Karlsruhe Institute of Technology

Efficient Algorithms for Core Augmentation Problems

Studienarbeit von

Roland Gröll

An der Fakultät für Informatik
Institut für theoretische Informatik

Gutachter:	Prof. Dr. Dorothea Wagner
Betreuender Mitarbeiter:	Dr. Ignaz Rutter
Zweiter betreuender Mitarbeiter:	Dr. Robert Görke

Bearbeitungszeit: Mai 2011 – Dezember 2011

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Studienarbeit selbständig und ohne unerlaubte Hilfsmittel angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Karlsruhe, den 16. Januar 2012

Gröll, Roland

Table of Contents

1 Introduction **7**

2 Preliminaries **9**

3 Core and Connectivity Augmentation **13**
3.1 Our rewiring operations 15
3.2 Iterative rewiring 20
3.3 Making the graph connected 24
3.4 Runtime 25

4 Core Augmentation for Subsets **29**
4.1 Augmenting to the 3-core 29
4.2 Augmentation of subsets for $k < 3$ 33

5 Conclusion **37**

1. Introduction

Given a graph, we consider problems of adding a minimum number of edges to the graph, such that the new graph satisfies a specific property. These problems are called graph augmentation problems. The properties that are to be satisfied can concern the robustness, reliability or connectivity of a graph. Graphs are popular theoretical models for different real-world networks, for example, communication networks. Hence graph augmentation problems can be used to model problems of efficiently making a network more robust.

An example of a graph augmentation problem is Connectivity Augmentation. Given an unweighted and undirected graph $G = (V, E)$, find a set $F \subseteq V$ such that the augmented graph $G' = (V, E \cup F)$ is connected. This can be generalized to problems of augmenting a graph, such that it is k-edge-connected or k-vertex-connected. For $k = 2$, Eswaran and Tarjan have shown that edge- and vertex-connectivity augmentation can be solved in linear time. They have also shown that 2-edge-connectivity and 2-vertex-connectivity augmentation is NP-hard in the weighted case[5]. For larger k, Watanabe and Nakamura have shown that k-edge-connectivity augmentation can be solved in $O(k \min\{k, |V|\}|V|^4(k|V| + |E|))$ time [15] for $k \geq 2$. This has been improved by Nagamochi and Ibaraki, who have shown that a k-edge-connectivity augmentation can be found in $O((|V||E| + |V|^2 \log |V|) \log |V|)$ time using the maximum adjacency ordering of the vertices [10]. These are results for the unweighted and undirected case. If we furthermore want the augmented graph to be planar, 2-vertex-connectivity and 2-edge-connectivity are NP-complete, even in the unweighted case, as was shown by Kant and Bodlaender [3] for 2-vertex-connectivity and Rutter and Wolff [11] for 2-edge-connectivity.

After examining the work that has been done in the field of graph augmentation, we consider a measurement, the coreness, for vertices of a graph. The k-core of a graph consists of the vertices that are adjacent to at least k other vertices of the k-core. It can be constructed by iteratively removing vertices with a degree less than k. We say a vertex has coreness k if it is a vertex of the k-core. Hence the coreness of a vertex, i.e., the minimum k-core to which the vertex belongs, can be seen as a robust variation of the degree. It can be seen as a measurement of the "importance" of a vertex. The core structure of a graph is used in the analysis of communication networks, for example, peer-to-peer filesharing networks like Gnutella [1]. A version of the core structure in a directed graph was used by Subramanian et al. to determine the dense core of the internet, which proved to be a reliable method [12]. Furthermore the core structure plays a role in the analysis of protein networks [16], so there are also applications in the field of biology to be found. Görke[8] described a core generator that can generate a graph that satisfies a predetermined core

structure, which can be used among other things to experimentally test graph algorithms. With this in mind, it is of interest to augment a given graph such that it satisfies a specific core structure. Afterwards it can be examined, which other properties of the augmented graph have changed and in what ways it differs from the original graph.

In this work we will examine problems about augmenting the core structure of an unweighted and undirected graph. It can be seen, that the coreness of a vertex does not merely depend on local properties, unlike the degree. An edge in a different part of the graph can have influence on the coreness of a vertex. We will examine in which cases this will pose a problem in deriving an efficient algorithm for the augmentation of the coreness of vertices.

The structure of this work is as follows. In Chapter 2 we give some basic definitions and define the two problems Core Augmentation and Connectivity Augmentation. In Chapter 3 we examine CoreAug, the problem that results from combining Core Augmentation and Connectivity Augmentation. We will give an algorithm that solves this problem in polynomial time. In Chapter 4 we consider the problem Fixed Subset k-Core Augementation, in which a subset of vertices of a graph is given. The graph is then to be augmented such that this subset is in the k-core. We show that this problem is NP-hard for $k = 3$, but solvable in polynomial time for $k \leq 2$.

2. Preliminaries

In this chapter we give some basic definitions and examine the two problems Core Augmentation and Connectivity Augmentation, which are the building blocks of the problem we consider in the next chapter.

Let $G = (V, E)$ a graph. Unless noted otherwise, graphs are always undirected and not weighted. Let $V(G)$ denote the set of vertices of G and let $E(G)$ denote the set of edges of G. The k-core of a graph $G = (V, E)$ consists of the vertices that are connected to at least k other vertices of the k-core. The k-core can be calculated by repeatedly removing vertices $v \in V$ with $\deg_G(v) < k$ until all vertices $v \in V$ have $\deg_G(v) \geq k$. We denote the minimum k for that $v \in V$ is in the k-core of G by $\text{coreness}_G(v)$. A list of comma separated edges and/or paths denotes a concatenation of said edges and/or paths to a new path. To make the notation of graphs augmented by a set of edges F more clear we write $G + F$ for $(V, E \cup F)$ Let $\text{coreness}(G) := \min_{v \in V}\{\text{coreness}(v)\}$. We denote the connected component in G including v by $\text{comp}_G(v)$.

Problem 1 (Core Augmentation)**.** *Given a graph $G = (V, E)$, and an integer $k \in \mathbb{N}$, find $F \subset V^2$, $F \cap E = \emptyset$ with minimum cardinality such that coreness$(G') \geq k$ for the*

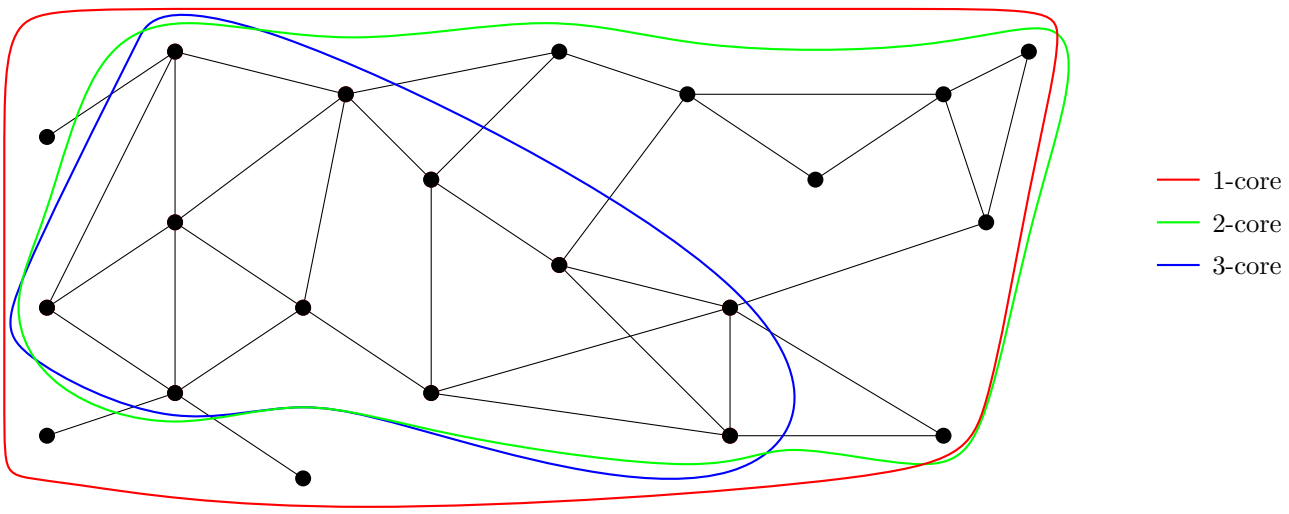


Figure 2.1: A graph with its core structure.

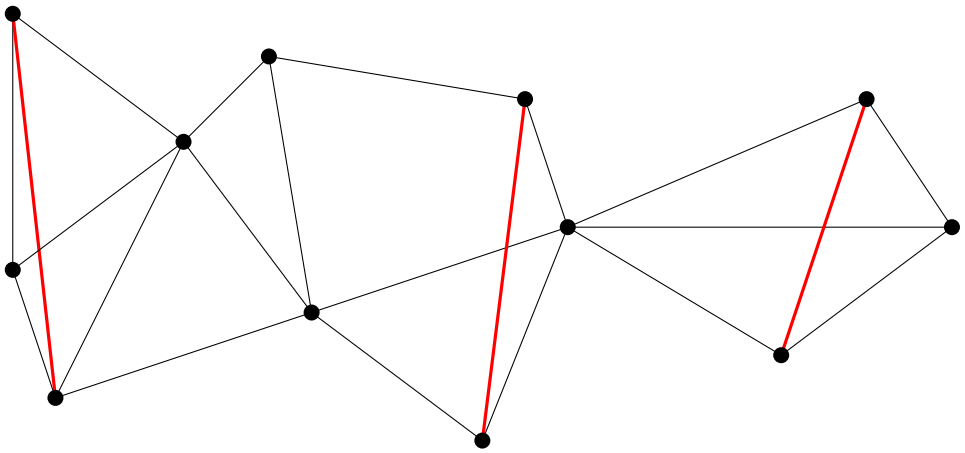


Figure 2.2: An example of a 3-core augmentation. Added edges are marked red.

graph $G' = G + F$

Let F be a solution to Core Augmentation. Observe that this problem can be simplified by finding F such that for all $v \in V : \deg_{G'}(v) \geq k$ holds, because we want every $v \in V$ to be in the k-core. If we only require a subset of V to be in the k-core we cannot use a similar simplification. See Figure 2.1 for a graph with vertices that have a degree of 3 but are not in the 3-core. This allows us to reduce Core Augmentation to a degree-constrained subgraph problem and use an existing algorithm to solve it. We denote the often used number of additional incident edges required to bring a vertex v into the k-core by $\text{req}_G(v) = \max\{\deg_G(v) - k, 0\}$.

Theorem 2.1. *For a graph* $G = (V, E)$ Core Augmentation *can be solved in* $O(|V||E^c| \min\{|E^c| \log |V|, |V|^2\})$ *time, with* E^c *being the edges of the complementary graph.*

Proof. To solve Core Augmentation we reduce it to a maximum weight degree-constrained subgraph problem (DCS), which is defined as follows. Given a weighted graph $G_g = (V_g, E_g)$ with integer bounds $\ell(v)$ and $\text{u}(v)$ for every vertex $v \in V_g$, find a subgraph H_g with maximum weight such that $\ell(v) \leq \deg_{H_g}(v) \leq \text{u}(v)$ for each $v \in V_g$. This can be solved in $\text{O}(\sum_{v \in V_g}(\text{u}(v)) \min\{|E_g| \log |V_g|, |V_g|^2\})$ time by an algorithm due to Gabow [7].

