To appear in Networks, 2009

Kernelization and Complexity Results for Connectivity Augmentation Problems*

Jiong Guo† and Johannes Uhlmann‡

Abstract

Connectivity augmentation problems ask for adding a set of at most k edges (called *links*) whose insertion makes a given graph satisfy a specified connectivity property, such as bridge-connectivity or biconnectivity. A bridge-connected (biconnected) graph is a connected graph that does not possess an edge (a vertex) whose removal results in a disconnected graph. We show that, for bridge-connectivity and biconnectivity, the respective connectivity augmentation problems admit problem kernels with $O(k^2)$ vertices and links. Moreover, we study partial connectivity augmentation problems, naturally generalizing connectivity augmentation problems. Here, we do not require that, after adding the edges, the entire graph should satisfy the connectivity property, but a large subgraph. In this setting, three polynomial-time solvable connectivity augmentation problems behave differently, namely, the *partial* bridge-connectivity augmentation problem and the *partial* biconnectivity augmentation problem remain polynomial-time solvable whereas the *partial* strong connectivity augmentation problem becomes W[2]-hard with respect to k.

Keywords: Combinatorial problem, NP-complete problem, graph algorithm, fixed-parameter (in)tractability, data reduction.

1 Introduction

Connectivity augmentation problems on undirected and directed graphs have as input a graph $G = (V, E)$, a set E' of edges, and a non-negative integer k, and ask for a set E'' of at most k edges from E' such that $(V, E \cup E'')$ satisfies

*A preliminary version appeared in Proc. of the 10th Workshop on Algorithms and Data Structures (WADS 07), volume 4619 in LNCS, pages 484-495, Springer 2007.

†Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany. Email: jiong.guo@uni-jena.de. Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4, and research project DARE (data reduction and problem kernels), GU 1023/1.

‡Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany. Email: johannes.uhlmann@uni-jena.de. Supported by the Deutsche Forschungsgemeinschaft (DFG), research projects OPAL (optimal solutions for hard problems in computational biology), NI 369/2, and PABI (parameterized algorithmics for bioinformatics), NI 369/7.

a specified connectivity property. The edges in E' are called the *links*. Eswaran and Tarjan [5] introduced connectivity augmentation problems and described their numerous applications.

We use $G = (V, E)$ and $D = (V, A)$ to denote undirected and directed graphs. A *path* from vertex u_1 to vertex u_l in $G = (V, E)$ (or $D = (V, A)$) is a sequence of edges $\{u_1, u_2\}$, $\{u_2, u_3\}$, ..., $\{u_{l-1}, u_l\}$ (or arcs (u_1, u_2), (u_2, u_3), ..., (u_{l-1}, u_l)). A *cycle* is a path with $u_1 = u_l$.

A vertex u in an undirected graph is called a *cut-vertex* if there are two vertices v, w with $v \neq u$ and $w \neq u$ such that every path from v to w contains u. If an undirected graph G is connected, has at least three vertices, and has no cut-vertex, then G is *biconnected*. A *bridge* in an undirected graph is an edge $\{u, v\}$ that is the only path between u and v. If G is connected and has no bridge, then G is *bridge-connected*. A directed graph $D = (V, A)$ is *strongly connected* if, for all pairs of vertices u and v, there is a path from u to v. The *connected (biconnected, bridge-connected, strongly connected) components* of a graph are its maximal connected (biconnected, bridge-connected, strongly connected) subgraphs. We consider here two connectivity augmentation problems, namely, Bridge-Connectivity Augmentation (BCA) and Biconnectivity Augmentation (BIA), where we are asked to add at most k links to make the given graph bridge-connected or biconnected.

There is a long history of research dealing with BCA and BIA starting in 1976 with the work of Eswaran and Tarjan [5]. They showed that, in the case that E' is *complete*, that is, graph (V, E') is a complete graph, both problems are polynomial-time solvable. In 1981, Frederickson and JáJá [10] proved the NP-completeness of both problems if E' is *incomplete*. Motivated by the NP-completeness, the polynomial-time approximability of the optimization versions of these problems has been extensively studied in the literature. Frederickson and JáJá [10] gave polynomial-time factor-2 approximation algorithms for both problems. For BCA, Nagamochi [16] improved the approximation factor to 1.875. Later, Even et al. [7] presented a polynomial-time factor-1.8 approximation algorithm for BCA.[1] In case of BIA, Khuller and Thurimella [13] improved the running time of the factor-2 approximation algorithm in [10]. On the negative site, Kortsarz et al. [14] proved that there exists an $\epsilon > 0$ for which it is NP-hard to approximate BCA and BIA within a factor of $1 + \epsilon$.

Concerning the parameterized complexity of these problems, we are only aware of one result due to Nagamochi [16]. Since the bridge-connected components of a graph form a tree, we may assume that the input graph of a BCA-instance is a tree by contracting these components. Nagamochi [16] showed that BCA is fixed-parameter tractable with respect to the number of leaves ℓ in the given tree. More precisely, there is an algorithm solving this problem in $O(\ell^{\ell+1} \log \ell \cdot (|V| + |E'|))$ time. Since the number of leaves provides a lower bound on the solution size k, BCA is fixed-parameter tractable with respect to k.

[1] In the conference version Even et al. [6] presented and sketched a proof of a polynomial-time factor-1.5 approximation algorithm for BCA.

Kernelization is a core concept of parameterized algorithmics [11, 12, 17]. However, nothing has been known so far concerning the kernelization of BCA and BIA. Kernelization is considered as one of the theoretically and practically most interesting techniques of parameterized algorithmics [8, 11, 12, 17]. Informally speaking, kernelization means polynomial-time preprocessing (data reduction) with provable performance guarantee. More specifically, a *kernelization* is a polynomial-time algorithm that transforms a given instance I with parameter k of a problem P into a new instance I' with parameter $k' \leq k$ of P such that the original instance I is a yes-instance with parameter k if and only if the new instance I' is a yes-instance with parameter k' and $|I'| \leq g(k)$ for a function g. The instance I' is called the *problem kernel*. That is, the goal is to derive in polynomial-time an instance *equivalent* to the original one, but whose size is bounded by a function only depending on the parameter k. Next, we provide some arguments for the usefulness of kernelization. First of all, there might be input instances for which preprocessing yields already an optimal solution for the problem or where preprocessing results in an instance that is small enough such that the use of simple exhaustive search is affordable (in terms of running time). Thus, in general, there is no doubt that polynomial-time data reduction or preprocessing is useful when dealing with NP-hard problems. However, although preprocessing is ubiquitous, so far only the kernelization technique of parameterized algorithmics provides a theoretical framework for the mathematical analysis of provable performance guarantees of a proposed data reduction. Although kernelization has its origins in parameterized algorithmics, it is a far more general concept not restricted to this area. Since kernelization results imply efficient preprocessing algorithms, they are useful for *any* problem solving approach, be it exact, approximative, or heuristic. For example, after data reduction the instance size might be small enough to exploit a search-tree strategy or an Integer Linear Program to solve the problem exactly. Moreover, it is conceivable that for many instances better approximative solutions are found after preprocessing. The practical usefulness of kernelization has been proven by successful empirical studies. For instance, experimental work based on kernelization has been done for Dominating Set [1, 15] and Cluster Editing [2, 3].

