core, then we do not need to augment the graph and obtain a relative performance guarantee of 1, despite the fraction in the theorem being undefined for $k = \text{coreness}(t)$. Otherwise, we have $\text{coreness}(t) < k$ and thus, the fraction in our theorem is well-defined. In this case, since the k-core of G is not empty, we can augment the graph by adding arbitrary edges $\{t, v\}$ where $v \in V$ is in the k-core, until t is in the k-core as well. Clearly, this happens after $k - \text{neigh}^k(t)$ edges, since t has k neighbors in the k-core at that point. Let F' be the optimal solution with $|F'| = \text{OPT}$. Lemma 3.3 implies that $\text{OPT} \geq k - \text{coreness}(t)$. Thus, we get

$$\frac{|F|}{\text{OPT}} \leq \frac{k - \text{neigh}^k(t)}{k - \text{coreness}(t)} \leq k - \text{neigh}^k(t) \leq k.$$

□

Using the same algorithm and lower bound on the size of the optimal solution, we can also show the following absolute performance guarantee for instances where the k-core of the input graph is not empty.

Theorem 3.5. *An instance $I = (G = (V, E), k, t)$ of* Single Vertex k-CoreAug *can be efficiently approximated resulting in an augmentation that uses at most* $\text{OPT} + \text{coreness}(t) - \text{neigh}^k(t) \leq \text{OPT} + k - \text{neigh}^k(t) - 1$ *edges, if the k-core of G is not empty.*

Proof. We employ the same algorithm as above and denote by F the obtained augmentation, connecting t to vertices in the k-core until t is in the k-core as well. This leads to an augmentation of size $|F| \leq k - \text{neigh}^k(t)$. Like in the previous theorem, we denote by $\text{OPT} \geq k - \text{coreness}(t) \geq 1$ the size of the optimal solution.

We calculate the difference between $|F|$ and the size of the optimal solution and obtain

$$|F| - \text{OPT} \leq k - \text{neigh}^k(t) - \text{OPT} \leq k - \text{neigh}^k(t) - (k - \text{coreness}(t)) \leq k - \text{neigh}^k(t) - 1$$

□

Unfortunately, the theoretical performance guarantees discussed in this chapter all have preconditions on the structure of the graph. In addition to that, the algorithms used in our proofs are all very naive. It seems reasonable to assume that for practical situations, we can give algorithms that perform better than our performance guarantees, or perform well even when the preconditions for our performance guarantees do not apply. In the next chapter, we give a selection of heuristic algorithms that should perform fairly well in practical situations.

4. Heuristic approaches

In Chapter 3, we developed efficient approximation algorithms for Single Vertex k-CoreAug, but were not able to give an efficient algorithm such that we could guarantee a relative performance better than $\frac{k^2+k}{2}$ for every instance. In this chapter, we expand on this by developing two heuristic algorithms for Single Vertex k-CoreAug. One works for instances where the k-core of the input graph is not empty. This algorithm is able to consistently perform at least as well as our k-approximation in chapter 3. We discuss in how far the heuristic algorithm can be used for instances where the k-core of the input graph is empty and come to the conclusion that it can not solve every instance where that is the case. To complement this, we introduce another heuristic algorithm that can solve every possible instance of Single Vertex k-CoreAug, but for which we cannot give a performance guarantee. Both heuristic algorithms run in polynomial time and are parameterized such that the parameter massively influences their runtime and the quality of the solutions that they calculate. Thus, the parameter can be adjusted according to the situation to trade solution quality for runtime and vice versa.
First, we prove the following lemma which, in addition to Lemma 3.3, allows us to gain a more precice understanding of what a single edge in an approximation can do in the context of Single Vertex k-CoreAug. We use this lemma frequently in the remainder of this chapter.

Lemma 4.1. *An augmentation $F = \{\{u, v\}\}$ of size $|F| = 1$ can only increase the coreness of a vertex w, if* $\text{coreness}(w) = \min\{\text{coreness}(u), \text{coreness}(v)\}$.