Let $G = (V, E)$, $k \in \mathbb{N}$ denote the input for Core Augmentation. Let $G^c = (V, E^c)$ be the complement of G. Then $F \subseteq E^c$ and for all $v \in V$ at least $\text{req}_G(v)$ edges in E^c incident to v must be chosen. We choose $\ell(v) = \max\{k - \deg(v), 0\}$ as a lower bound and $\text{u}(v) = |E^c|$ as a rough upper bound for each vertex. We set the weight of all edges to -1, thus a solution with maximum weight uses the minimum number of edges. We can now apply the algorithm for the maximum weight DCS problem given by Gabow to G^c with the mentioned bounds and weights. We call the resulting subgraph $S = (V, E^*)$ and the weight of the solution w_S. We use this solution for the degree-constrained subgraph problem to construct a solution to Core Augmentation and show that this is a valid solution. Let $G' = (V, E \cup E^*)$. Note that S is a subgraph of the complement of G, so $E \cap E^* = \emptyset$. For every vertex $v \in V$ it holds that $\deg_{G'}(v) \geq \deg_G(v) + \max\{k - \deg_G(v), 0\} \geq k$, so every vertex has degree at least k, and thus all vertices are in the k-core.

Now we show that G' is an optimal solution to Core Augmentation by assuming that G' does not have minimum cardinality. We then obtain a contradiction by constructing a maximum weight DCS solution to G^c with a heigher weight than the optimal solution

$S = (V, E^*)$. Assume that G' is not of minimum size, i.e., there is a solution $G^* = (V, E \cup F')$ with $|F'| < |E^*|$. We construct a solution S^* for the degree-constrained subgraph problem for G^c. For every edge in F' we add the corresponding edge to $E(S^*)$. It holds that for all $v \in V$ we have $\deg_{S^*}(v) \geq \max\{k - \deg_G(v), 0\}$, because there are at least $\max\{k - \deg_G(v), 0\}$ edges in F' incident to v. Let w_{S^*} be the weight of S^*. It follows that $w_{S^*} = -1|F'| > -1|E^*| = w_S$, a contradiction. Hence G' is of minimum size, and therefore an optimal solution to Core Augmentation.

The time complexity is

$$\mathrm{O}(\sum_{v \in V} (\mathrm{u}(v)) \min\{|E^c| \log |V|, |V|^2\}) = \mathrm{O}(|V||E^c| \min\{|E^c| \log |V|, |V|^2\})$$

□

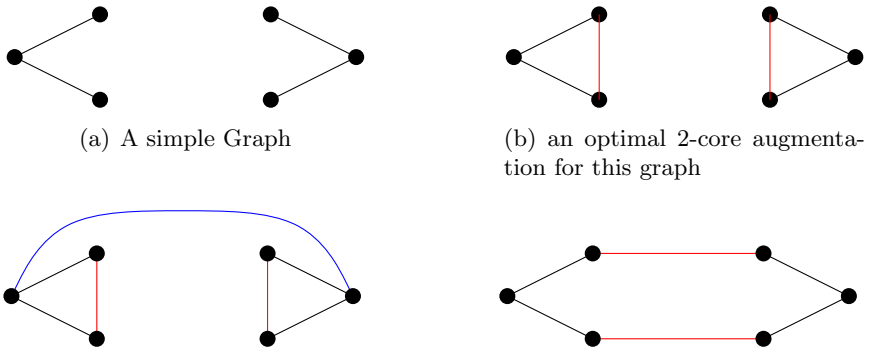
So we can get a solution to Core Augmentation by calculating a solution to Maximum Weight DCS. Another well-known augmentation problem concerns the connectivity of a graph.

Problem 2 (Connectivity Augmentation)**.** *Given a graph $G = (V, E)$ find $E_c \subseteq V^2$ with minimum cardinality such that $G' = (V, E \cup E_c)$ is connected.*

Let c be the number of connected components in G. Connectivity Augmentation can easily be solved in linear time by finding all connected components and connect them, using $c - 1$ edges.

3. Core and Connectivity Augmentation

As shown in the previous section, the augmentation problems Core Augmentation and Connectivity Augmentation can be easily solved seperately. If we solve these problems one by one, the number of added edges is generally not minimal though. An example for this can be seen in Figure 3.1. The graph consists of two paths of three vertices. It can be augmented into the 2-core by adding two edges, either making two cycles of three vertices or one big cycle of six vertices. The latter also connects the graph while the former does not. Hence some more work is necessary to solve both problems optimally at the same time. In this section we will devise an algorithm to do exactly that.



(a) A simple Graph

(b) an optimal 2-core augmentation for this graph

(c) a connectivity augmentation for the 2-core augmented graph

(d) An optimal augmentation that assures both coreness and connectivity

Figure 3.1: A simple example of core and connectivity augmentation

Problem 3 (ConnCore)**.** *Given a graph $G = (V, E)$, $k \in \mathbb{N}$ find $F \subseteq \binom{V}{2}$, $F \cap E = \emptyset$ with minimum cardinality such that $G + F$ is connected and all vertices $v \in V$ have coreness$(v) \geq k$ in the graph $G + F$.*

Problem ConnCore describes the problem of solving Core Augmentation and Connectivity Augmentation at the same time. It makes sense to differentiate between the connected components of the graph that need edges to lift its vertices into the k-core and and the connected components in which the vertices already have sufficient coreness.

Definition 1. *Given a graph $G = (V, E)$ and a number $k \in \mathbb{N}$, a connected component C of G is called a* satellite *when coreness$_G(v) \geq k$ for every vertex $v \in C$. The graph excluding satellites is called* centre. *For a given set of edges F we call the satellites that are in $G + F$ connected to a connected component of the centre* bound *satellites and those satellites that are not connected to connected components of the centre as* unbound *satellites.*

For the remainder of this section we can assume without loss of generality that every graph has a centre; for a graph without a centre ConnCore is equivalent to the problem Connectivity Augmentation, because nothing is to be done to achieve the coreness of the vertices. We formulate an algorithm to find a set of edges F of minimum size that solves ConnCore. The idea is to first solve Core Augmentation, i.e., use Theorem 2.1, and then rewire some of the new edges to ensure connectivity. Every rewiring step is desigend such that it reduces the number of connected components. If we get to the situation that we cannot apply one of our rewiring steps and the graph is not yet connected, additional edges are needed to ensure connectivity. This means that we first reduce the number of connected components as much as possible without increasing the number of edges and connect the graph afterwards.

Let F be an optimal solution to Core Augmentation and let $G' := G + F$ be the graph corresponding to this solution. The added edges in F have important attributes: We differentiate edges by the number of incidences that are needed to ensure coreness. We call an edge $\{u, v\}$ *essential* if $\deg_{G'}(u) = \deg_{G'}(v) = k$. An edge $\{u, v\}$ with $\deg_{G'}(u) = k$ and $\deg_{G'}(v) > k$ is called *half-essential* and an edge $\{u, v\}$ with $\deg_{G'}(u) > k$ and $\deg_{G'}(v) > k$ is called *non-essential*. This distinction is made because half-essential edges can be rewired in more circumstances than essential edges. Non-essential edges are superflous, i.e., they are not part of an optimal solution as we see in Lemma 3.1. A vertex $v \in V$ with $\deg_{G'}(v) > k$ is called *over-saturated*. Note that we only regard over-saturated edges incident to half-essential edges. Furthermore, it is important whether an edge in F is a bridge in G' or not because we can remove non-bridges without disconnecting previously connected components. The following lemma gives information about the distribution of half-essential and non-essential edges. We say a connected component C includes an edge e if $e \in C$.

Lemma 3.1. *Given a graph $G = (V, E)$ and a solution F of minimum size to* Core Augmentation *it holds that*

1. *there is at most one connected component in $G' = G + F$ that includes half-essential edges $e \in F$ and*
2. *there are no* non-essential *edges in F.*

Proof. Let $G = (V, E)$ and $G' = G + F$ the solution to Core Augmentation with G as input.

For 1. assume that there are two half-essential edges $e_1 = \{u, v\}, e_2 = \{x, y\} \in F$ in different connected components of G'. Assume without loss of generality that $\deg_{G'}(v) > k$ and $\deg_{G'}(y) > k$. After removing e_1 and e_2 and adding $\{u, x\}$ the degrees of all vertices except v and y remain the same. And it holds that $\deg_{G'}(v) \geq k$ and $\deg_{G'}(y) \geq k$. Therefore $F' = (F \setminus \{\{u, v\}, \{x, u\}\}) \cup \{\{u, x\}\}$ is a smaller solution contradicting the minimality of F.

For 2. assume that there were a non-essential edge $\{u, v\}$, but then $F' = F \setminus \{\{u, v\}\}$ would be a solution to Core Augmentation with $|F'| < |F|$. Therefore a solution with $\{u, v\}$ is not of minimum size.

□

The runtime for the algorithm solving Core Augmentation described in the previous section is quite large, because we choose $u(v) = |E^c|$ as an upper bound for every vertex $v \in V$. We can achieve a smaller runtime by adjusting the upper bounds for the underlying degree constraint subgraph problem. Let $G = (V, E)$ be the graph to be augmented into the k-core. A graph needs at least $k+1$ vertices to be in the k-core. Hence, we can assume that G has at least $k+1$ vertices. We choose an arbitrary set $W \subset V$ of $k+1$ vertices. For every vertex $w \in W$ we choose the upper bound $u(w) = |E^c|$. For every other vertex $v \in V \setminus W$ we choose the upper bound $u(w) = k$. We can transform every solution of the original problem to a solution of the changed problem with these new upper bounds. Let F be the solution to the original algorithm with the old upper bounds. First note, that there are no non-essential edges as seen in Lemma 3.1. For every half-essential edge $\{u, v\}$ we examine whether the over-saturated vertex u is in W. If it is not, we can exchange the edge with $\{v, x\}$, with $x \in W$. This works always because v has k neighbours in V and $k-1$ in W because $u \in V \setminus W$. Hence, at least one vertex of W is not yet a neighbour of v. If the over-saturated u is in W, nothing is to be done, because both u and v do not violate the new upper bounds. After every half-essential edge is exchanged in this way, every vertex in $V \setminus W$ has k neighbours in $G + F$ and every vertex $w \in W$ has at least k and at most $|E^c|$ neighbours. So there is a solution for the changed problem which also augments G into the k-core and has the same number of added edges as an optimal solution to the original problem. This reduces the runtime for the algorithm to $O(k|V| \min\{|E^c| \log(|V|), |V|^2\})$.