Our contributions. We complement the result of Nagamochi with a problem kernel consisting of $O(k^2)$ vertices and links for BCA and BIA. To this end, we provide several polynomial-time data reduction rules for BCA and BIA. Furthermore, we study *partial* connectivity augmentation problems, a natural generalization of the connectivity augmentation problems, where we have as input a graph $G = (V, E)$, a set E' of links, and two non-negative integers k, Φ and ask for a subset E'' of E' with $|E''| \leq k$ such that graph $(V, E \cup E'')$ has a subgraph that contains at least Φ vertices and satisfies the given connectivity property. Clearly, if $\Phi = |V|$, then we have the connectivity augmentation problems. We consider three partial connectivity augmentation problems, Partial Bridge-Connectivity Augmentation, Partial Biconnectivity Augmentation, and Partial Strong Connectivity Augmentation. For all three con-

nectivity properties, their corresponding non-partial connectivity augmentation problems are solvable in polynomial-time if E' is complete [5]. In Sect. 5, we show that Partial Bridge-Connectivity Augmentation and Partial Biconnectivity Augmentation with a complete link set remain polynomial-time solvable but Partial Strong Connectivity Augmentation with a complete link set is W[2]-hard (and NP-hard), that is, it is very unlikely that this problem is fixed-parameter tractable with respect to the parameter k.[2] Some technical details omitted from this work for the sake of readability can be found in the second author's diploma thesis [20].

2 Preliminaries

Parameterized complexity is a two-dimensional framework for studying the computational complexity of problems [4, 9, 17]. One dimension is the input size n (as in classical complexity theory) and the other one the *parameter* k (usually a positive integer). A problem is called *fixed-parameter tractable* (fpt) if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction rules*, often yielding a *kernelization*. Herein, the goal is, given any problem instance I with parameter k, to transform it in polynomial time into a new instance I' with parameter k' such that the size of I' is bounded from above by some function only depending on k, $k' \leq k$, and (I, k) is a yes-instance if and only if (I', k') is a yes-instance. A data reduction rule is *correct* if the new instance after an application of this rule is a yes-instance if and only if the original instance is a yes-instance. Throughout this paper, we call a problem instance *reduced* if the corresponding data reduction rules cannot be applied anymore.

A formal framework to show *fixed-parameter intractability* was developed by Downey and Fellows [4] who introduced the concept of *parameterized reductions*. A parameterized reduction from a parameterized problem L to another parameterized problem L' is a function that, given an instance (I, k), computes in $f(k) \cdot n^{O(1)}$ time an instance (I', k') (with k' only depending on k) such that $(I, k) \in L \Leftrightarrow (I', k') \in L'$. The basic complexity class for fixed-parameter intractability is W[1] as there is good reason to believe that W[1]-hard problems are not fixed-parameter tractable [4]. There is a whole hierarchy W[1] $\subseteq$ W[2] $\subseteq$ W[3] $\subseteq \ldots$ W[P] of presumable fixed-parameter intractability, called the W-Hierarchy. A parameterized problem P is called W[t]-*hard* if all problems in W[t] can be reduced to P by means of a parameterized reduction. Note that the notion of W[t]-hardness is defined analogously to NP-hardness and that in parameterized complexity the classes of the W-Hierarchy play the role of the class NP in classical complexity.

Throughout this paper, we set $n := |V|$ for a given graph $G = (V, E)$ and $m := |E'|$ for a given link set E'. For a graph G, we also use $V(G)$ and $E(G)$

[2] There exists a whole hierarchy of presumable fixed-parameter intractability and the first two levels of this hierarchy are captured by the complexity classes W[1] and W[2] (see Sect. 2).

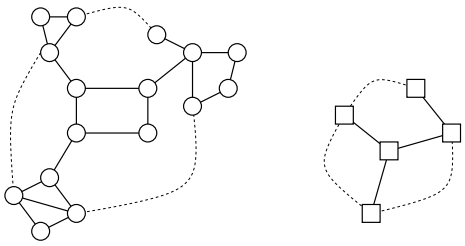

Figure 1: Example of contracting bridge-connected components of a given graph. The links are drawn as dashed lines.

to denote its vertex and edge set, respectively. The *open neighborhood* $N(v)$ of a vertex $v \in V$ is the set of vertices that are adjacent to v. The *degree* of a vertex v, denoted by $\deg(v)$, is the size of $N(v)$.

3 The Bridge-Connectivity Augmentation Problem

The main result of this section is a data reduction for Bridge-Connectivity Augmentation (BCA) that leads to a quadratic-size problem kernel. Given an instance of BCA, we can assume that the input graph G is a tree [5, 10, 7]: Each bridge-connected component of $G = (V, E)$ can be contracted into a single vertex by contracting all edges in this component, resulting in a tree. The set of links has to be adapted accordingly. The contraction of the bridge-connected components can be done in $O(|V| + |E|)$ time [19]. See Fig. 1 for an example. Hence, in the following, the input instance is always denoted by T.

In contrast to the tree edges, denoted by $\{u, v\}$, we denote links by (u, v). We use $p_{u,v}$ to denote the uniquely determined path between two vertices u and v in T. In the course of the data reduction process, if a link $(u, v) \in E'$ is added to a solution E'', then the vertices from the path $p_{u,v}$ form a bridge-connected component and we contract all edges in this component, obtaining a tree again. We say a link (u, v) *covers* an edge e if e lies on $p_{u,v}$. For an edge $e \in E$, we use $l(e)$ to denote the set of links covering e. A link $(u, v) \in E'$ is called a *shadow* if there exists a link $(x, y) \in E'$ with $V(p_{u,v}) \subsetneq V(p_{x,y})$, that is, the path between u and v is entirely contained in the path between x and y and $\{u, v\} \neq \{x, y\}$. See Fig. 2 for an illustration of the shadow definition. Let $N_{E'}(u) := \{v \mid (u, v) \in E'\}$. For a vertex $v \in V$ and an edge $e \in E$, we use $T_{v,e}$ to denote the subtree of $(V, E \setminus \{e\})$ that contains v.

The following observation provides the starting point for our kernelization:

Lemma 3.1 ([5])**.** *Let $L(T)$ be the set of leaves of the tree T of a BCA-instance. Every solution of this instance contains at least $|L(T)|/2$ many links, that is, $k \geq |L(T)|/2$.*

We can conclude that every yes-instance of BCA contains at most $2k$ leaves

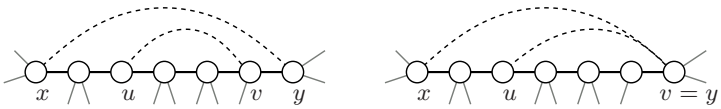

Figure 2: Illustration of the shadow definition. Herein, edges are drawn as solid lines and links are drawn as dashed lines. In both examples, link (u, v) is a shadow since the path between u and v is a proper subpath of the path between x and y.

and, also, at most $2k-1$ internal vertices with degree at least three. It remains to upper-bound the number of internal vertices of degree two. If we can bound the maximum length of the paths that consist solely of degree-2 vertices, then we can achieve an upper bound on the number of degree-2 vertices. To this end, we apply four data reduction rules. We begin with three data reduction rules that are also used in [7, 16] and whose correctness is easy to verify.

Shadow Deletion: Delete all shadows in E'.

Unit Link: If there is an edge $e \in E$ with $l(e) = \{(u, v)\}$, then contract $p_{u,v}$ and decrease the parameter k by one.

Covered Edge: If $l(e_1) \subseteq l(e_2)$ for two edges $e_1, e_2 \in E$, then contract e_2.

The polynomial running time of these rules is not hard to see [20].

Lemma 3.2 ([20])**.** *The above three rules can be executed in $O(n \cdot m^3 + n^3 \cdot m)$ time.*

Before we present the fourth data reduction rule, we show some structural properties of a BCA-instance that is reduced with respect to the above three rules. In particular, we show that, in a reduced instance, the links over a path consisting solely of degree-2 vertices have some "consecutiveness" property, which provides the basis for the fourth data reduction rule.

The first property concerns the links with a degree-2 vertex as one of their endpoints.