Proof. Let $G = (V, E)$ be a graph with at least one edge and let $x \in V$ arbitrary. We remove an arbitrary edge $e = \{u, v\} \in E$ and denote by $G' = (V, E \setminus e)$ the graph obtained by removing e from G. The coreness of x in G is denoted by $k = \text{coreness}_G(x)$. The k-core of G and G' are denoted by S and S', respectively.
We prove that if $\text{coreness}_{G'}(x) \neq \text{coreness}_G(x)$, then $\text{coreness}_{G'}(x) = \min\{\text{coreness}_{G'}(u), \text{coreness}_{G'}(v)\}$. We have already shown in the proof of Lemma 3.3 that if $\text{coreness}_{G'}(x) \neq \text{coreness}_G(x)$, then $\text{coreness}_{G'}(x) = \text{coreness}_G(x) - 1$ as well as u and v are both in S. There are three cases to consider:

- If both u and v are in S', then $\deg_{G'}(u) \geq k$ and $\deg_{G'}(v) \geq k$. But since we only remove e, the degree of no other vertex in S changes. Thus, we have $S' = S$ and $\text{coreness}_{G'}(x) = \text{coreness}_G(x)$.

- Either u or v is in S'. Without loss of generality, let $u \in S'$. Since $v \notin S'$, Lemma 3.3 implies that $\text{coreness}_{G'}(v) = k - 1$ and, if the coreness of x changed at all, then we have $\text{coreness}_{G'}(x) = k - 1 = \text{coreness}_{G'}(v) = \min\{\text{coreness}_{G'}(u), \text{coreness}_{G'}(v)\}$.
- If neither u nor v is in S', then Lemma 3.3 implies that $\text{coreness}_{G'}(v) = \text{coreness}_{G'}(u) = k - 1$ and thus, if the coreness of x changed at all, then we have $\text{coreness}_{G'}(x) = k - 1 = \text{coreness}_{G'}(v) = \text{coreness}_{G'}(u) = \min\{\text{coreness}_{G'}(u), \text{coreness}_{G'}(v)\}$.

□

Our approximation algorithms for Single Vertex k-CoreAug depend heavily on the structural parameters of the input graph. First, we examine cases where the k-core of the input graph is not empty.

4.1 Non-empty k-core

If the k-core of the input graph is not empty, then we can use the basic algorithm from Theorem 3.4 and connect t to vertices in the k-core until t is in the k-core as well. This lifts t to the k-core with an augmentation F of size $|F| = k - \text{neigh}^{\text{k}}(t)$. In the following, we examine in which cases the size of the augmentation can be further reduced. For this, we define the *viability* of an augmentation as follows.

Definition 4.2. *Let $I = (G, k, t)$ an instance of* Single Vertex k-CoreAug *and let F be an augmentation for G. We denote by $G' := G + F$ the augmented graph. The* viability *of F is defined as* $\text{viability}(F) := \min\{k, \text{neigh}^{\text{k}}{}_{G'}(t)\} - \text{neigh}^{\text{k}}{}_{G}(t) - |F|$. *For the integer c with $c = \text{viability}(F)$, we say that F is c-*viable *on G.*

Put less formally, an augmentation has positive viability if at adds more neigbors in the k-core to t than it contains edges, but adding unnecessary edges after t is already lifted to the k-core does not count. In particular, the augmentation obtained by our approximation algorithm is always 0-viable. We focus on trying to find an augmentation F that has the highest possible viability, since the augmentation with the highest viability among those that lift t to the k-core is obviously the smallest, and thus, optimal. If this augmentation does not yet lift t to the k-core, then we can expand it by connecting t to vertices in the k-core until it is in the k-core as well. In the following, we prove that this does not change the viability.
Let G be a graph and t a target vertex in G that should be lifted to the k-core. Furthermore, let F be an augmentation for G that does not lift t to the k-core and let F_2 be an augmentation obtained by connecting t to neighbors in the k-core in $G + F$ until t is in the k-core as well. Since we have $k = \min\{k, \text{neigh}^{\text{k}}{}_{G+F+F_2}(t)\}$ and $k > \text{neigh}^{\text{k}}{}_{G+F}(t)$, we obtain