Observation 1. Core Augmentation *can be solved in* $O(k|V| \min\{|E^c| \log(|V|), |V|^2\})$ *time.*

3.1 Our rewiring operations

We use four different rewiring operations to transform an arbitrary solution of Core Augmentation into a solution that minimizes the number of connected components while still being a valid solution. This is the first step to get a solution to ConnCore. The operations are illustrated in Figure 3.2 and explained in Algorithm 1, 2, 3, 4 and 5, respectively. Descriptions of the rewire operations and lemmas describing their consequences follow.

RewireA needs a non-bridge $\{u, v\}$ and another edge $\{x, y\}$ in different connected components and connects these two components by removing $\{u, v\}$ and $\{x, y\}$ and adding $\{u, x\}$ and $\{v, y\}$, see Alg. 1. This operation does not change the degrees of the involved vertices as is stated in the following lemma.

Lemma 3.2. *Let $G = (V, E)$ be a graph and let F be a solution to* Core Augmentation *with edges $\{u, v\}$, $\{x, y\} \in F$ with $\{u, v\}$ being a non-bridge. Let $G' = G + F$ denote the graph for solution F. Let $F_r = F \setminus \{\{u, v\}, \{x, y\}\} \cup \{\{u, x\}, \{v, y\}\}$ be a new solution to* Core Augmentation *and $G_r = G + F_r$ be the graph corresponding to this new solution resulting from applying* RewireA *to G' . It holds that*

1. $\deg_{G'}(v) = \deg_{G_r}(v)$ *for all $v \in V$ and*
2. *G_r has one connected component less than G'.*

Proof. To see 1. note that for every vertex $v \in V \setminus \{u, v, x, y\}$ the incident edges do not change, and moreover each of the vertices in $\{u, v, x, y\}$ swaps one incident edge for another.

For 2. There is a path p in G' connecting u and v that uses neither $\{u, v\}$ nor $\{x, y\}$. This path also exists in G_r. The vertices x and y in G_r are connected by the path $\{x, u\}p\{v, y\}$. So in a path in G', $\{u, v\}$ can be replaced by p and $\{x, y\}$ can be replaced by $\{x, u\}p\{v, y\}$. Therefore, every vertex pair connected in G' is also connected in G_r and $\text{comp}_{G_r}(u) = \text{comp}_{G_r}(x)$ because of the added edges. □

Algorithm 1 RewireA(G, e_1, e_2)

Input: A graph $G = (V, E)$, two edges $e_1 = \{u, v\}$ and $e_2 = \{x, y\}$ with $\text{comp}_G(u) \neq \text{comp}_G(x)$ and one edge being a non-bridge
Output: A Graph $G_r = (V, E_r)$

1: $G_r = G$
2: remove $\{u, v\}$ and $\{x, y\}$ from G_r
3: add $\{u, x\}$ and $\{v, y\}$ to G_r
4: **return** G_r

RewireB takes a half-essential non-bridge $\{u, v\}$ with v being the oversaturated vertex, and another connected component C and changes $\{u, v\}$ to $\{u, x\}$ with $x \in C$, see Alg. 2. The following lemma states that RewireB produces a valid Core Augmentation solution, and creates no new non-bridges.

Lemma 3.3. *Let $G = (V, E)$ be a graph and let F be a solution to* Core Augmentation *and $G' = G+F$ be the corresponding graph. Let F include a half-essential non-bridge $\{u, v\}$, with v being the over-saturated vertex, and let x be a vertex with $comp_{G'}(x) \neq comp_{G'}(u)$. Let $F_r = (F \setminus \{\{u, v\}\}) \cup \{\{u, x\}\}$ be a new solution to* Core Augmentation *resulting from applying* RewireB *to G'. Let $G_r = G + F_r$ be the graph corresponding to this new solution. It holds that*

1. $\deg_{G_r}(v) \geq k$ *for all* $v \in V$,
2. *G_r has fewer connected components than G' and*
3. *the newly added edge $\{u, x\}$ is a bridge.*

Proof. To see 1., note that for every vertex $v \in V \setminus \{u, v, x\}$ the incident edges do not change. The over-saturated vertex u loses an incident edge, so $\deg_{G_r}(u) \geq k + 1 - 1 = k$ because $\deg_G(u) > k$. Vertex v swaps an incident edge for another. Vertex x gains an incident edge. So the rewired solution remains a valid solution.

For 2. there is a path p in G connecting u and v that does not use the edge $\{u, v\}$. So in a path in G, $\{u, v\}$ can be replaced by p to get a path in G_r. The connected components $\text{comp}_G(u)$ and $\text{comp}_G(v)$ are connected in G_r by $\{v, x\}$ while the other connected components remain as in G'.

To see 3., note that the vertices u and x are in different connected components in G', so $\{u, x\}$ is the only path connecting u and x in G_r. □

Algorithm 2 RewireB$(G, e, C \subset V)$

Input: A graph $G = (V, E)$, a half-essential non-bridge edge $e = \{u, v\}$ and a connected component C with $\text{comp}_G(u) \neq C$
Output: A Graph $G_r = (V, E_r)$

1: $G_r = G$
2: remove $\{u, v\}$ from G_r
3: $x =$ an arbitray vertex in C
4: add $\{u, x\}$ to G_r
5: **return** G_r

RewireC uses a half-essential bridge $\{r, s\}$ with s being the over-saturated vertex, a non-bridge $\{u, v\} \in \text{comp}_{G+F\setminus\{r,s\}}(r)$, an edge $\{x, y\} \in \text{comp}_{G+F\setminus\{r,s\}}(s)$ and a connected

component $C \neq \text{comp}_G(r)$ of G. The components C and $\text{comp}_G(r)$ become connected by removing $\{u, v\}$, $\{r, s\}$ and $\{x, y\}$ and adding $\{u, x\}$, $\{v, y\}$ and $\{r, z\}$, where z denotes an arbitrary vertex of C, see Alg. 3. The following Lemma shows that this works as intended and does produce a solution with fewer connected components.

Lemma 3.4. *Let $G = (V, E)$ be a graph and let F be a solution to* Core Augmentation *and $G' = G + F$ be the corresponding graph. Let F include a half-essential edge $\{r, s\}$ with s being the over-saturated vertex, a non-bridge $\{u, v\} \in comp_{G' \setminus \{r,s\}}(r)$, an edge $\{x, y\} \in comp_{G' \setminus \{r,s\}}(s)$ and a vertex z with $comp_{G'}(z) \neq comp_{G'}(u)$. Let $F_r = (F \setminus \{\{r, s\}, \{u, v\}, \{x, y\}\}) \cup \{\{u, x\}, \{v, y\}, \{r, z\}\}$ be a new solution to* Core Augmentation *resulting from applying* RewireC *to G'. Let $G_r = G + F_r$ be the graph corresponding to this new solution. It holds that*

1. *$\deg_{G_r}(v) \geq k$ for all $v \in V$,*
2. *G_r has fewer connected components than G',*
3. *if neither $\{u, v\}$ nor $\{x, y\}$ is half-essential, the added edges $\{u, x\}$ and $\{v, y\}$ are both not half-essential, and*
4. *the edge $\{r, z\}$ is a bridge.*

Proof. To see 1., note that for every vertex $v \in V \setminus \{r, s, u, v, x, y, z\}$ the incident edges do not change. Vertex s loses an incident edge, so $\deg_{G_r}(s) = \deg_{G'}(s) - 1 \geq k$ because $\deg_{G'}(s) > k$. Each vertex in $\{s, u, v, x, y\}$ swaps an incident edge for another. Vertex z gains an incident edge.

To see 2., note that there are paths p_1, p_2 and p_3 in G', connecting u with v, u with r and s with x respectively in G. Each of these paths can be chosen such that they do not use the edges $\{u, v\}$, $\{r, s\}$ and $\{x, y\}$. So in a path in G', $\{u, v\}$ can be replaced by p_1; $\{r, s\}$ can be replaced by $p_2, \{u, x\}, p_3$ and $\{x, y\}$ can be replaced by $\{x, u\}p_1\{v, y\}$ to get a path in G_r. Hence, vertices connected by a path in G' are also connected by a path in G_r, i.e., a connected component in G' is also a connected component in G_r. It holds that $\text{comp}_G(u)$ and $\text{comp}_G(z)$ are connected in G_r by $\{r, z\}$.

For 3. assume that one of the edges $\{u, x\}$ and $\{v, y\}$ in G_r is half-essential. Then at least one of the vertices in $\{u, v, x, y\}$ is over-saturated. These vertices have the same degree in G' and G_r. Hence one of these vertices is also over-saturated in G' and one of the edges $\{u, v\}$ and $\{x, y\}$ is half-essential in G'.

To see 4., note that the vertices r and z are chosen to be in different connected components of G', so $\{r, z\}$ is a bridge. □

Algorithm 3 RewireC$(G, e_1, e_2, e_3, C \subset V)$

Input: A graph $G = (V, E)$; three edges $e_1 = \{u, v\}$, $e_2 = \{x, y\}$ and $e_3 = \{r, s\}$ with $\{u, v\}$ being a non-bridge, $\{r, s\}$ being a half-essential bridge with s being over-saturated, $\text{comp}_{G \setminus \{r,s\}}(u) \neq \text{comp}_{G \setminus \{r,s\}}(x)$, and a connected component $C \neq \text{comp}_G(x)$

Output: A Graph $G_r = (V, E_r)$

1: $G_r = G$
2: remove $\{u, v\}$, $\{x, y\}$ and $\{r, s\}$ from G_r
3: $z =$ an arbitray vertex in C
4: add $\{u, x\}$, $\{v, y\}$ and $\{r, z\}$ to G_r
5: **return** G_r

RewireC' uses a half-essential bridge $\{r, s\}$ with s being the over-saturated vertex, a non-bridge $\{u, v\} \in \text{comp}_{G\setminus\{r,s\}}(r)$, an edge $\{x, y\} \in \text{comp}_{G\setminus\{r,s\}}(s)$ and a connected component $C \neq \text{comp}_G(r)$ of G. The components C and $\text{comp}_G(r)$ become connected by removing $\{u, v\}$, $\{r, s\}$ and $\{x, y\}$ and adding $\{u, x\}$, $\{v, y\}$ and $\{s, z\}$, where z denotes an arbitrary vertex of C, see Alg. 4. The following Lemma shows that this works as intended and does produce a solution with fewer connected components. This rewiring operation is similar to RewireC. The difference is the over-saturated vertex in the half-essential edge $\{r, s\}$ and the added edge $\{r, z\}$ in RewireC is exchanged for $\{s, z\}$. In the remainder of this work we will not differentiate between RewireC and RewireC' and consider RewireC and RewireC' as one rewire operation in which the position of the over-saturated vertex in $\{r, s\}$ is flexible.