Lemma 3.3. *Let $(T = (V, E), E', k)$ be a reduced instance with respect to the above three rules, let $v \in V$ be a degree-2 vertex in T, and let e, e' be the edges incident to v. Then, there exists at least one link (v, x) in E' with $x \in V(T_{v,e})$ and at least one link (v, y) in E' with $y \in V(T_{v,e'})$.*

Proof. This lemma follows directly from the fact that the instance is reduced with respect to the Covered Edge rule: Suppose that there is no link between v and the vertices in $T_{v,e}$. Then, all links in $l(e')$ have to be between vertices in $T_{v,e}$ and $T_{v,e'}$ and have to cover both e and e'. This means $l(e') \subseteq l(e)$ and the Covered Edge rule would apply. □

The next lemma shows that every link has to cover at least two edges.

Lemma 3.4. *Let $(T = (V, E), E', k)$ be a reduced instance with respect to the above three rules. Then, for every link $(u, v) \in E'$, it holds that $|E(p_{u,v})| \geq 2$.*

Proof. Suppose that the lemma is not true. Then, we have a link $(u, v) \in E'$ with $\{u, v\} \in E$. Then, either (u, v) is the only link covering $\{u, v\}$ or it is a shadow of other links. In the former case, the Unit Link rule would apply; in the latter case, the Shadow Deletion rule would apply. □

The next lemma concerns the paths which consist exclusively of degree-2 vertices. More specifically, we show that the links with both endpoints on such a path P fulfill a certain "consecutiveness" property. That is, these links can be ordered in such a way that in comparison to its direct predecessor a link is shifted by exactly one vertex, that is, compared with its direct preprocessor it starts and ends at one vertex later on P.

Lemma 3.5. *Let $(T = (V, E), E', k)$ be a reduced instance with respect to the above three rules. Consider a path $P = \{u, v_1\}, \{v_1, v_2\}, \ldots, \{v_l, w\}$ in T with $deg(v_i) = 2$ for all $1 \leq i \leq l$, $deg(u) \geq 3$, and $deg(w) \geq 3$. Let E'_v denote the set of links with both endpoints in $\{v_1, v_2, \ldots, v_l\}$. If $E'_v \neq \emptyset$, then there exists an integer N with $1 \leq N \leq l-1$ such that $E'_v = \{(v_i, v_{i+N}) \mid 1 \leq i \leq l-N\}$ and there exists no link (x, y) with $x \in V(T_{u,\{u,v_1\}})$ and $y \in V(T_{w,\{v_l,w\}})$.*

Proof. If $E'_v \neq \emptyset$, then there is no link (x, y) with $x \in V(T_{u,\{u,v_1\}})$ and $y \in V(T_{w,\{v_l,w\}})$; otherwise, all links in E'_v would be shadows of (x, y) and the Shadow Deletion rule would apply.

By Lemma 3.3, there exists at least one link (v_1, x) with $x \in V(T_{v_2,\{v_1,v_2\}})$. Then, x has to be one of $v_2, \ldots, v_l$; otherwise, all links in E'_v would be shadows of (v_1, x). Furthermore, there can be only one vertex from $v_2, \ldots, v_l$ which together with v_1 forms a link; otherwise, we would have a shadow. Let $x = v_j$ with $2 \leq j \leq l$. If $j = l$, then we set $N := l - 1$. Obviously, since all links with both endpoints in $v_1, \ldots v_l$ are shadows of (v_1, v_l), we have $E_v = \{(v_1, v_l)\}$ and the lemma follows.

In the case that $j < l$, we first show that $(v_i, v_j) \in E'_v$ implies $(v_{i+1}, v_{j+1}) \in E'_v$, for $1 \leq i < j < l$. By Lemma 3.3, there exists a link $(z, v_{j+1}) \in E'$ with $z \in V(T_{v_j,\{v_j,v_{j+1}\}})$. Since all shadows are deleted by the Shadow Deletion rule and, by Lemma 3.4, every link covers at least two edges, there must exist a vertex v_r with $i < r < j$ and $z = v_r$. To show that $(v_{i+1}, v_{j+1}) \in E'_v$, it suffices to show that $r = i + 1$. Suppose that $r > i + 1$. By Lemma 3.3, we know that there is a link (v_{i+1}, z') with $z' \in V(T_{v_{i+2},\{v_{i+1},v_{i+2}\}})$. Vertex z' can only be from $\{v_{i+2}, \ldots, v_j\}$; otherwise, (z, v_{j+1}) would be a shadow of (v_{i+1}, z'). Moreover, z' cannot be a vertex with index smaller than $j+1$, since, otherwise, link (v_{i+1}, z') would be a shadow of (v_i, v_j). Thus, $r = i + 1$ and $(v_{i+1}, v_{j+1}) \in E'_v$.

Then, by setting $N := j - 1$ and using the facts that $(v_1, v_{N+1}) \in E'_v$ and $(v_i, v_j) \in E'_v \Rightarrow (v_{i+1}, v_{j+1}) \in E'_v$, we can show that $\{(v_i, v_{i+N}) | 1 \leq i \leq l - N\} \subseteq E'_v$. Moreover, it is easy to observe that including any other link

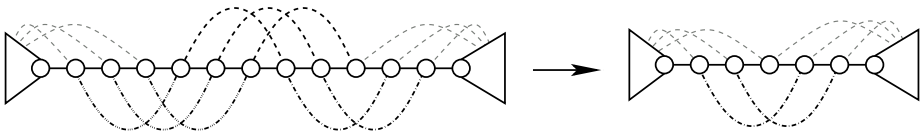

Figure 3: Example for the Degree-2-Path rule for $N = 3$, $l = 11$, $c = 2$, and $d = 2$.

between two vertices from $\{v_1, \ldots, v_l\}$ in E'_v would contradict the fact that the given instance is reduced with respect to the Shadow Deletion rule. This completes the proof. □

The fourth data reduction rule restricts the length of paths that consist solely of degree-2 vertices. By Lemma 3.5, the links with both endpoints from such a path admit a consecutiveness property. By making use of this property, the next data reduction rule replaces a long degree-2 path by a shorter "equivalent" degree-2 path.

> **Degree-2-Path:** Let $(T = (V, E), E', k)$ be a reduced instance with respect to the above three rules. Let $P = \{u, v_1\}, \{v_1, v_2\}, \ldots, \{v_l, w\}$ be a path in T such that $\deg(v_i) = 2$ for all $1 \leq i \leq l$, $\deg(u) \geq 3$, and $\deg(w) \geq 3$ and let E'_v denote the set of links with both endpoints from $v_1, v_2, \ldots, v_l$. If there exists an integer N with $l \geq 2N$ such that $E'_v = \{(v_i, v_{i+N}) \mid 1 \leq i \leq l-N\}$, then proceed as follows: Let $c := \lfloor \frac{l}{N} \rfloor - 1$ and $d := (l \mod N)$. Replace P by a path $P' = \{u, x_1\}, \{x_1, x_2\}, \ldots, \{x_{N+d}, w\}$. Remove all links in E'_v from E' and add the links (x_i, x_{i+N}) for $1 \leq i \leq d$ to E'. Replace every link (v_i, y) with $y \in V(T_{u,\{u,v_1\}})$ by the link (x_i, y) and replace every link (v_i, y) with $y \in V(T_{w,\{v_l,w\}})$ by the link $(x_{N+d-(l-i)}, y)$. Finally, decrease parameter k by c.

See Fig. 3 for an example of the application of the Degree-2-Path rule. Next, we prove that this rule is correct.