$$\begin{aligned}
\text{viability}(F \cup F_2) &= \min\{k, \text{neigh}^{\text{k}}{}_{G+F+F_2}(t)\} - \text{neigh}^{\text{k}}{}_{G}(t) - (|F| + |F_2|) \\
&= \text{neigh}^{\text{k}}{}_{G+F}(t) - \text{neigh}^{\text{k}}{}_{G+F}(t) + \min\{k, \text{neigh}^{\text{k}}{}_{G+F+F_2}(t)\} \\
&\quad - \text{neigh}^{\text{k}}{}_{G}(t) - |F| - |F_2| \\
&= \text{neigh}^{\text{k}}{}_{G+F}(t) - \text{neigh}^{\text{k}}{}_{G}(t) - |F| \\
&\quad + \min\{k, \text{neigh}^{\text{k}}{}_{G+F+F_2}(t)\} - \text{neigh}^{\text{k}}{}_{G+F}(t) - |F_2| \\
&= \min\{k, \text{neigh}^{\text{k}}{}_{G+F}(t)\} - \text{neigh}^{\text{k}}{}_{G}(t) - |F| \\
&\quad + \min\{k, \text{neigh}^{\text{k}}{}_{G+F+F_2}(t)\} - \text{neigh}^{\text{k}}{}_{G+F}(t) - |F_2| \\
&= \text{viability}(F) + \text{viability}(F_2) \\
&= \text{viability}(F).
\end{aligned}$$

Thus, if we manage to find an augmentation with maximal viability, then even if it does not lift t to the k-core, we can expand it to an augmentation with maximal viability that

does lift t to the k-core. The resulting augmentation $F \cup F_2$ is clearly optimal among those which lift t to the k-core, since it has maximal viability.
Note that although the calculation above may suggest it, viability is not a homomorphism between every augmentation for a graph and whole numbers.

In our search for an augmentation with maximum viability, we first focus on excluding augmentations that have negative viability and are thus worse than the augmentation obtained by connecting t to vertices in the k-core.
First, we show that if t only has neigbors in the k-core in G, then we can only find augmentations that are at most 0-viable. For this, consider that for any graph G, we clearly have $\text{neigh}^k{}_G(t) \leq \deg_G(t)$. Thus, for an augmentation F, we obtain

$$\begin{aligned} \text{viability}(F) &= \text{neigh}^k{}_{G+F}(t) - \text{neigh}^k{}_G(t) - |F| \\ &= \text{neigh}^k{}_{G+F}(t) - \deg_G(t) - |F| \\ &\leq \deg_{G+F}(t) - \deg_G(t) - |F|. \end{aligned}$$

But since every edge in F can only increase the degree of t by at most 1, we have $|F| \geq \deg_{G+F}(t) - \deg_G(t)$ and obtain the maximum $\text{viability}(F) \leq 0$. Hence, the 0-viable augmentation obtained by connecting t to vertices in the k-core is always an optimal solution if t only has neighbors in the k-core in G.
To decrease the running time of the algorithm presented later, we aim to exclude edges that can never lead to an optimal solution. For this, we show that an augmentation F that lifts t to the k-core must be less than 0-viable, if it lifts a vertex $v \in V$ with $\text{coreness}_G(v) < \text{neigh}^k{}_G(t)$ to the k-core. This implies that connecting t to vertices in the k-core always returns a solution that is better than F. For this, we refer to Lemma 3.3, which states that F must contain $|F| \geq k - \text{coreness}_G(v) > k - \text{neigh}^k{}_G(t)$ edges in order for v to be in the k-core in $G + F$. This yields the following formula for the viability of F, which proves our claim.

$$\begin{aligned} \text{viability}(F) &= \min\{k, \text{neigh}^k{}_{G+F}(t)\} - \text{neigh}^k{}_G(t) - |F| \\ &= k - \text{neigh}^k{}_G(t) - |F| \\ &< |F| - |F| \\ &= 0 \end{aligned}$$