Lemma 3.5. *Let $G = (V, E)$ be a graph and let F be a solution to* Core Augmentation *and $G' = G + F$ be the corresponding graph. Let F include a half-essential edge $\{r, s\}$ with r being the over-saturated vertex, a non-bridge $\{u, v\} \in comp_{G'\setminus\{r,s\}}(r)$, an edge $\{x, y\} \in comp_{G'\setminus\{r,s\}}(s)$ and a vertex z with $comp_{G'}(z) \neq comp_{G'}(u)$. Let $F_r = (F \setminus \{\{r, s\}, \{u, v\}, \{x, y\}\}) \cup \{\{u, x\}, \{v, y\}, \{s, z\}\}$ be a new solution to* Core Augmentation *resulting from applying* RewireC' *to G'. Let $G_r = G + F_r$ be the graph corresponding to this new solution. It holds that*

1. *$\deg_{G_r}(v) \geq k$ for all $v \in V$,*
2. *G_r has fewer connected components than G',*
3. *if neither $\{u, v\}$ nor $\{x, y\}$ is a half-essential, the added edges $\{u, x\}$ and $\{v, y\}$ are both not half-essential, and*
4. *the edge $\{r, z\}$ is a bridge.*

Proof. To see 1., note that for every vertex $v \in V \setminus \{r, s, u, v, x, y, z\}$ the incident edges do not change. Vertex r loses an incident edge, so $\deg_{G_r}(r) = \deg_{G'}(r) - 1 \geq k$ because $\deg_{G'}(r) > k$. Each vertex in $\{r, u, v, x, y\}$ swaps an incident edge for another. Vertex z gains an incident edge.

To see 2., note that there are paths p_1, p_2 and p_3 in G', connecting u with v, u with r and s with x respectively in G. Each of these paths can be chosen such that they do not use the edges $\{u, v\}$, $\{r, s\}$ and $\{x, y\}$. So in a path in G', $\{u, v\}$ can be replaced by p_1; $\{r, s\}$ can be replaced by $p_2, \{u, x\}, p_3$ and $\{x, y\}$ can be replaced by $\{x, u\}p_1\{v, y\}$ to get a path in G_r. Hence, vertices connected by a path in G' are also connected by a path in G_r, i.e., a connected component in G' is also a connected component in G_r. It holds that $\text{comp}_G(u)$ and $\text{comp}_G(z)$ are connected in G_r by $\{s, z\}$.

For 3. assume that one of the edges $\{u, x\}$ and vy in G_r is half-essential. Then at least one of the vertices in $\{u, v, x, y\}$ is over-saturated. These vertices have the same degree in G' and G_r. Hence one of these vertices is also over-saturated in G' and one of the edges $\{u, v\}$ and $\{x, y\}$ is half-essential in G'.

To see 4., note that the vertices r and z are chosen to be in different connected components of G', so $\{s, z\}$ is a bridge. □

RewireD uses a half-essential bridge $\{x, y\}$, with x being the over-saturated vertex and a non-bridge $\{u, v\}$ on the over-saturated side of $\{x, y\}$ to connect the connected component containing $\{x, y\}$ and $\{u, v\}$ to another connected component C of G. This is achieved by removing $\{x, y\}$ and $\{u, v\}$ and adding $\{u, z\}$ and $\{v, y\}$. This operation does not invalidate a solution and reduces the number of connected components, as is shown in the following lemma.

Algorithm 4 RewireC'$(G, e_1, e_2, e_3, C \subset V)$

Input: A graph $G = (V, E)$; three edges $e_1 = \{u, v\}$, $e_2 = \{x, y\}$ and $e_3 = \{r, s\}$ with $\{u, v\}$ being a non-bridge, $\{r, s\}$ being a half-essential bridge with r being over-saturated, $\text{comp}_{G\setminus\{r,s\}}(u) \neq \text{comp}_{G\setminus\{r,s\}}(x)$, and a connected component $C \neq \text{comp}_G(x)$

Output: A Graph $G_r = (V, E_r)$

1: $G_r = G$
2: remove $\{u, v\}$, $\{x, y\}$ and $\{r, s\}$ from G_r
3: $z =$ an arbitray vertex in C
4: add $\{u, x\}$, $\{v, y\}$ and $\{s, z\}$ to G_r
5: **return** G_r

Lemma 3.6. *Let $G = (V, E)$ be a graph and let F be a solution to* Core Augmentation *and $G' = G + F$ be the corresponding graph. Let F include a half-essential bridge $\{x, y\}$, with x being the over-saturated vertex, a non-bridge $\{u, v\} \in \text{comp}_{G'\setminus\{x,y\}}(x)$. Let z be a vertex with $\text{comp}_{G'}(z) \neq comp'_G(u)$. Let $G' = G + F$ denote the graph for solution F. Let $F_r = (F \setminus \{\{u, v\}, \{x, y\}\}) \cup \{\{u, z\}, \{v, y\}\}$ be a new solution resulting from applying* RewireD *to G'. The corresponding graph to this new solution is denoted by $G_r = G + F_r$. It holds that*

1. *$\deg_{G_r}(v) \geq k$ for all $v \in V$,*
2. *G_r has fewer connected components than G' and*
3. *the added edges $\{u, z\}$ and $\{v, y\}$ are both bridges.*

Proof. 1. For 1. note that for every vertex $v \in V \setminus \{u, v, x, y, z\}$ the incident edges do not change. Vertex x loses an incident edge, so $\deg_{G_r}(x) = \deg_G(x) - 1 \geq k$, because $\deg_G(x) > k$. Each vertex in $\{u, v, y\}$ swaps an incident edge for another. Vertex z gains an incident edge.

To see 2., note that in G' there is a path p connecting u and v that does not use $\{u, v\}$ because $\{u, v\}$ is a non-bridge. So in a path in G', $\{u, v\}$ can be replaced by p to obtain a path between thes same vertices in G_r. So vertices connected in G' are also connected in G_r. Moreover, it holds that $\text{comp}_G(u)$ and $\text{comp}_G(z)$ are connected in G_r by $\{u, z\}$.

For 3. note that the edge $\{u, z\}$ is a bridge because u and z are in different connected components in G and there is no other edge added that connects these components. The edge $\{v, y\}$ is a bridge because v and y are in different components in $(V, E \setminus \{\{x, y\}\})$ and there is no other edge added connecting these components.

□

So we get to the conclusion that our four defined rewiring operations do not invalidate a solution. This is noted in the following corollary

Corollary 3.7. *A solution F that was derived from a valid solution of* Core Augmentation *by applying one of our four rewiring operations is a valid solution.*

RewireA does not alter the degrees of the vertices, while RewireB, RewireC and RewireD reduce the degrees of an over-saturated vertex by one.

Algorithm 5 RewireD$(G, e_1, e_2, C \subset V)$

Input: A graph $G = (V, E)$; two edges $e_1 = \{u, v\}$ and $e_2 = \{x, y\}$ with $\{u, v\}$ being a non bridge and a connected component $C \neq \text{comp}_G(u) = comp_G(x)$

Output: A Graph $G_r = (V, E_r)$

1: $G_r = G$
2: remove $\{u, v\}$ and $\{x, y\}$ from G_r
3: $z =$ an arbitray vertex in C
4: add $\{u, z\}$ and $\{v, y\}$ to G_r
5: **return** G_r

3.2 Iterative rewiring

To transform an arbitrary solution F to a solution with a minimum number of connected components, we perform four steps using the four rewiring operations from above iteratively. There are two possible situations that can occur after every rewiring operation. The first is, that there is only one connected component left. The other one appears, if there are no non-bridges left in F. In both cases none of our four rewiring operations can be applied. If we reach one of these conditions or have completed all four steps, we are done with rewiring. We will show that this reduction of connected components is optimal, i.e., that we need to add more edges to a minimal solution to Core Augmentation to get a minimal solution to ConnCore in the last two cases.

The following four steps are used to gradually reduce the number of connected components. We do this by first connecting the components in the centre, i.e., the connected components of the graph $G + F$ that include edges of F, and then try to connect the satellites to the remainder of the graph. To achieve the former we use RewireA and for the latter we use RewireB, RewireC and RewireD.

1. Apply RewireA to G' and edges in F as often as we can. If we cannot apply RewireA anymore and G' consists of more than one connected component and there are still non-bridges in F left, all edges in F are in the same component of G'. Otherwise we could apply RewireA once more. In this case we continue with the next step.

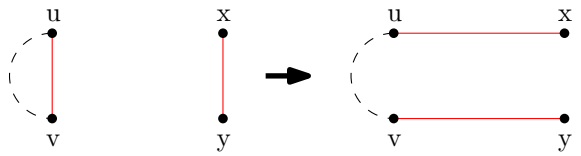
   Note that no rewiring operation seperates connected components, therefore we get the following observation.
   Observation 2. *In all steps after step 1 there is only one connected component containing edges in F, if there are still non-bridges in F.*

   We denote this component by M. For every connected component $C \neq M$ it holds that C is a component in G with $\deg_G(v) \geq k$ for all $v \in C$, i.e., C is a satellite.

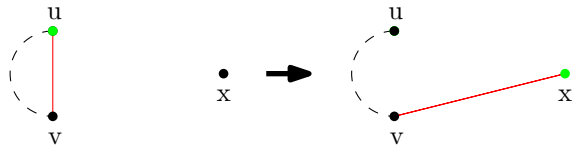
2. Apply RewireB to G' and edges in F as often as we can. If we cannot apply RewireB anymore and there is more than one connected component, then there are no half-essential non-bridges left. Otherwise we could apply RewireB once more. In this case we continue with the next step.

3. Apply RewireC to G' and edges in F as often as we can. If we cannot apply RewireC anymore and there is more than one connected component, then there is only one component $\text{comp}_{G' \setminus \{u,v\}}(x), x \in \{u, v\}$ with edges in F for every half-essential bridge $\{u, v\}$

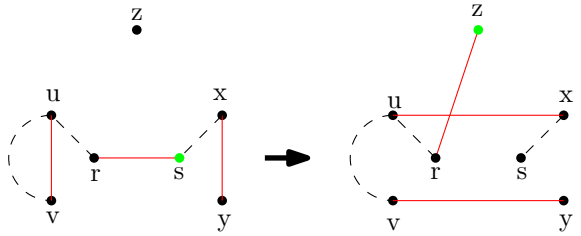
4. Apply RewireD to G' and edges in F as often as we can. If we cannot apply RewireD anymore and there is more than one connected component left, then



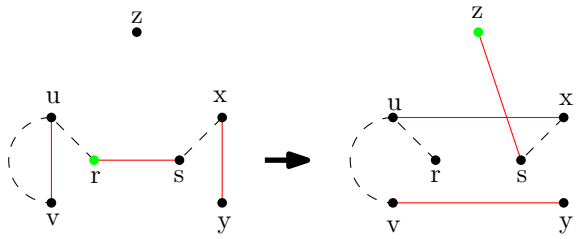
(a) Rewire A



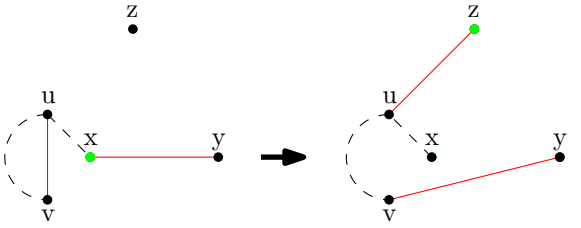
(b) Rewire B



(c) Rewire C



(d) Rewire C'



(e) Rewire D

Figure 3.2: The rewiring operations - a black dot denotes a vertex in G', a green dot denotes a vertex $v \in G'$ with $\deg_{G'}(v) > k$, a red line denotes an edge $e \in F$, a dashed line denotes a path between vertices without using edges represented by red lines

it holds that for every half-essential bridge $\{u, v\}$ with u being the over-saturated vertex, there are no edges of F in $\text{comp}_{G' \setminus \{u,v\}}(u)$.