Lemma 3.6. *The Degree-2-Path rule is correct and can be executed in $O(n^2 + nm)$ time.*

Proof. Let $(T = (V, E), E', k)$ be a BCA-instance reduced with respect to the first three rules and let (T_a, E'_a, k_a) be the resulting instance after one application of the Degree-2-Path rule. Let $P = \{u, v_1\}, \{v_1, v_2\}, \ldots, \{v_l, w\}$ be the path for which the conditions of the Degree-2-Path rule are fulfilled and let E'_v be the set of links with both endpoints from $v_1, \ldots, v_l$. By Lemma 3.5, either $E'_v = \emptyset$ or there is an integer N with $E'_v = \{(v_i, v_{N+i}) \mid 1 \leq i \leq l - N\}$ and there is no link between a vertex in $V(T_{u,\{u,v_1\}})$ and a vertex in $V(T_{w,\{v_l,w\}})$. Since the Degree-2-Path rule is applicable to P, we know $E'_v \neq \emptyset$ and $l \geq 2N$. Then, we have $c = \lfloor \frac{l}{N} \rfloor - 1$, $d = (l \mod N)$, and $k_a = k - c$. We show that (T, E', k) is a yes-instance if and only if (T_a, E'_a, k_a) is a yes-instance. The correctness of the Degree-2-Path rule then follows by induction on the number of applications of the rule.

"⇒": Let $E'' \subseteq E'$ with $|E''| \leq k$ be a solution for the original instance. We first show some properties of $E'' \cap E'_v$ and then construct a solution E''_a for the new instance with $|E''_a| \leq k_a$ from E''.

We can assume that $E'' \cap E'_v$ contains only pairwise "non-overlapping" links, that is, there are no two links $(v_i, v_{i+N}), (v_j, v_{j+N}) \in E'' \cap E'_v$ with $i < j < i+N$: If there are two links $(v_i, v_{i+N}), (v_j, v_{j+N}) \in E'' \cap E'_v$ with $i < j < i + N$, then we construct another solution from E'' by replacing (v_j, v_{j+N}) by (v_{i+N}, z), where $z = v_{i+2N}$ or z is a vertex from $T_{w,\{v_l,w\}}$. Link (v_{i+N}, z) exists due to Lemma 3.3. Obviously, this yields a solution of the same size (or even of smaller size if (v_{i+N}, z) is already part of the solution).

Next, we show that $c \leq |E'' \cap E'_v| \leq c + 1$. On the one hand, since every link in E'_v covers exactly N edges and the links in $E'' \cap E'_v$ are pairwise non-overlapping, there can be at most $\lfloor \frac{l-1}{N} \rfloor = \lfloor \frac{(c+1)N+d-1}{N} \rfloor = c + 1$ pairwise non-overlapping links in $E'' \cap E'_v$. On the other hand, all edges of path P which lie between vertex v_N and vertex v_{l-N+1} have to be covered by links in E'_v. This means that $E'' \cap E'_v$ has cardinality at least $\lceil \frac{l-2N+1}{N} \rceil = \lceil \frac{(c+1)N+d-2N+1}{N} \rceil = (c-1) + \lceil \frac{d+1}{N} \rceil = c$.

In the following, let $i_l := \max\{i \mid (y, v_i) \in E'' \wedge y \in V(T_{u,\{u,v_1\}})\}$ and $i_r := \min\{i \mid (v_i, z) \in E'' \wedge z \in V(T_{w,\{v_l,w\}})\}$. We distinguish two cases, namely, $|E'' \cap E'_v| = c$ and $|E'' \cap E'_v| = c + 1$, and construct in both cases a solution for the new instance.

In the first case, we can assume that $E'' \cap E'_v = \{(v_{i_l}, v_{i_l+N}), (v_{i_l+N}, v_{i_l+2N}), \ldots, (v_{i_l+(c-1)N}, v_{i_l+cN})\}$. We construct the new solution E''_a from E'' as follows: We replace every link $(y, v_i) \in E''$ by a link (y, x_i). Note that, since the given instance is reduced with respect to the first three rules, the existence of the link (y, v_i) implies that $i < N \leq N + d$ and, thus, the link (y, x_i) exists in E'_a. The links $(v_i, z) \in E''$ with $z \in V(T_{w,\{v_l,w\}})$ are also replaced by links $(x_{N+d-(l-i)}, z)$. With the same reason as above, links (x_i, z) exist in E'_a. Finally, we remove all links in $E'' \cap E'_v$ from E''. The resulting set E''_a is then a solution for the new instance. Since $|E'' \cap E'_v| = c$ in this case, we have $|E''_a| \leq k_a$. Obviously, all edges of T_a that are not between the new vertices $x_1, \ldots, x_{N+d}$ are covered. To show that the edges between the new vertices are also covered, observe that the edges on the path between x_1 and x_{i_l} and the edges on the path between $x_{N+d-(l-i_r)}$ and x_{N+d} are covered by the links (y, x_{i_l}) with $y \in V(T_{u,\{u,v_1\}})$ and $(x_{N+d-(l-i_r)}, z)$ with $z \in V(T_{w,\{v_l,w\}})$ that replace the links (y, v_{i_l}) and (v_{i_r}, z), respectively. Then, it suffices to show that $N + d - (l - i_r) \leq i_l$. Since E'' is a solution of the non-reduced instance, we get $i_l + cN \geq i_r$. This is equivalent to

$$i_l \geq i_r - cN = N + d - cN - N - d + i_r = N + d - (l - i_r).$$

In the second case, we assume that $E'' \cap E'_v = \{(v_{i_l}, v_{i_l+N}), (v_{i_l+N}, v_{i_l+2N}), \ldots, (v_{i_l+cN}, v_{i_l+(c+1)N})\}$. We construct the solution E''_a for the new instance from E'' as follows: We replace every link $(y, v_i) \in E''$ with $y \in V(T_{u,\{u,v_1\}})$ by a link (y, x_i). The links $(v_i, z) \in E''$ with $z \in V(T_{w,\{v_l,w\}})$ are also replaced by links $(x_{N+d-(l-i)}, z)$. We remove all links in $E'' \cap E'_v$ from E''. Finally, we add

link (x_{i_l}, x_{i_l+N}) to E''. With a similar argument as in the first case, we can show that the resulting set is a solution of the new instance.

"$\Leftarrow$": Let E''_a with $|E''_a| \leq k_a$ be a solution of the reduced instance. Let E'_x denote the set of links from E'_a that have both their endpoints from $\{x_1, x_2, \ldots, x_{N+d}\}$. With a similar argument as in the proof of the reverse direction, we can assume that all links in $E''_a \cap E'_x$ are pairwise non-overlapping. By the construction of the links in E'_a, we can conclude that $0 \leq |E''_a \cap E'_x| \leq 1$. Next, we distinguish again two cases based on whether $|E''_a \cap E'_x| = 0$ or $|E''_a \cap E'_x| = 1$.

In the first case, we let $i_l := \max\{i \mid (y, x_i) \in E''_a \wedge y \in V(T_{u,\{u,x_1\}})\}$. We replace the links (z, x_i) with $z \notin \{x_1, x_2, \ldots, x_{N+d}\}$ in E''_a by their corresponding links in E' and, then, add c links (v_{i_l}, v_{i_l+N}), (v_{i_l+N}, v_{i_l+2N}), …, $(v_{i_l+(c-1)N}, v_{i_l+cN})$ to E''_a. In analog to the proof of the reverse direction, we can show that the resulting set is a solution E'' of the non-reduced instance with $|E''| \leq k$.

In the second case, let $E''_a \cap E'_x = \{(x_i, x_{i+N})\}$. We replace the links (z, x_i) with $z \notin \{x_1, x_2, \ldots, x_{N+d}\}$ in E''_a by their corresponding links in E'. Then, we replace (x_i, x_{i+N}) by (v_i, v_{i+N}) and add c links (v_{i+N}, v_{i+2N}), …, $(v_{i+(c-1)N}, v_{i+cN})$, $(v_{i+cN}, v_{i+(c+1)N})$ to E''_a. Analogously, we can show that the resulting set is a solution of the non-reduced instance.