We now discuss how an augmentation with high viability can be obtained for the input graph. For this, we refer to Theorem 2.8, which shows that for a fixed $d \in \mathbb{N}$, we can scan all possible augmentations of size at most d in a running time of $d \cdot |V|^{2 \cdot d} \cdot f(G)$. Here, $f(G)$ denotes the (polynomial) running time for computing the k-core of the input graph and possible augmented graphs. We use this result by trying each possible augmentation of size at most d and choosing the augmentation with the highest viability. If this augmentation lifts t to the k-core, then we have found the optimal solution. If, however, the most viable augmentation does not lift t to the k-core, then we further augment the resulting graph by connecting t to vertices in the k-core until t is in the k-core as well. We have already proven above that this retains the viability of the solution.
We have also proven above that an augmentation that lifts vertices of coreness less than $\text{neigh}^k(t)$ to the k-core is less than 0-viable. Lemma 4.1 shows that edges incident to those vertices can not influence the coreness of vertices which already are in a higher core, thus we can omit any edges incident to these vertices when searching for augmentations with positive viability. We define $S = \{v \in V | \, \text{coreness}(v) < \text{neigh}^k(t)\}$ and obtain a running time of $\mathcal{O}(d \cdot (|V| - |S|)^{2 \cdot d})$. The parameter d must be chosen depending on the situation. We show in Theorem 3.4 that our algorithm is a $\frac{k - \text{neigh}^k(t)}{k - \text{coreness}(t)}$-approximation for every parameter d, since the approximation it returns is at least 0-viable. This is because scanning every possible augmentation of size d includes in particular those augmentations

connecting t to vertices in the k-core.
Theorem 2.8 implies that for $d = k - \text{neigh}^k(t) - 1$, we always obtain an optimal augmentation. For values $0 < d < k - \text{neigh}^k(t) - 1$, we return an optimal augmentation, if its size is smaller than or equal to d. This means that if our algorithm does not find an optimal solution and needs to further augment the graph by connecting t to vertices in the k-core, then we can safely assume $\text{OPT} > d$. We thus obtain a relative performance guarantee of

$$\frac{|F|}{\text{OPT}} < \frac{k - \text{neigh}^k(t)}{d}.$$

In absolute terms, we obtain a performance guarantee of

$$|F| - \text{OPT} \leq k - \text{neigh}^k(t) - d.$$

4.2 Empty k-core

In the following, we discuss how our results from the previous section can be used for input graphs with an empty k-core. We show that in some cases, our algorithm still yields a solution for which we can even prove a performance guarantee, while in other cases, we need another algorithm to solve the instance. Therefore, we conclude this section after the following discussion by giving an efficient heuristic algorithm that solves every possible instance of Single Vertex k-CoreAug, but for which we unfortunately cannot prove a performance guarantee.

If the k-core of the input graph G is empty, then, obviously, we cannot simply connect t to vertices which are in the k-core. We can, however, still select a fixed parameter $d \in \mathbb{N}$ and scan every possible augmentation of size d. We re-use the previous definition for the viability of an augmentation, but note that for an empty k-core in the input graph, we can no longer guarantee that there is an augmentation that is 0-viable. Thus, we can no longer restrict the search to only add edges between vertices with sufficient coreness and are left with the full runtime of $d \cdot |V|^{2 \cdot d} \cdot f(G)$. Additionally, there is a higher range for the parameter d, because augmentations now can contain up to $\frac{k^2+k}{2}$ edges. There are three possible outcomes of scanning every solution of size smaller than or equal to d.
If we find an augmentation that lifts t to the k-core, then since we scan the solutions in ascending order of size, we have found an optimal solution. We can return this solution without performing any further calculations.
If we find an augmentation F such that the k-core of $G + F$ is not empty, then we can execute our heuristic algorithm for input graphs with non-empty k-core on the graph $G+F$. We choose another parameter $d_2 \in \mathbb{N}$ such that $1 \leq d_2 < k - \text{neigh}^k{}_{G+F}(t)$ and calculate the set $S = \{v \in V \mid \text{coreness}(v) < \text{neigh}^k{}_{G+F}(t)\}$. This way, we obtain a total running time of $f(G) \cdot (d \cdot |V|^{2 \cdot d} + d_2 \cdot (|V| - |S|)^{2 \cdot d_2})$ for both algorithms. We denote by F_2 the augmentation returned by the second algorithm. Obviously, the augmentation $F \cup F_2$ lifts t to the k-core.
We adjust the performance guarantees of the heuristic algorithm for input graphs with non-empty k-core to match this scenario. For this, note that we can only prove that $\text{OPT} > d$, since otherwise we would have found an augmentation F that lifts t to the k-core in $G + F$. Unfortunately, we can not make such a connection between d_2 and OPT. We obtain a relative performance guarantee of

$$\frac{|F| + |F_2|}{\text{OPT}} \leq \frac{d + k - \text{neigh}^k{}_{G+F}(t)}{d} = 1 + \frac{k - \text{neigh}^k{}_{G+F}(t)}{d} \leq 1 + \frac{k}{d}.$$