Note that no rewiring operation after step 2 creates a new non-essential half-bridge. Hence, RewireB cannot be applied after step 2 is done. A theorem proving that this algorithm works as intended follows.

Theorem 3.8. *Let $G = (V, E)$ be a graph and F a solution to* Core Augmentation. *After altering F with the four steps, F is a solution to* Core Augmentation *that minimizes the number of connected components in $G' = G + F$.*

There are three cases to be considered, in particular, these are the two stopping criteria from above and a final, third case where neither applies. We have to show that in all cases the solution derived by the four step algorithm minimizes the number of connected components. To prove Theorem 3.8, we first show a number of lemmata, dealing with these cases seperately. In the first case there is only one connected component left. In the second case there is no non-bridge left. In the third case the rewiring steps are completed and the first two cases did not occur.

Case 1: One connected component

We get to the situation that there is only one connected component left. Because no rewire operation invalidates a valid solution to Core Augmentation (see Corollary 3.7) the following observation holds.

Observation 3. *Given a graph $G = (V, E)$, let F be a solution to* Core Augmentation. *If $G' = G + F$ has only one connected component, then F minimizes the number of connected components.*

Case 2: No non-bridges left

We get to a situation where there is more than one connected component left and there are no non-bridges in F left. To show that such a solution has a minimum number of connected components, we state a lower bound on the number of connected components in a graph with $|F|$ added edges and then show that our solution achieves this lower bound. Let $G = (V, E)$ be a graph with c connected components. We consider a set of edges E_{new} added to G to build a new graph $G' = G + E_{\text{new}}$. To get a lower bound on the number c' of connected components in G', we add the new edges individually. If we add an edge $e = \{u, v\}$ to G there are two cases to be considered. In the first case u and v are in the same connected component in G, and consequently e is not a bridge. The number of connected components c'' in $G'' = G + e$ is the same as in G. In the second case u and v are in different connected components of G and e is a bridge. The connected components of G including u and v are connected in the new graph G'' by this new edge e, while the other components of G remain the same in G''. Therefore, the number of components is reduced by 1, when we add a bridge to the graph. So if we add a set of new edges E_{new} to a graph, the lower bound for the number of connected components of the graph G' is $c - |E_{\text{new}}|$. More precisely, the number of connected components in G' is $c - b$, with b being the number of edges that are bridges at the moment of insertion in E_{new} by the argument above. Therefore, in this case the solution F minimizes the number of connected components of $G' = G + F$ because all edges in F are bridges in G'. This proves the following lemma

Lemma 3.9. *Given a graph $G = (V, E)$, let F be a solution to* Core Augmentation. *If all edges of F are bridges in the graph $G' = G + F$, then F minimizes the number of connected components.*

Case 3: All four steps are executed

There is more than one connected component in $G' = G + F$ and there are non-bridges in F. In this case all four steps are completed and G' has a specific form. Namely all edges in F are in the same connected component of G', there are no half-essential non-bridges left and for all half-essential bridges $\{u, v\}$ with u being the over-saturated vertex, it holds that there are no edges of F in $\text{comp}_{G' \setminus \{u,v\}}(u)$, i.e., $\text{comp}_G(u)$ is a satellite.

To show that F is an optimal solution, we quantify the degree of "over-saturatedness" of a solution of Core Augmentation to show that a solution with fewer connected

components has a higher degree of "over-saturatedness" and therefore more edges than F. Let $i(v, F)$ be the number of edges incident to v in F. The excess $\text{ex}(F)$ of a solution F of Core Augmentation is $\text{ex}(F) = \sum_{v \in V}(i(v, F) - \text{req}(v))$. Note that the excess is at least the number over-saturated vertices of a solution, because $i(v, F) > \text{req}(v)$ holds for these vertices.

After we rewired the edges in the four steps, all half-essential edges are used to connect satellites to other connected components, with the over-saturated vertex in the satellite. So for every vertex v the incident edges in F are either needed to increase the degree of v, or v is in a satellite and the only incident edge in F is half-essential. So $i(v, F) - \text{req}(v) = 0$ if v is not in a satellite and $i(v, F) - \text{req}(v) = 1$ if v is in a satellite. Hence, in $G' = G + F$ the excess is equal to the number of half-essential edges and equal to the number of bound satellites.

Lemma 3.10. *Given two different optimal solutions F and F' of* Core Augmentation*, the excess of these solutions is equal.*

Proof. Because both solutions F and F' are optimal it holds that $|F| = |F'|$. Hence, the excesses of these solutions are also equal.

$$\begin{aligned}
\text{ex}(F) &= \sum_{v \in V}(i(v, F) - \text{req}(v)) \\
&= \sum_{v \in V} i(v, F) + \sum_{v \in V} -\text{req}(v) \\
&= \sum_{v \in V} i(v, F') + \sum_{v \in V} -\text{req}(v) \\
&= \sum_{v \in V}(i(v, F') - \text{req}(v)) = \text{ex}(F')
\end{aligned}$$

□

Let F' be an optimal solution to Core Augmentation that mimimizes the number of connected components. We show, that the number of connected components in $G^* = G + F'$ is the same as in $G' = G + F$. The graph $G^* = (V, E \cup F')$ cannot have an edge connecting two satellites by Lemma 3.1, since such an edge would be a non-essential edge. So every bound satellite is connected to a component of the centre, i.e., the non-satellite part of the graph. Therefore, we can calculate the number of components of a solution by adding the number of components in the centre and the unbound satellites. The solution F' cannot connect more satellites to the rest of the graph than F. Otherwise, the excess of F' would be higher than the excess of F, because there is at least one over-saturated edge for every bound satellite. Let s be the number of satellites in G and let b be the number of bound satellites in solution F and b' be the number of bound satellites in solution F'. Furthermore, let c denote the number of connected components in the centre of G^*. Because $b \geq b'$ and $1 \leq c$ holds, we have $\text{components}(G') = s - b + 1 \leq s - b' + c = \text{components}(G^*)$. The optimal solution F' minimizes the number of connected components, so we also have $\text{components}(G') \geq \text{components}(G^*)$, and therefore $\text{components}(G') = \text{components}(G^*)$. The following lemma holds.

Lemma 3.11. *Given a graph $G = (V, E)$, let F be a solution to* Core Augmentation*. After changing F with the four steps algorithm and neither Case 1 nor Case 2 occurs, the new solution F minimizes the number of connected components.*

We see that the four steps transform a solution of Core Augmentation to a solution that minimizes the number of connected components. In all three possible end cases, the solution minimizes the number of connected components, by Observation 3 and Lemmata 3.9 and 3.11, and thus is a minimal solution to Core Augmentation.

3.3 Making the graph connected

As shown in the previous section, we can construct a solution to CoreAugmenation that minimizes the number of connected components in the resulting graph. We can use this solution to construct a minimum solution to ConnCore that simultaneously brings every vertex into the k-core and connects the graph. If the constructed solution F to Core Augmentation is already connected, there is nothing to do. If this is not the case, we need to add edges to reduce the number of connected components to 1. If there are non-bridges in F, we can use a simple operation to reduce the number of connected components by two with one added edge. If there are no non-bridges in F, we have to use one edge to connect one other connected component to another, reducing the number of connected components by one.

Given a graph $G = (V, E)$, a solution F to Core Augmentation and $G' = G + F$ let $\{u, v\} \in F$ be a non-bridge in G'. We can remove $\{u, v\}$ from F and add two half-essential bridges $\{u, x\}$ and $\{v, y\}$ to F with x and y being vertices of distinct connected components other than $\text{comp}_{G'}(u)$. We call this operation RewireE. The solution is still a valid one because no degree of the vertices of G' is reduced. Furthermore, the added edges $\{u, x\}$ and $\{v, y\}$ both are bridges, because u and x, respectively v and y were in different connected components before RewireE. This allows us to to reduce the number of connected components by two, using only one additional edge.

With this in mind we formulate an algorithm to fully connect a graph. As long as there are more than two connected components in G' and non-bridges in F, apply RewireE like above. After this we add one edge for every remaining satellite to connect this satellite to the remainder of the graph. This will be referred to as the *connecting algorithm*.

This connects the graph optimally, i.e., the least number of extra edges is used, as is shown in the following lemma.

Lemma 3.12. *Given a graph $G = (V, E)$ and a solution to* Core Augmentation *that minimizes the number of connected components of the corresponding graph $G' = G + F$. The solution of* ConnCore *obtained by applying the connecting algorithm is of minimum size.*

Proof. Let $G = (V, E)$ be a graph and let F be a solution to Core Augmentation that minimizes the number of connected components. We assume that $G + F$ is not connected. Otherwise the minimal solution to Core Augmentation is already a minimal solution to ConnCore. Let F_r be the solution to ConnCore obtained by the algorithm given above. We assume without loss of generality, that F is not empty, i.e., there is a centre. Otherwise ConnCore would degrade into a simple connectivity problem.

We will examine two cases. The first case is that there are only bridges in F_r. In the second case, there are non-bridges in F_r. Let c be the number of connected components in G. To connect the c connected components of G with a minimal number of edges we have to add $c - 1$ bridges. Hence, if our solution F_r consists solely of bridges, this solution is optimal.

In the second case there are non-bridges in F_r. If the four step algorithm to solve Core Augmentation stops, because there are only bridges in F left, the solution F_r of the

connecting algorithm consists solely of bridges, because only bridges are added to F. Hence we can assume without loss of generality, that the four step algorithm did not end prematurely, i.e., that all four rewiring steps were used if there are non-bridges in F_r. Because of this, every non-bridge in F is essential, otherwise RewireB could have been applied. Let s be the number of unbound satellites in $G+F$. There are at least s incidences needed to connect these unbound satellites to the remainder of the graph. The connection algorithm removes non-bridges from F as long as there are at least two unbound satellites left and adds two bridges for every removed non-bridge. If we use up all non-bridges in F to do this, F_r consists solely of bridges but we assumed that F_r includes non-bridges and the case of F_r consisting only of bridges was our first case. Hence we can assume without loss of generality, that there are enough non-bridges in F to apply RewireE until the graph is connected. So $\lfloor s/2 \rfloor$ edges are removed and s edges are added. Hence, F_r has $s - s/2 = s/2$ more edges, i.e., s more incidences, than F if s is even. If s is uneven, F_r has $s - (s-1)/2 = (s+1)/2$ more edges, i.e., $s+1$ more incidences, than F. Because at least s incidences were needed to connect the unbound satellites to the remainder of the graph, this is optimal. □

To summarize the results of this section we will give a short version of the algorithm that solves ConnCore for a graph $G = (V, E)$. First, calculate a solution for Core Augmentation using Theorem 2.1. Second, reduce the number of connected components by altering this solution with the four step algorithm. Third, connect the remainder of the graph by altering the solution using the connecting algorithm.