Altogether, we have shown that the two instances are equivalent and, thus, the Degree-2-Path rule is correct. Next, we prove that this rule can be executed in $O(n^2 + n \cdot m)$ time. Using a depth-first traversal of the input tree T, we can determine in $O(n)$ time all paths of the form $P = \{u, v_1\}, \ldots, \{v_l, w\}$ with $\deg(u) \geq 3$, $\deg(w) \geq 3$, and $\deg(v_i) = 2$ for all $1 \leq i \leq l$. Next, we check for every such path whether the Degree-2-Path rule can be applied, that is, we have to determine an integer N with $l \geq 2N$ such that it holds for the set E'_v of links with both endpoints from $\{v_1, \ldots, v_l\}$ that $E'_v = \{(v_i, v_{i+N}) \mid 1 \leq i \leq l\}$. This is clearly doable in $O(n+m)$ time. If this rule is applicable for a path, then the replacing operation described in this rule can be carried out in $O(n+m)$ time: Replacing path P by the path $P' = \{u, x_1\}, \ldots, \{x_{N+d}, w\}$ can be executed in $O(n)$ time. Next, to find the links with exactly one endpoints in $\{v_1, \ldots, v_l\}$ and to replace them with links with exactly one endpoints in $\{x_1, \ldots, x_{N+d}\}$ can be done in $O(n + m)$ time. Therefore, the overall running time of the Degree-2-Path rule is $O(n^2 + n \cdot m)$. □

Next, we show that a BCA-instance reduced with respect to the four data reduction rules has $O(k^2)$ vertices and $O(k^2)$ links. As stated in the introduction of this section, we can easily bound the number of the leaves and the number of the internal vertices with degree at least three of a reduced instance by $O(k)$. Thus, the key point in the following is to upper-bound the number of the internal vertices with degree two. Herein, we consider the paths formed by degree-two vertices. The next two lemmas are used to show the upper bounds on the number and the length of such paths. The first one is due to Even et al. [7] and shows that there is no such degree-two vertex path between a leaf and an internal vertex of degree at least three.

Lemma 3.7 ([7]). *Let (T, E', k) be a BCA-instance to which the Shadow Deletion rule, the Unit Link rule, and the Covered Edge rule cannot be applied. Let v be a leaf of T and u be the parent of v. Then, $deg(u) \geq 3$.*

In the next lemma, we upper-bound the length of a path in a reduced instance that consists of degree-two vertices. Herein, we use $L(T)$ to denote the set of leaves in tree T.

Lemma 3.8. *Let (T, E', k) be a reduced BCA-instance and let $P = \{u, v_1\}, \{v_1, v_2\}, \ldots, \{v_l, w\}$ be a path in T with $deg(u) \geq 3$, $deg(w) \geq 3$, and $deg(v_i) = 2$ for all $1 \leq i \leq l$. Let $L_u := L(T_{u,\{u,v_1\}})$ and $L_w := L(T_{w,\{v_l,w\}})$. Then,*

(1) there are at most $2|L_u| + 2|L_w|$ links that have exactly one endpoint in $V_P := \{v_1, v_2, \ldots, v_l\}$.

(2) $l \leq 4 \cdot \min(|L_u|, |L_w|)$.

Proof. In the following, we use T_u and T_w to denote $T_{u,\{u,v_1\}}$ and $T_{w,\{v_l,w\}}$, respectively, and consider them as two rooted trees with roots u and w, respectively. For a vertex x in T_u or T_w, we use T_x to denote the subtree of T_u or T_w rooted at x. Moreover, we use A_u and A_w to denote the sets of internal vertices of degree at least 3 in T_u and T_w. Recall that $V_P := \{v_1, v_2, \ldots, v_l\}$.

The key for proving the lemma is the following observation: There exist at most $|A_u| + |L_u|$ links in E' with one endpoint in T_u and one endpoint in V_P and there exist at most $|A_w| + |L_w|$ links in E' with one endpoint in T_w and one endpoint in V_P. Here, we prove this observation for T_u. Consider a degree-2 vertex x in T_u. By Lemma 3.3, there exists a link $(x, y) \in E'$ with $y \in V(T_x)$. The existence of the link (x, y) then excludes any link (a, b) with $a \in V(T_y)$ and $b \in V_P$, since, otherwise, (x, y) would be a shadow and the Shadow Deletion rule would be applied. In this way, every degree-2 vertex x in T_u "blocks" at least one vertex from T_x from building links with the vertices in V_P. Thus, by a simple calculation, we arrive at the $|A_u| + |L_u|$-bound on the number of links between the vertices in T_u and the vertices in V_P.

From the above observation, we know that there are at most $|A_u| + |L_u| + |A_w| + |L_w|$ links with exactly one endpoint in V_P. Since $|A_u| \leq |L_u| - 1$ and $|A_w| \leq |L_w| - 1$, the first part of the lemma follows.

To prove the second part of the lemma, we distinguish two cases. First, suppose that there exists at least one link $(x, y) \in E'$ with $x \in V(T_u)$ and $y \in V(T_w)$. Then, since the instance is reduced with respect to the Shadow Deletion rule, there is no link in E' between two vertices in V_P. By Lemma 3.3, for every vertex v_i in V_P, there are at least two links $(v_i, a), (v_i, b) \in E'$ with $a \in V(T_u)$ and $b \in V(T_w)$. According to the above observation, there are at most $2|L_u| - 1$ (or $2|L_w| - 1$) links between V_P-vertices and the vertices in $V(T_u)$ (or $V(T_w)$). Therefore, $|V_P| \leq \min(2|L_u| - 1, 2|L_w| - 1)$.

In the second case, there is no link $(x, y) \in E'$ with $x \in V(T_u)$ and $y \in V(T_w)$. Let v_{i_l} be the vertex in V_P such that there exist a link $(v_{i_l}, z) \in E'$ with $z \in V(T_u)$ and, for all $i_l \leq i \leq l$, there is no link $(v_i, z) \in E'$ with $z \in V(T_u)$. From the above observation, we know $i_l \leq 2|L_u| - 1$. By Lemma 3.3

and the fact that the instance is reduced with respect to the Degree-2-Path rule, for every vertex v_i with $i_l \leq i \leq l$, there is a link $(v_i, v_j) \in E'$ with $1 \leq j \leq i_l$. Since the instance is reduced with respect to the Shadow Deletion rule, there cannot be two vertices from $\{v_{i_l+1}, \ldots, v_l\}$ which form two links with one vertex from $\{v_1, \ldots, v_{i_l}\}$, we can conclude $l \leq 2i_l \leq 4|L_u| - 2$. Obviously, the same argument can also be applied to obtain $l \leq 4|L_w| - 2$. The second part of the lemma follows. □

Now, we prove the size bound of the problem kernel for BCA.

Theorem 3.1. Bridge-Connectivity Augmentation *admits a problem kernel with $O(k^2)$ vertices and $O(k^2)$ links.*

Proof. Let (T, E', k) be a reduced BCA-instance. Let L be the set of leaves of T, A be the set of internal vertices of degree at least three of T, and B be the set of internal vertices of degree two. Clearly, $V = L \cup A \cup B$. By Lemma 3.1, we know that $|L| \leq 2k$ and $|A| \leq 2k - 1$. It remains to upper-bound $|B|$. By Lemma 3.7, we know that all degree-two vertices form paths between vertices in A. Thus, there can be at most $2k-2$ such paths. Moreover, according to the second part of Lemma 3.8, each of these paths contains at most $4 \cdot \frac{|L|}{2} = 2L \leq 4k$ vertices. Thus, we have $|B| \leq (2k-2) \cdot 4k$. Altogether, $|V| \leq 2k + 2k - 1 + (2k-2) \cdot 4k = O(k^2)$.