The absolute performance guarantee evaluates to

$$|F| + |F_2| - \text{OPT} \leq d + k - \text{neigh}^k{}_{G+F}(t) - d = k - \text{neigh}^k{}_{G+F}(t) \leq k.$$

If we do not find an augmentation that lifts any vertices in the input graph to the k-core, then we clearly can not invoke our previous heuristic algorithm. Lemma 3.3 proves that if we choose $d < k - \max\{\text{coreness}(v) | v \in V\}$, then the none of the scanned augmentations can lift vertices to the k-core.

Thus, we need a more general approach which works for every instance. In Theorem 3.1, we give a $\frac{k^2+k}{2}$-approximation which works for every possible instance of Single Vertex k-CoreAug. This approximation selects k arbitrary vertices except t and then connects those vertices and t such that they form a $(k+1)$-clique. It is obvious that the performance of this solution depends heavily on the selected vertices. Clearly, the best selection of vertices would be a set such that the subgraph induced by the selected vertices and t contains a maximal amount of edges. But such a set of vertices is hard to find.
Ultimately, every strategy for selecting vertices heavily depends on the structural parameters of the input graph. Instead of trying to guess those beforehand, it seems reasonable to implement an algorithm that greedily selects vertices by assessing how many edges they add to the subgraph induced by all selected vertices. Additionally, such an algorithm does not need to be constrained to selecting only $k+1$ vertices including t, since it can calculate if selecting a larger amount of vertices will decrease the solution size.

Algorithm 1 constitutes a simple greedy algorithm for selecting vertices. It is parameterized by the *search depth* $c \in \mathbb{N}$ for $c \geq 1$. In every iteration, it calculates for every possible combination $T \in V$ of at most c unselected vertices, how many edges the subgraph induced by T and the previously selected vertices would contain. Based on this, it chooses to select that combination T such that after every iteration, the subgraph induced by all selected vertices contains a maximal amount of edges. It continues to do so until it has selected at least $k+1$ vertices including t and it can not find any combination T such that selecting each vertex in T would lower the amount of edges needed to lift every selected vertex to the k-core.
Let $S \subseteq V$ be the set of vertices that our algorithm selects. We create an augmentation F that lifts t to the k-core by iteratively choosing two selected vertices $u, v \in S$ such that their degree in $S + F$ is as low as possible and $\{u, v\} \notin E \cup F$. We add the edge $\{u, v\}$ to F and repeat this procedure as long as there are vertices $w \in S$ such that $\deg_{S+F}(w) < k$. Obviously, every vertex $v \in S$ has $\deg_{S+F} \geq k$ once we are done and is thus in the k-core. This holds in particular for t. Thus, our algorithm calculates a correct solution.

In the following, we calculate the running time for our algorithm. Calculating the internal edges in the subgraph induced by any subset of vertices takes $|E| \leq |V|^2$ time, since an implementation only needs to iterate over every edge and check whether it is in the induced subgraph or not. In every iteration, the for-loop is executed one time for each possible subset T of V with size $|T| \leq c$. This amounts to $|V|^c$ executions of the for-loop in every iteration. Clearly, the amount of iterations is limited by the number of vertices, since the algorithm can only select every vertex once. This amounts to a total running time in $\mathcal{O}(|V| \cdot |V|^c \cdot |V|^2) \subseteq \mathcal{O}(|V|^{c+3})$ for selecting the vertices.
Creating the edges between selected vertices is possible in $\mathcal{O}(k \cdot |V|^3 \cdot \log |V|) \subseteq \mathcal{O}(|V|^4)$, since an implementation can simply sort the selected vertices by degree for every edge that it needs to add. Then it can choose the vertex with the lowest degree and iterate through the remaining vertices in ascending order of degree until it finds another vertex such that no edge exists between them. This amounts to a total running time in $\mathcal{O}(|V|^{c+3} + |V|^3) \subseteq \mathcal{O}(|V|^{c+3})$.