3.4 Runtime

In this section we examine the time complexity of solving ConnCore for a graph G by the method above. We get a solution to Core Augmentation in $O(k|V| \min\{|E^c| \log(|V|), |V|^2\})$ time by Observation 1. The next part consists of rewiring to reduce the number of connected components. Because every rewiring operation reduces the number of connected components, there are at most $c \leq |V|$ rewiring operations needed, with c being the number of connected components. To apply a rewiring operation, we need to find edges of F that have specific properties, i.e., whether it is essential or not and if it is a bridge. The degree of each vertex can be maintained. Hence, we can see in constant time if a vertex is over-saturated, i.e., if incident edges are either half-essential or essential. Note that, whereas the rewiring operations do not change bridges into non-bridges, the converse is generally not true. As an example consider a Graph G with a solution F for Core Augmentation. Let G consist of three connected components A, B and C. In A there are two vertices x and y with $\deg_G(x) = \deg_G(y) = k-1$. In B there is one vertex z with $\deg_G(z) = k-1$. Every other vertex has a degree of at least k. Let r be an arbitrary vertex of B. Then, $F = \{\{x, z\}, \{y, r\}\}$ is a solution with a half-essential and an essential non-bridge. RewireB can be applied to connect C to the remainder of the graph. After this rewiring F consists of two bridges, i.e., the former non-bridge $\{x, z\}$ is a bridge although it was not directly involved in a rewire operation. To see if an edge is a bridge, we use the dynamic connectivity data structure given by Holm and De Lichtenberg[9]. It uses $O(\log^2 n)$ time per insertion or deletion of an edge to the graph and $O(\log n / \log \log n)$ time per query to see if two vertices belong to the same connected component. This allows us to test whether $\{u, v\}$ is a bridge by deleting $\{u, v\}$ from the graph, querying if u and v are still connected and readding $\{u, v\}$ to the graph.

To examine the running time of the iterative rewiring, we first show how it can be implemented and then examine the runtime of such an implementation. To efficiently apply RewireA, we use two lists per connected component of $G + F$. In one list we store the

bridges of F and in the other list we store the potential non-bridges of F. The lists can be constructed in $O(|V| + |E|)$ time, using algorithm due to Tarjan.[13] To apply RewireA we need a potential non-bridge e from one connected component and an arbitrary edge f from another connected component. The potential non-bridge e has to be tested by the method above if it is really a non-bridge. This needs $O(\log^2 n)$ time. If e is a bridge, we add it to the list of bridges of the same component and pop another edge from the list with potential non-bridges and repeat the testing. We also test whether f is a bridge or not. If f is a bridge, the edges that are created by RewireA are bridges. If f is not a bridge, these new edges are non-bridges. The edges created by RewireA are inserted to the appropiate lists. When RewireA is completed, we concatenate the involved lists, i.e. we merge the lists for potential non-bridges and the lists for bridges. Every test if a potential non-bridge e is a a non-bridge either removes e from the list of potential non-bridges or leads to an application of RewireA. So there are at most $O(|V| + |E|)$ tests needed. The runtime for applying RewireA iteratively is therefore $O((|V| + |E|) \log^2 |V|)$.

To apply RewireB we need a half-essential non-bridge and a vertex of an unbound satellite. As in RewireA we maintain lists of potential non-bridges and of bridges. Because after iteratively applying RewireA all edges of F are in a single connected component, we only need two lists. We get the non-bridge as above in RewireA. To get the vertices of the other connected components, we save a list with one vertex per connected component that does not have an edge of F in it. In an individual RewireB we get the head of said list, apply RewireB and delete the used vertex from the list. Analogous to RewireA the runtime of iteratively applying RewireB is $O((|V| + |E|) \log^2 |V|)$.

To apply RewireC we need three edges and a vertex of an unbound satellite. One of the edges is a half-essential bridge $\{r, s\}$. The other edges are in $\text{comp}_{G+F-\{r,s\}}(r)$ and $\text{comp}_{G+F-\{r,s\}}(s)$ respectively. One of these two edges is a non-bridge. The needed vertex can be found by building a list with one vertex per connected component that does not have an edge of F in it. If a vertex of this list is used in a RewireC we remove it from the list. To find such a triple of edges, we temporarily remove the half-essential bridges of F from the graph and build lists of potential non-bridges and bridges per connected component as in RewireA. More formally let $H \subset F$ be the set of half-essential bridges of F. Then there are two lists for every connected component in $G+F-H$. After the lists are built we readd the half-essential bridges to the graph. Now we can search for a pair of edges of F, with one edge being a non-bridge, in different connected components in $G + F - H$ like in RewireA. Because every edge of F is in the same connected component of $G + F$, these two edges $\{u, v\}$ and $\{x, y\}$ are connected in $G + F$ by an edge of H. If we find such a half-essential bridge $\{r, s\}$ it holds that $\{u, v\} \in \text{comp}_{G+F}(r)$ and $\{x, y\} \in \text{comp}_{G+F}(s)$. To find such a half-essential bridge, we remove a half-essential bridge temporarily and test if u and x are connected. This has to be done at most $c - 1 \leq |V|$ times.

Now we can apply RewireC. Iteratively applying RewireC is similar to iteratively applying RewireA with the additional searching for an appropiate half-essential bridge. So the runtime for iteratively applying RewireC is $O(|V|(|V| + |E|) \log^2 |V|)$.

For RewireD we need a half-essential bridge and a non-bridge. Because all edges of F are in the same connected component and for every half-essential bridge $\{u, v\}$ there is only one connected component out of $\text{comp}_{G+F-\{u,v\}}(u)$ and $\text{comp}_{G+F-\{u,v\}}(v)$ with edges of F we just need to find a non-bridge and a half-essential bridge where the non-bridge is in $\text{comp}_{G+F}(u)$ with u being the over-saturated edge of the half-essential bridge. We again maintain a list of potential non-bridges of F and a list of half-essential bridges of F. The non-bridge can be found as in RewireA and for the half-essential bridge, we iterate through the list until we find a half-essential bridge $\{u, v\}$ with u being the over-saturated vertex and the non-bridge in $\text{comp}_{G+F}(u)$. To test for a half-essential bridge

$\{u, v\}$ with over-saturated vertex u if the non-bridge is in $\text{comp}_{G+F}(u)$ we temporarily remove $\{u, v\}$ and test if u and a vertex of the non-bridge are connected. If this is the case we can apply RewireD. If this is not the case, we can remove the half-essential bridge from the list, because there is only one connected component out of $\text{comp}_{G+F-\{u,v\}}(u)$ and $\text{comp}_{G+F-\{u,v\}}(v)$ with edges of F, $\{u, v\}$ cannot be used in RewireD. So after every test there is one less half-essential bridge in the list. Therefore there are at most $|V|$ tests needed. Hence the runtime of iteratively applying RewireD is $O((|V| + |E|) \log^2 |V|)$.

These observations lead to the following theorem.

Theorem 3.13. ConnCore *can be solved in* $O(k|V| \min\{|E^c| \log(|V|), |V|^2\})$ *time.*

4. Core Augmentation for Subsets

After we have examined the problem of getting all vertices of a graph into the k-core, we now consider a problem regarding a subset of all vertices in a graph. This problem is Fixed Subset k-Core Augmentation of augmenting a graph $G = (V, E)$, such that a given subset $W \subseteq V$ is in the k-core. We show that this is NP-hard for $k \geq 3$ and solvable in polynomial time for $k \leq 2$.

Problem 4 (Fixed Subset k-Core Augmentation)**.** *Given a graph $G = (V, E)$, a subset $W \subseteq V$ of vertices and a natural number k, find $F \subseteq V^2$ of minimum size such that for each $v \in W$ it holds that $coreness_{G+F}(v) \geq k$.*

We generally cannot simplify this problem to a degree problem like Core Augmentation. We will show that this problem is NP-hard, and in fact W[2]-hard, for $k \geq 3$ by reducing Dominating Set to Fixed Subset k-Core Augmentation. Furthermore Fixed Subset k-Core Augmentation is not approximable within a factor of $(1-\epsilon)\ln|V|$ for any $\epsilon > 0$ unless NP $\subset$ DTIME. In the case of $k \leq 2$ we show, that Fixed Subset k-Core Augmentation can be solved in polynomial time.

4.1 Augmenting to the 3-core

In this section we reduce Dominating Set to Fixed Subset 3-Core Augmentation.

Problem 5 (Dominating Set)**.** *Given a graph $G = (V, E)$ find a subset $D \subset V$ such that every vertex $v \in V$ is either in D or adjacent to a vertex in D.*

Let $G = (V, E)$ be an arbitrary instance of Dominating Set. To reduce Dominating Set to Fixed Subset k-Core Augmentation we construct an instance $G' = (V', E')$ of Fixed Subset k-Core Augmentation and show that a solution to G' implies a solution to G. The reduction works by constructing two gadgets for every vertex in V. One is used to mark vertices as a part of the corresponding dominating set and the other is used to check if the vertex is marked or adjacent to a marked vertex. We denote the former by *marking gadget* and the latter by *checking gadget*. We construct an instance of Fixed Subset Core Augmentation by linking the marking gadget of a vertex to the checking gadgets of every adjacent vertex including itself. The gadgets are constructed such that a checking gadget is in the k-core if and only if a linked marking gadget has an added edge

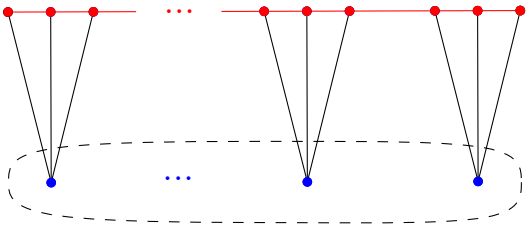


Figure 4.1: One marking gadget (red) connected with multiple checking gadgets (blue); the circled vertices are in the subset of vertices to be in the 3-core

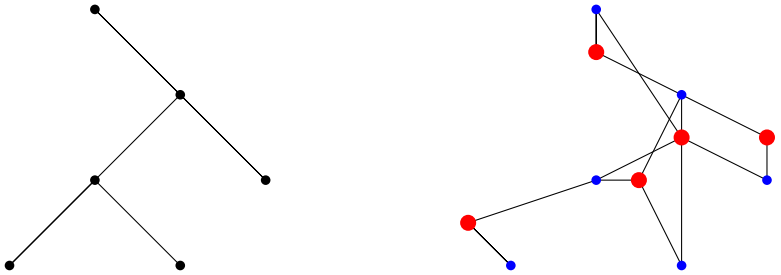


Figure 4.2: An example for a reduction from Dominating Set to Subset Core Augmentation. (a) A graph as an input to Dominating Set. (b) A graph as an input to Subset Core Augmentation. Blue vertices represent checking gadgets, red vertices represent marking gadgets and an edge represent four edges as illustrated in Figure 4.1.