In order to show $|E'| = O(k^2)$, observe that there can be at most $O(k^2)$ links with both endpoints in L. Furthermore, for every vertex $u \in A$, there can be at most $|L|$ links with one endpoint being u: Root T at u. If there is a link $(u, v) \in E'$, then there exists no link (u, w) with w being in the subtree of T rooted at v; otherwise, there would be a shadow. Then, there can be at most $O(k^2)$ links with at least one endpoint from A. Finally, consider a path $P = \{u, v_1\}, \ldots, \{v_l, w\}$ with $\deg(u) \geq 3$, $\deg(w) \geq 3$, and $\deg(v_i) = 2$ for all $1 \leq i \leq l$. By the first part of Lemma 3.8, we know that there exists at most $2|L|$ links with exactly one endpoint from $\{v_1, \ldots, v_l\}$. Let E_v be the set of links with both endpoints from $\{v_1, \ldots, v_l\}$. By Lemma 3.5, these links in E_v are of a "consecutive" form, that is, there exists an integer N such that $E_v = \{(v_i, v_{i+N}) \mid 1 \leq i \leq l\}$. Moreover, from the second part of Lemma 3.8, we know that $l \leq 2|L|$. Hence, we can conclude $|E_v| \leq 2|L|$. Since, due to Lemma 3.7, there are at most $2k-2$ such degree-two-vertex paths, there can be at most $(2k-2) \cdot (2|L| + 2|L|) = O(k^2)$ links in E' with at least one endpoint from B. Altogether, we arrive at the claimed upper bound on the number of links in E', namely, $|E'| = O(k^2)$. □

4 The Biconnectivity Augmentation Problem

Recall the definition of the Biconnectivity Augmentation (BIA) problem: Given a graph $G = (V, E)$, a set of links E' on V, and a non-negative integer k, BIA asks for a set E'' of at most k links from E' whose insertion biconnects G. In this section, by studying Biconnectivity Augmentation (BIA), we deal with a more general problem setting than in the previous section. Hence, based on the

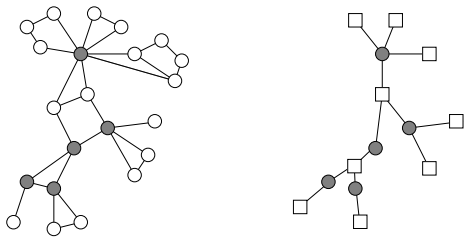

Figure 4: A graph together with its block tree. The cut-vertices are colored gray and the block-vertices are drawn as rectangles.

previous section, we extend and refine our kernelization technique introduced there. As shown by Frederickson and JáJá [10], we can assume that the input graph is a so-called *block tree*. A block tree $T = (V_T, E_T)$ is a tree over the vertex set $V_T := B \cup C$ with $B \cap C = \emptyset$ where the leaves of T form a subset of B and the edges in E_T have one endpoint from B and one endpoint from C.

A maximal connected induced subgraph that has no cut-vertex is called a *block*.[3] We can easily compute a block tree from a given undirected and connected graph $G = (V, E)$: Identify B as the set of blocks of G and C as the set of cut-vertices of G. Insert an edge between a block and a cut-vertex into E_T if the cut-vertex belongs to the block. In the following, the vertices in B are called *block-vertices* and the vertices in C are called *cut-vertices*. See Fig. 4 for an example of a graph and its block tree.

Bridge-Connectivity Augmentation (BCA) is a special case of BIA: For a given BCA-instance (T, E', k), we construct a BIA-instance by replacing every vertex in T by a block-vertex and every edge e in T by a degree-two cut-vertex that is adjacent to the endpoints of e. The link set and the parameter remain the same. In the following, we generalize the kernelization for BCA in Sect. 3 to a kernelization for BIA which leads to a quadratic-size problem kernel.

Eswaran and Tarjan [5] gave a lower bound on the size of the solutions of a BIA-instance.

Lemma 4.1 ([5])**.** *If (T, E', k) is a yes-instance of BIA, then $k \geq \lceil |L|/2 \rceil$ where L is the set of leaves of T.*

By Lemma 4.1, the number of leaves and the number of internal vertices of degree at least three of a given block tree can be easily bounded from above by $2k$ and $2k - 1$. Again, we focus on the internal vertices of degree two of T. The decisive difference between a BIA-instance and a BCA-instance lies in the partition of the tree vertices into two subsets, the block-vertices and the cut-vertices. A block-vertex can only have cut-vertices as neighbors and vice versa. In the following, we present first a preprocessing, which ensures that the links in E' are all between block-vertices.

[3]Note that a block of a connected graph is either a maximal biconnected subgraph or a bridge.

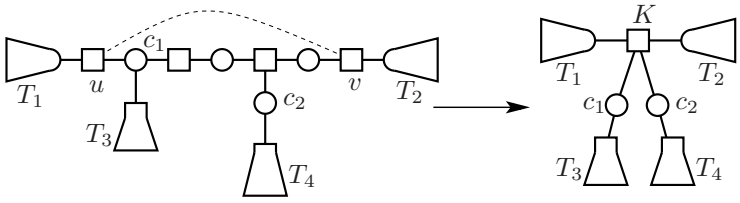

Figure 5: An illustration of the modification made after adding a link (u, v) to the solution set.

> **Preprocessing**: While there exists a link $(u, v) \in E'$ with $u \in C$ and $v \in B \cup C$, replace (u, v) by the link (w, v) where $w \in B$ is the neighbor of u that lies on the path between u and v in T. Finally, for all $u \in B \cup C$, remove all links (u, u) from E'.

To see the correctness of the preprocessing, the following equivalent formulation of BIA is helpful: Given a block tree $T = (B \cup C, E_T)$, a set of links E', and a non-negative integer k, find a subset E'' of links with $|E''| \leq k$ such that, for every $c \in C$, if c is removed from the graph $(B \cup C, E_T \cup E'')$, then the resulting graph is still connected.

Lemma 4.2. *The preprocessing is correct and can be executed in $O(n \cdot m)$ time.*

Proof. Based on the equivalent formulation of BIA, observe that, in order to ensure that a cut-vertex c is not a cut-vertex anymore after adding the links in E'', we have to add at least one link (u, v) that "covers" c, that is, c lies on the path between u and v in T, $c \neq u$, and $c \neq v$. Based on this observation, it is easy to see that a link (u, v) with one endpoint u being a cut-vertex does not really cover u. This link covers exactly the same set of cut-vertices as the link (v, w) with w being the neighbor of u that lies on the path between u and v. Thus, the preprocessing is correct.

Concerning the running time, there are m links in E'. For each link, we examine its endpoints and, in the case that at least one of its endpoints is a cut-vertex, we compute the unique determined path between its endpoints in T. This is clearly doable in $O(n)$ time. □

Next, we present the data reduction rules for BIA which generalize the data reduction rules in Sect. 3. Herein, if we add a link (u, v) to the solution E'', then, following Rosenthal and Goldner [18], we modify the instance as follows: Let P denote the path in T between u, v, let C be the set of cut-vertices on P that have degree at least three, and let N be the set of cut-vertices which are neighbors of the block-vertices in P and do not lie on P. Replace P by a single block-vertex K. Every link (u, v) with at least one endpoint being in P, say u, is replaced by link (K, v). For every vertex $v \in N$, add edge $\{K, v\}$ and, for every $c \in C$, add edge $\{K, c\}$. An illustration is given in Fig. 5.

The data reduction rules use the following terms and notations: Recall that a link (u, v) is called a *shadow*, if there exists a link (x, y) with $\{x, y\} \neq \{u, v\}$

such that, in T, the path between u and v is entirely contained in the path between x and y. For a vertex u, we use E'_u to denote the links in E' which cover u or have u as one of its endpoints. We call a path between two cut-vertices a *degree-2-cut-path* if all vertices on this path are degree-two vertices. A degree-2-cut-path is *maximal* if it is not a proper subpath of another degree-2-cut-path.