Concluding this chapter, we discuss how our heuristic algorithm behaves for different parameters c. Clearly, choosing $c = 1$ will not lead to consistently good results, because the first selected vertex is either a random neighbor of t or a completely random vertex in G, if t does not have any neighbors. After selecting the first vertex, the algorithm

will preferredly traverse edges between already selected and unselected vertices such that the number of edges between already selected vertices and the newly selected vertex is maximal. We reason that the performance of our algorithm for $c = 1$ depends largely on chance as well as the structural parameters of the input graph. Note that for $c = 1$, in order for our algorithm to select more than $k + 1$ vertices overall, each vertex that is selected above that amount must be connected to k already selected vertices. For $c = 2$, our algorithm is already able to detect if two neighbors of t are connected to each other. Consequently, we expect our algorithm to continuously perform better, the larger c gets.

Algorithm 4.1: Greedily select vertices, try to maximize internal edges in the subgraph that they induce.

$S_0 := \{t\}$;
$i := 0$;
repeat
  $w := \infty$;
  $A := \emptyset$;
  forall the *possible* $T \subseteq V \setminus S_i$ *with* $1 \leq |T| \leq c$ **do**
    $weight := |T| \cdot k$
    $-\#\{\{u, v\} \in E | u \in T \wedge v \in T\}$
    $-\#\{\{u, v\} \in E | u \in T \wedge v \in S_i \vee u \in S_i \wedge v \in T\}$
    if $weight < w$ **then**
      $w := weight$;
      $A := T$;
  $S_{i+1} := S_i$;
  if $|S_i| \leq k \vee w < 0$ **then**
    $S_{i+1} := S_i \cup A$;
  $i := i + 1$;
until $|S_i| = |S_{i-1}|$;

5. Conclusion

In this thesis we have studied the problem of lifting a single vertex in a graph to the k-core by adding as few edges as possible. We proposed an algorithm that solves this problem in polynomial time for a fixed k and proved that if k is part of the input, then the problem is $\mathcal{NP}$-hard to solve optimally and to approximate within a factor that is logarithmic in both k and the number of vertices in the graph. On the positive side, we gave an efficient algorithm that approximates the problem within a factor that is quadratic in k and showed that depending on the structure of the graph, there are efficient algorithms that approximate the best possible result for some families of graphs within a factor that is linear in k or better. Furthermore, we discussed efficient heuristic algorithms which may provide better results than our theoretically sound approximation algorithms at the expense of increased running time.
Our most important question is how to generate realistic instances of the problem such that our heuristic algorithms can be evaluated against the optimal solution and the theoretically sound approximation algorithms. Another interesting question is whether there exists an efficient algorithm that approximates the best possible result within a factor of k for every possible instance, or if there exist efficient algorithms that provide a relative performance guarantee that is logarithmic in the input size for some or even all instances of the problem.

Bibliography

[AFG+06] Vinay Aggarwal, Anja Feldmann, Marco Gaertler, Robert Görke, and Dorothea Wagner. Analysis of overlay-underlay topology correlation using visualization. In *Proc. 5th IADIS International Conference WWW/Internet*, pages 385–392, Murcia, Spain, October 2006. Outstanding Paper Award.

[BYM84] Reuven Bar-Yehuda and Shlomo Moran. On approximation problems related to the independent set and vertex cover problems. *Discrete Applied Mathematics*, 9:1–10, 1984.

[CF10] Nicholas A Christakis and James H Fowler. Social network sensors for early detection of contagious outbreaks. *PloS one*, 5(9):e12948, 2010.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. 45:634–652, 1998.

[GGR12] Robert Görke, Roland Gröll, and Ignaz Rutter. Efficient algorithms for core augmentation problems. 2012.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition, 1979.

[WA05] Stefan Wuchty and Eivind Almaas. Peeling the yeast protein network. *Proteomics*, 5(2):444–449, 2005.