$e \in F$ in a solution F to Fixed Subset k-Core Augmentation or if F can be rewired to be such a solution.

The marking gadget for a vertex v is a path of $3(\deg(v)+1)$ vertices. To be more precise, it consists of the vertices a_i, b_i, c_i for $i \in \{0..\deg(v)\}$ for every vertex $v \in V$. The edges are $\{a_i, b_i\}, \{b_i, c_i\}$ for $i \in \{0, \ldots, \deg(v)\}$ and $\{c_i, a_{i+1}\}$ for $i \in \{0, \ldots, \deg(v)-1\}$. The vertices a_0 and $c_{\deg(v)}$ are called *ends* of the marking gadget. The checking gadget is a single vertex z. To connect a marking gadget of a vertex v with the checking gadgets of the adjacent vertices, the adjacent vertices are numbered 1 to a. Let z_i denote the single vertex of the checking gadget of the vertex with number i. The edges used to connect the marking gadget with the checking gadgets are $\{a_i, z_i\}$, $\{b_i, z_i\}$ and $\{c_i, z_i\}$ for $i \in \{1, \ldots, a\}$. Furthermore, the marking gadget of v is connected to the checking gadget of v by the edges $\{a_0, z_0\}$, $\{b_0, z_0\}$ and $\{c_0, z_0\}$, where z_0 denotes the vertex of the checking gadget of v. We call the resulting graph $G' = (V', E')$. The subset $W' \subset V'$ of vertices to be augmented to the k-core consists of the vertices of all checking gadgets. In the remainder of this section we will refer to the marking gadgets that are connected to a checking gadget by the three edges in the construction above as marking gadgets *linked* to the checking gadget. A marking gadget linked to multiple checking gadgets can be seen in Figure 4.1. An example for the new graph G' as input to Fixed Subset 3-Core Augmentation corresponding to a graph G as input to Dominating Set can be seen in Figure 4.2.

To prove the correctness of the reduction, we will first show how a solution to the Dominating Set instance G can be derived in linear time from a solution to the Fixed Subset 3-Core Augmentation instance G' that has a special property. Second, we will show that every minimal solution of G can be transformed to such a solution without losing minimality.

Lemma 4.1. *Given an instance $G = (V, E)$ of* Dominating Set*, let $G' = (V', E')$ be an instance of* Fixed Subset 3-Core Augmentation *constructed as above. It holds that F is a solution to G' of a size less than d if and only if it induces an solution D of G of a size less than d.*

Proof. First we will show how a solution to G can be easily derived from a special solution to G'. Let F be a solution to G' with the property that every edge $e \in F$ of the solution is an edge between the ends of a marking gadget. We call these marking gadgets *saved marking gadgets*. The vertices of each saved marking gadget of a vertex $v \in V$ and all checking gadgets linked to it are in the 3-core, because the induced subgraph of these vertices in $G + F$ is 3-regular. The vertices of each marking gadget without an edge between the ends are only in the 2-core. A checking gadget without a linked saved marking gadget is in the 2-core but not in the 3-core, because all adjacent vertices are also in the 2-core and not in the 3-core. Recall, that the instance G' is constructed such that the vertices we have to augment into the 3-core are the vertices of the checking gadgets. Hence, every checking gadget of a vertex $v \in V$ is linked to at least one saved marking gadget. Let $D \subseteq V$ be the set of vertices in G corresponding to the saved marking gadgets in $G' + F$. Because every checking gadget of a vertex $v \in V$ is linked to a saved marking gadget of a vertex $w \in D$ and the gadgets in G' are linked if and only if v and w are adjacent or the same vertex, for every vertex $v \in V \setminus D$ it holds that it is adjacent to a vertex in D. So D is a solution to Dominating Set by definition. Note that this also works in the other direction, i.e., given a dominating set D, we construct F by chosing an edge between the ends of the marking gadget of each vertex $w \in D$ of a dominating set D of G. In both directions the size of the solutions stay the same. Hence, $|D| = |F|$.

Next, we show how an arbitrary minimal solution to G' can be transformed into a solution of the aforementioned specific form without increasing its size, i.e., the edges of the solution are edges between the ends of a marking gadget. We will temporarily allow loops and parallel edges in this construction. Let F be an arbitary solution to G'. We assume without loss of generality that vertices incident to edges of F are in the 3-core of $G' + F$. Otherwise edges of F incident to vertices that are not in the 3-core would be superflous, i.e., they are ignored in the calculation of the 3-core.

We first change the solution such that all vertices of marking gadget that are incident to edges of the solution are ends of a marking gadget. We first analyse the amount of incidences in a marking gadget. If only one vertex of a marking gadget is incident to exactly one edge of F and no other vertex of the marking gadget is incident to edges of F, the marking gadget is in the 2-core but not in the 3-core and e is superfluous. Hence we can assume that there are either zero or at least two incidences to edges of F in a marking gadget. We now consider a marking gadget with at least two incidences to edges of F. Let r and s be the two ends of the marking gadget. Let $I \subseteq F$ be the edges of F that are incident to vertices of the marking gadget. We choose an arbitrary edge $\{u, v\}$ of I with u being a vertex of the marking gadget and replace it by $\{r, v\}$. Every other edge $\{x, y\}$ of I with x being a vertex of the marking gadget is replaced by $\{s, y\}$. If a replacement produces a loop $\{r, r\}$ or $\{s, s\}$, we replace this loop by $\{r, s\}$. After these replacements, the marking gadget is in the 3-core and no incidences outside of the marking gadget are changed. Hence this replacement does not invalidate the solution. We call the new solution that results from exchanging edges for all marking gadgets in this way F_1. In this changed solution F_1 there are either no vertices in a marking gadget incident to edges of F_1 or both ends are incident to edges of F. Hence, either a whole marking gadget is in the 3-core or it is not in the 3-core at all.

Now we exchange the edges of F_1 incident to marking gadgets such that there are only edges incident to both ends of a single marking gadget left, i.e., we create saved marking

gadgets to get a solution of the special form from above. Let r and s be two vertices incident to edges $\{r, u\}$ and $\{s, v\}$ in F_1 of a not already saved marking gadget. Due to the steps above r and s are the ends of the marking gadget. We replace these edges by $\{r, s\}$ and $\{u, v\}$. Note that this replacement may produce a loop $\{u, u\}$. If this is the case and u is a vertex of another marking gadget, we exchange this loop for an edge between the ends of the marking gadget containing u. If u is a vertex of a checking gadget we choose an arbitrary linked marking gadget and replace the loop by an edge between the ends of the chosen marking gadget. This ensures that all involved vertices r, s and $u = v$ remain in the 3-core in the changed solution. If $u \neq v$, these vertices also remain in the 3-core, which can be seen by examining a subgraph where every vertex as a degree of at least 3 and that contains r, s, u and v. Such a subgraph exists, because these vertices are in the 3-core. The replacement does not change the degrees of the vertices in this subgraph. So the involved vertices remain in the 3-core after the replacement. This replacement saves at least one marking gadget with the edge $\{r, s\}$. We do this iteratively until all marking gadgets with incident edges of the solution are saved marking gadgets. This new solution is called F_2. Note that there may be left-over augmented edges that are incident to ends of the same marking gadget, e.g., if there are more than two incidences to augmented edges in a marking gadget. Then such an edge is an edge between two saved marking gadgets. Hence it is superfluous and not part of a minimal solution. Furthermore, if these replacements produce parallel edges, the original solution uses more edges than needed, i.e., it is not minimal.

Now we replace edges between marking gadgets and checking gadgets from the solution such that there are only edges between marking gadgets, respectivley edges between checking gadgets, left in the solution. Let $e = \{u, v\} \in F_2$ be an edge of the solution between a vertex u of marking gadget M and a vertex v of a checking gadget C. Due to the construction above, each marking gadget is either saved or has no incidences to edges of the solution. Hence the marking gadget M is saved and does not need the incidence of e and we replace e by an edge between the ends of an arbitrary marking gadget linked to C. Applying this replacements iteratively results in a new solution F_3. It holds that there are no edges between checking gadgets and marking gadgets in F_3. All marking gadgets that have incident edges of F_3 are saved marking gadgets.

Furthermore, we show that there are no edges of F_3 between vertices of checking gadgets in a minimal solution F_3. Let $\{u, v\}$ be an edge of F_3 between vertices of two checking gadgets. If both checking gadgets are linked to a marking gadget in the 3-core, the edge $\{u, v\}$ is superflous, i.e., not part of a minimal solution. If only one checking gadget is linked to a marking gadget in the 3-core, we can replace $\{u, v\}$ by an edge between the ends of the marking gadget that is not yet in the 3-core. So we can assume without loss of generality, that all checking gagdets incident to edges of F_3 between two checking gadgets are not linked to marking gadgets in the 3-core. Let $P \subset W$ be the vertices of the checking gadgets without linked marking gadgets in the 3-core. There are at least $|P| \cdot \frac{3}{2}$ edges needed to bring all vertices in P into the 3-core. This is worse than just selecting a linked marking gadget for every vertex in P and add an edge between the ends, which only needs $|P|$ edges. So we can replace edges of F_3 between vertices of checking gadgets by edges between ends of marking gadgets to construct a new Solution F_4. Every edge in this new solution F_4 is an edge between two ends of marking gadgets.

So we have transformed an arbitrary minimal solution F to a solution F_4 of our special form, containing only edges that are needed to save a marking gadget. This solution F_3 corresponds directly to a solution to G of size F_4, our initial instance of Dominating Set, as seen above. □

We have reduced Fixed Subset 3-Core Augmentation to Dominating Set in po-

lynomial time. Note that the dominating set constructed by this reduction is not larger than the given solution of Fixed Subset 3-Core Augmentation. Hence the approximation constraints of Dominating Set are carried over to Fixed Subset 3-Core Augmentation. Dominating Set is NP-hard and futhermore W[2]-hard[4]. Furthermore, it is NP-hard to approximate within a factor of $(1-\epsilon)\ln n$ for any $\epsilon > 0$ unless NP $\subset$ DTIME($|V|^{\log\log|V|}$) [2] [6]. Therefore Fixed Subset 3-Core Augmentation is NP-hard and in fact W[2]-hard. This result is recorded in the following theorem.

Theorem 4.2. Fixed Subset 3-core Augmentation *is W[2]-hard and NP-hard to approximate within a factor of* $\log n$.

4.2 Augmentation of subsets for $k < 3$

In the previous section we have examined the problem of bringing a given subset into the k-core with $k \geq 3$. Now we look upon the cases $k = 1$ and $k = 2$.