Shadow Deletion: Delete all shadows.

Unit Link: If there exists a cut-vertex u with $E'_u = \{(x, y)\}$, then add (x, y) to E'' and decrease the parameter k by one.

Covered Cut-Vertex: If there are two cut-vertices u and v with $E'_u \subseteq E'_v$ and $N(v) = \{w_1, w_2\}$, then add a new block-vertex w and make it adjacent to the vertices in $(N(w_1) \cup N(w_2)) \setminus \{v\}$ and replace every link of the form (w_1, x) or (w_2, x) by a link (w, x). Finally, remove v, w_1, w_2 from T.

Degree-2-Cut-Path: Let (T, E', k) be a BIA-instance to which the first three rules do not apply, let $P = \{c_1, b_1\}, \{b_1, c_2\}, \{c_2, b_2\}, \ldots, \{c_l, b_l\}, \{b_l, c_{l+1}\}$ be a maximal degree-2-cut-path in T with $\{b_1, \ldots, b_l\} \subseteq B$ and $\{c_1, \ldots, c_{l+1}\} \subseteq C$, and E'_b be the set of links with both endpoints from $\{b_1, \ldots, b_l\}$. If there exists an integer N with $2N \leq l$ such that $E'_b = \{(b_i, b_{i+N}) \mid 1 \leq i \leq l - N\}$, then proceed as follows: Let $c := \lfloor \frac{l}{N} \rfloor - 1$ and $d := (l \mod N)$. Replace P by a path $P' = \{c'_1, b'_1\}, \ldots, \{c'_{N+d}, b'_{N+d}\}, \{b'_{N+d}, c'_{N+d+1}\}$. Remove all links in E'_b from E' and add links (b'_i, b'_{i+N}) for $1 \leq i \leq d$ to E'. Replace every link (b_i, y) with $y \in V(T_{c_1,\{c_1,b_1\}})$ by link (b'_i, y) and replace every link (b_i, y) with $y \in V(T_{c_{l+1},\{b_l,c_{l+1}\}})$ by link $(b'_{N+d-(l-i)}, y)$. Finally, decrease the parameter k by c.

The correctness of the first three data reduction rules is easy to verify. The correctness of the Degree-2-Cut-Path rule can be shown in a similar way as the the Degree-2-Path rule in Sect. 3. Due to the similarity of the four rules to the ones in Sect. 3, the running time follows from Lemmas 3.2 and 3.6. All details omitted here can be found in the corresponding diploma thesis [20].

Lemma 4.3. *The four data reduction rules are correct and can be executed in $O(n \cdot m^3 + n^3 \cdot m)$ time.*

The proof of the next theorem works in analog to the proof of Theorem 3.1.

Theorem 4.1. Biconnectivity Augmentation *admits a problem kernel with $O(k^2)$ vertices and $O(k^2)$ links.*

5 Partial Augmentation Problems

In this section, we study partial augmentation problems. Here, given a graph, a set of links, and two non-negative integers Φ, k, one asks for a set of at most k links whose insertion in the graph results in a graph that has a subgraph with

at least Φ vertices that satisfies a given connectivity property. The three connectivity properties considered here are bridge-connectivity, biconnectivity, and strong connectivity. We show that, in the case that the link set is complete, that is, it contains all possible edges or arcs, Partial Bridge-Connectivity Augmentation (PBCA) and Partial Biconnectivity Augmentation (PBIA) are polynomial-time solvable and Partial Strong Connectivity Augmentation (PSCA) is W[2]-hard with respect to k. Note that the non-partial versions of all three problems, where, after adding at most k links, the whole graph should satisfy the given connectivity property, are polynomial-time solvable if the link set is complete.

5.1 Partial Bridge-Connectivity Augmentation

The Partial Bridge-Connectivity Augmentation problem (PBCA) is defined as follows: Given an undirected and connected graph $G = (V, E)$, a set of links E', and two non-negative integers Φ, k, find a set E'' of at most k links such that the graph $(V, E \cup E'')$ contains a bridge-connected component with at least Φ vertices. Note that in the case that E' is incomplete, Bridge-Connectivity Augmentation is NP-complete, which implies that PBCA is also NP-complete in this case. We show here that PBCA becomes polynomial-time solvable if E' is complete. This extends a result by Eswaran and Tarjan [5] saying that BCA is polynomial-time solvable in the case of a complete link set. Our solving strategy consists of two steps: The first step reduces the augmentation problem to a special subtree problem and the second step applies a dynamic programming approach to solve the subtree problem. The special subtree problem, Maximum d-Leaves Subtree (MLST), is defined as follows: Given a tree $T = (V_T, E_T)$, two non-negative integers N, d, and a weight function $w : V_T \to \mathbb{N}$, find a subtree of T with at most d leaves such that the total weight of the vertices in this subtree is at least N.

Lemma 5.1. Partial Bridge-Connectivity Augmentation *with complete link sets can be reduced to* Maximum d-Leaves Subtree *in* $O(|V|+|E|)$ *time.*

Proof. The reduction from the augmentation problem to MLST works as follows: Given a PBCA-instance (G, E', k, Φ), we contract the bridge-connected components of G into vertices, resulting in a tree T as described in Sect. 3. See Fig. 1 for an illustration. The weight $w(v)$ of a tree vertex v is set equal to the number of vertices in the corresponding bridge-connected component. Finally, we set $N := \Phi$ and $d := 2k$. The resulting MLST-instance consists of (T, N, d, w). Since the contraction of the bridge-connected components is doable in $O(|V| + |E|)$ time [19], the same time bound holds for this reduction.

Suppose that we have a solution set E'' of the PBCA-instance with $|E''| \leq k$. It is easy to observe that every link in E'' is between two bridge-connected components of G which correspond to leaves of T. Let L' denote the set of these leaves and let T' be the subtree of T with the leaf set L'. Then, the resulting bridge-connected component C of graph $(V, E \cup E'')$ with $|C| \geq \Phi$ contains the vertices which are contained in the bridge-connected components

of G which correspond to the vertices in T'. Therefore, the total weight of the vertices of T' is at least Φ and since $|E''| \leq k$, we have $|L'| \leq 2k = d$.

The other direction is easier. Given a subtree T' of T with at most d leaves and total vertex weight at least N, we can add $d/2$ links to make T' bridge-connected: Eswaran and Tarjan [5] showed that at least $\lceil \frac{|L|}{2} \rceil$ links are needed, where L is the set of leaves of T'. Moreover, they proved that this lower bound is achievable in linear time in the case of a complete link set. Therefore, we can find a set of $d/2 = k$ links to get a bridge-connected subgraph with $N = \Phi$ vertices. □

Next, we describe the dynamic programming approach solving MLST.

Theorem 5.1. Maximum d-Leaves Subtree *can be solved in* $O(|V_T| \cdot d^2)$ *time.*

Proof. We root the input tree T at an arbitrary vertex r and use T_v to denote the subtree rooted at a vertex v. The bottom-up dynamic program maintains a table A_v for each vertex v, which has d entries, $A_v(1), \ldots, A_v(d)$. Entry $A_v(i)$ stores the maximal total vertex weight achievable by a subtree of T_v that is rooted at v and has at most i leaves.The goal of the dynamic programming procedure is to compute $A_v(d)$ for all vertices $v \in V_T$. Then, $\max_{v \in V_T} A_v(d)$, by definition of $A_v(d)$, equals the total vertex weight achievable by a subtree of T.