Augmenting graphs to bring a fixed subset into the 1-core is easy. Let $G = (V, E)$ be the input graph and let $W \subseteq V$ be the subset of vertices that is to be augmented into the 1-core. Note that the only vertices not in the 1-core are the vertices with a degree of zero. Let F denote the set of augmented edges. For a vertex $w \in W$ it holds that, if $\deg_G(w) \geq 1$ it needs no additional incidences in F because it is already in the 1-core. We assume without loss of generality, that W only includes vertices that are not already in the 1-core. If $\deg_G(w) = 0$, the vertex w needs one incidence in F. So a solution F needs at least $\lceil |W|/2 \rceil$ edges. Hence we we can construct a minimal solution F by iteratively choosing two vertices $v, w \in W$ with $\deg_G(v) = \deg_G(w) = 0$ and adding an edge between these vertices. If there is still a vertex u with a degree of zero, we add another edge incident to u. This results in the following theorem.

Theorem 4.3. Fixed Subset 1-Core Augmentation *can be solved in linear time.*

Augmenting graphs to bring a fixed subset into the 2-core can be reduced to a simple degree-problem like CoreAugmention. Let $G = (V, E)$ be a graph and let $W \subseteq V$ be a subset of vertices. We assume without loss of generality that W consists only of vertices that are not already in the 2-core. The idea is to transform this graph into a forest with the leafs being either vertices of W or contracted biconnected components. To do this we contract the biconnected components into single vertices and then delete leaves iteratively, until all leaves are vertices of W. Afterwards we find a matching on the leaves of W of the forest and connect matched leaves to construct the solution F to Fixed Subset 2-Core Augmentation. However, this does not work in certain cases when the connected components get to small, because we lose the information that a connected component of size three can be brought into the 2-core by a single edge. We solve these small cases independently from the remainder of the graph.

We now explain this algorithm in detail. Consider a connected component containing three vertices with at least one vertex $v \in W$. This component can be augmented by a single edge, making it a cycle, thereby bringing the connected component into the 2-core. Otherwise two edges are needed to lift the connected component into the 2-core. In both cases there are two incidences needed to bring the vertices of the connected component into the 2-core. Hence the single edge used is optimal because it uses the least amount of incidences.

Now we consider the connected components with two or less vertices that include at least one vertex $v \in W$. Let n be the number of such components. To bring each of these vertices into the 2-core, two incidences are needed. If there are at least two such vertices, we can

connect these building a cycle using n added edges. Because $2n$ incidences are needed, these edges are used optimally. This does not work if there is only one such connected component. If this is the case we first complete the remainder of the augmentation and then replace one edge $\{u, v\}$ of the solution by two edges $\{u, x\}$ and $\{v, y\}$ with x and y being two vertices of said connected component. Note that the connected component can consist of a single vertex. In this case x is the same vertex as y. If there is no augmented edge, i.e., the remainder of the graph is already in the 2-core, we add two edges to bring the connected component into the 2-core. This uses the least amount of incidences. Hence it is optimal.

We now consider the graph without connected components containing three or less vertices. We furthermore ignore connected components without vertices in W. We contract each biconnected component to a single vertex and mark it as being a previously biconnected component. After the biconnected components are contracted, we have a forest. We iteratively delete each leaf that is neither in W nor marked in the step above, in a connect component of at least four vertices with its neighbour. If this produces connected components consisting of three vertices, there are two cases whether there are marked vertices in the connected component. If there are no marked vertices, we treat the component like in the special case above, i.e., we add an edge to the solution to make a 3-cycle out of the connected component. If there are marked vertices in the connected component we furthermore delete the leaves that are neither in W nor marked. Note that his produces connected components with a minimum size of two, because there is at least one marked vertex and one vertex of W in the component. The graph that is constructed by this contractions and deletions is called G' Let $W' \subseteq V$ be the set of vertices that are in W and leaves of the forest.

After this iterative deletions of leaves is done we have the following situation. The graph G' is a forest with its leaves being either vertices of W or contracted vertices that were biconnected components in the original graph. We now find a matching on the vertices of W' and connect matched leaves to construct a solution F'. Note that no vertex of W' is incident to another vertex of W'. Hence we can find the needed matching greedily, similar to the case of $k = 1$ above. If there is a single unmatched vertex v of W' we add an edge between v and an arbitrary other leaf that is no neighbour of v or an arbitrary marked vertex if there is no such leaf. Note that an augmenting edge between a leaf of W and a marked vertex is only needed if this leaf is the only vertex in W and that this edge might be a parallel edge. We also add the augmenting edges of the smaller special cases to the solution F'. The only vertices v in $G' + F'$ with $\deg(v) < 2$ are marked vertices.

We transfer this solution to the original graph G. If there is an edge $\{u, v\}$ of F' incident to a marked vertex u we choose an arbitrary vertex x of the original biconnected component to get an augmenting edge $\{u, x\}$ for the original graph. Every other edge $\{u, v\}$ between unmarked vertices remains the same in the solution for the original graph. We call this solution F. It can be easily seen that this is a valid solution to Fixed Subset 2-Core Augmentation by iteratively removing the vertices with a degree one to calculate the 2-core of $G + F$. The remaining 2-core is G' with the difference that the biconnected components are not contracted. Hence all vertices of G' are in the 2-core.

To show that this solution is of minimum size we count the incidences that are needed for a solution. Let F^* be an arbitrary solution. For every vertex v in W it holds that it is on a path between two vertices that are either in the 2-core in G or incident to an edge of F^*, respectively. If this is not the case, v is not in the 2-core in $G+F^*$. Note that this path can be of length zero. This holds in particular for vertices in W'. Every vertex v of W' is part of an induced subtree that consists of itself and vertices that were deleted in the algorithm above. Let W_v be the set of vertices that consists of v and the vertices that induce a subtree

of maximum size without using vertices of G', i.e., we only use vertices that were deleted in the algorithm above. Because of the observation above at least one vertex of W_v has to be incident to an edge of F^*. Every vertex v of W' has its distinct subtree W_v, i.e, for $u, v \in W', u \neq v$ it holds that $W_u \cap W_v = \emptyset$. Hence for every vertex of W' at least one incidence is needed in a solution to Fixed Subset 2-Core Augmentation. So we need at least $\lceil |W'|/2 \rceil$ edges in a solution. We furthermore need the edges for the special cases of connected component of size three or less. The solution F that is constructed by our algorithm uses this minimum amount of edges.

We now examine the runtime of the algorithm. We can find and contract the biconnected components of G in linear time, using an algorithm due to Tarjan [14]. We can find and delete a leaf in linear time. We can delete the leaves in linear time. We can find the matching in linear time similar to the algorithm for the case of $k = 1$. Overall the algorithm needs $O(|V|)$ time. This leads to the following theorem.

Theorem 4.4. Fixed Subset 2-Core Augmentation *can be solved in* $O(|V|)$ *time.*

5. Conclusion

The core structure of a graph is an interessting measurement that is used in the analysis of many different networks. We examined how we can change the core structure of a given graph by adding edges. This can be used to make certain vertices more "important" than others. In this work we have studied two problems regarding the augmentation of the core structure of a graph.

The first problem Core Augmentation of lifting all vertices of a graph into the k-core was solved simply by finding a degree constrained subgraph on the complement. The more complicated case ConnCore was to find a minimum augmentation that solves CoreAug while also ensuring that the augmented graph is connected. It proved to be feasible to start with an arbitrary solution to Core Augmentation and then optimise this solution with different rewire operations piece by piece to ensure the connectivity. Each of the rewiring operations reduces the number of connected components while not invalidating a solution. We have shown that this produces an optimal solution to ConnCore

This could be done, because the problem could be reduced to a local degree problem. If every vertex has a degree greater than or equal k, every vertex is in the k-core. If we consider only subsets of vertices that are to be lifted into the k-core, these vertices obviously have to have a degree greater than or equal k, but it is difficult to say which other vertices have to be in the k-core, i.e., where we have to insert augmenting edges that are not incident to vertices of the given subset, such that the set of augmenting edges is minimal. Hence, the main difference between lifting all vertices into the k-core versus a subset of vertices is that the non-locality of the coreness of a vertex plays a huge role in the latter. It does not suffice to just consider the vertices of the subset and small changes in other parts of the graph can change the coreness of a vertex.

The second problem Fixed Subset k-Core Augmentation regarding only subsets of vertices that are to be lifted into the k-core is NP-hard for $k >= 3$. This was proven by a reduction from Dominating Set. Furthermore, the reduction has shown that Fixed Subset 3-Core Augmentation is W[2]-hard with respect to the size of the solution and hard to approximate within a factor of $\log n$. Fixed Subset k-Core Augmentation is solvable in linear time for $k \leq 3$.

The interessting question remains what happens if we have constraints for the subset of vertices that we want to lift into the k-core. For example the problem of lifting a single vertex into the k-core. Other interessting problems regard a maximum number of edges we can use for augmentation and optimise the core structure as much as possible. For example:

What is the maximum coreness we can lift a vertex v into, using only ℓ additional edges. And what happens, if we want to augment planar graphs such that they stay planar and satisfy different core properties?

Literaturverzeichnis

[1] Vinay Aggarwal, Anja Feldmann, Marco Gaertler, Robert Görke, and Dorothea Wagner. Analysis of overlay-underlay topology correlation using visualization. In *Proceedings of the 5th IADIS International Conference WWW/Internet Geometry*, pages 385 – 392, 2006.

[2] R. Bar-Yehuda and S. Moran. On approximation problems related to the independent set and vertex cover problems. *Discrete Applied Mathematics*, 9(1):1 – 10, 1984.

[3] P. O. Bex, Goos Kant, Goos Kant, Hans L. Bodlaender, and Hans L. Bodlaender. Planar graph augmentation problems. In *In Proc. 2nd Workshop Algorithms Data Struct*, pages 286–298. Springer-Verlag, 1991.

[4] Michael R. Downey, Rod G. ; Fellows. *Parameterized complexity.* Monographs in computer science. Springer, New York, 1999.

[5] Kapali P. Eswaran and R. Endre Tarjan. Augmentation problems. *SIAM J. Comp.*, 5(4):653–665, 1976.

[6] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45:634–652, July 1998.

[7] Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 448–456, New York, NY, USA, 1983. ACM.

[8] Robert Görke. *An Algorithmic Walk from Static to Dynamic Graph Clustering.* PhD thesis, Karlsruher Institut für Technologie, 2010.

[9] Jacob Holm and Kristian de Lichtenberg. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001.

[10] Hiroshi Nagamochi and Toshihide Ibaraki. Graph connectivity and its augmentation: applications of ma orderings. *Discrete Applied Mathematics*, 123(1-3):447 – 472, 2002.

[11] Ignaz Rutter and Alexander Wolff. Augmenting the connectivity of planar and geometric graphs. *Electronic Notes in Discrete Mathematics*, 31(0):53 – 56, 2008.

[12] L. Subramanian, S. Agarwal, J. Rexford, and R.H. Katz. Characterizing the internet hierarchy from multiple vantage points. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 618 – 627 vol.2, 2002.

[13] Robert Endre Tarjan. A note on finding the bridges of a graph. *Inf. Process. Lett.*, 2:160–161, 1974.

[14] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comp.*, 1(2):146–160, 2001.

[15] Toshimasa Watanabe and Akira Nakamura. Edge-connectivity augmentation problems. *Journal of Computer and System Sciences*, 35(1):96 – 144, 1987.

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