The initialization of the table A_v for a leaf v is trivial, that is, we set $A_v(0) = 0$ and $A_v(i) = w(v)$ for $1 \leq i \leq d$. At an internal vertex v, let $c(v) = \{u_1, \ldots, u_j\}$ denote the set of v's children. We define table B_v, where for $1 \leq i \leq d$ and $0 \leq l \leq j$ table entry $B_v(i, l)$ is defined to be the maximal total vertex weight achievable by a subtree of T_v that is rooted at v, has at most i leaves, and contains no vertex from $\{u_1, \ldots, u_l\}$. Note that by this definition table entry $B_v(i, 0)$ equals $A_v(i)$. Indeed, B_v is just an "auxiliary" table needed for the computation of A_v. From $l := j$ to 0, we compute $B_v(i, l)$. For $l = j$, we set $B_v(i, j) = w(v)$ for all i. This is correct, since the subtree cannot contain a vertex from $c(v)$ and, thus, no vertex below v. For $l < j$, we set

$$B_v(i, l) := \max_{0 \leq i' \leq i} (B_v((i - i'), l + 1) + A_{u_{l+1}}(i')).$$

This is correct because every subtree rooted at v and containing no vertex from $\{u_1, \ldots, u_l\}$ can be decomposed into two trees, one is rooted at v and contains vertices from subtrees $T_{u_{l+2}}, \ldots, T_{u_j}$ and the other is rooted at u_{l+1} and is a subtree of $T_{u_{l+1}}$. Finally, we set $A_v(i) := B_v(i, 0)$ for all i.

At the root r, we know that the maximal total vertex weight achievable by a subtree of T with at most d leaves is then $\max_{v \in V_T} A_v(d)$. Using a simple traceback, this subtree can also be computed in the same time bound.

Concerning the running time, the dynamic program goes over all vertices. At each vertex v, it fills the table A_v. To do this, it computes the functions $B_v(i, l)$ with $1 \leq i \leq d$ and $0 \leq l \leq j$ where j denotes the number of v's children. For each of these functions, it computes again at most d values (see the recursion

above). Altogether, it needs $O(j \cdot d^2)$ time for a vertex having j children. Summing up over all vertices in T, we arrive at a total running time of $O(|V_T| \cdot d^2)$. □

Combining Lemma 5.1 and Theorem 5.1, we arrive at the main result of this subsection.

Theorem 5.2. *In the case of a complete link set,* Partial Bridge-Connectivity Augmentation *is solvable in* $O(|V| \cdot k^2)$ *time.*

5.2 Partial Biconnectivity Augmentation

In this section, we introduce the Partial Biconnectivity Augmentation problem (PBIA). Given an undirected and connected graph $G = (V, E)$, a set of links E', and two non-negative integers Φ, k, PBIA asks whether there exists a set $E'' \subseteq E'$ of size at most k such that $(V, E \cup E'')$ contains a biconnected component with at least Φ vertices. We show that, as Partial Bridge-Connectivity Augmentation, this problem is polynomial-time solvable if the link set is complete.

The idea of the solving strategy is to find, by dynamic programming on the block tree representation of G, a subgraph of G with at least Φ vertices that can be made biconnected by inserting at most k links.

Recall that the block tree $T_G = (V_T, E_T)$ of an undirected and connected graph G is a tree over the vertex set $V_T = B \cup C$, where B are the blocks and C are the cut-vertices of G (see Sect. 4). Moreover, a cut-vertex $c \in C$ is adjacent to a block $b \in B$ if and only if the cut-vertex c is contained in the block b. Given a graph G and the block tree $T_G = (C \cup B, E_T)$ of G, the weight of a subtree $T' = (B' \cup C', E'_T)$ with $C' \subseteq C$, $B' \subseteq B$, and $E'_T \subseteq E_T$ of the block tree T_G is defined as the size of the union of the blocks in B', that is, $w(T') := |\bigcup_{b \in B'} b|$. See Fig. 6 for an example.

We show that PBIA reduces to a constrained subtree problem on the block tree of the input graph.

Lemma 5.2. *If E' is complete, then (G, E', k, Φ) is a yes-instance for* Partial Biconnectivity Augmentation *if and only if the block tree $T_G = (C \cup B, E_T)$ of G contains a subtree $T' = (C' \cup B', E'_T)$ such that*

- *the leaves of T' are contained in B,*
- *the number of leaves of T' is at most $2k$,*
- *the maximum degree of a cut-vertex in T' is at most $k + 1$, and*
- *the weight $w(T') = |\bigcup_{b \in B'} b|$ of T' is at least Φ.*

Proof. The proof is similar to the proof of Lemma 5.1. It is based on the result by Eswaran and Tarjan [5] that for a connected graph one needs to add at least $\max\{\lceil |L_{T_G}|/2 \rceil, \Delta_c - 1\}$ links to turn it biconnected, where L_{T_G} denotes the set of leaves of the block tree of G and Δ_c denotes the maximum degree of

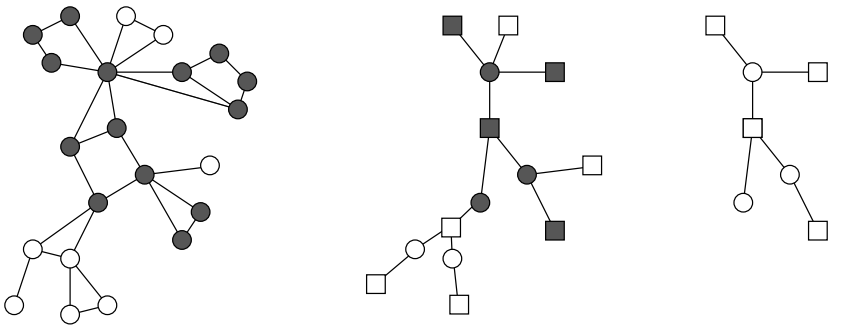

Figure 6: An example for a graph a) its block tree b) and a subtree of the block tree c). The vertices that are contained in the subtree, either as cut-vertices or contained in a block, are colored gray in a). The weight of the subtree in c) is 14 in accordance with the number of gray vertices in the graph shown in a).

a cut-vertex in the block tree of G. Moreover, they showed that, in the case of a complete link set, a set of $\max\{\lceil |L_{T_G}|/2 \rceil, \Delta_c - 1\}$ links whose addition turns the graph biconnected can be found in polynomial time. Next, we prove the correctness of the lemma.

"$\Leftarrow$": Consider a subtree $T' = (C' \cup B', E'_T)$ fulfilling the conditions of the lemma. Let $V' := \bigcup_{b \in B'} b$. Note that T' is the block tree of $G[V']$ and, hence, by the above mentioned result it can be turned biconnected by the addition of a set E'' of at most k links. Thus, V' forms a biconnected component in $(V, E \cup E'')$ of size at least Φ.

"$\Rightarrow$": Given a set E'' of at most k link such that $(V, E \cup E'')$ contains a biconnected component with at least Φ vertices. Moreover, let V' be the set of vertices in such a biconnected component and let $T' = (C' \cup B', E'_T)$ denote the block tree of $G[V']$. It is easy to verify that T' is a subtree of the block tree of G (indeed, $C' \subseteq C$, $B' \subseteq B$, and, hence, $T' = T_G[C' \cup B']$). It is well known that all leaves of a block tree are blocks. Hence, all leaves of T' are contained in B. Moreover, the weight $|\bigcup_{b \in B'} b|$ of T' is at least Φ since every vertex in V' occurs in at least one block in B'. Finally, note that T' has at most $2k$ leaves and a maximum cut-vertex degree of $k+1$. Otherwise, since the addition of E'' turns $G[V']$ biconnected, by the above mentioned result, E'' would contain at least $k+1$ links. □

The subtree problem introduced in Lemma 5.2 can be solved in $O(k^3 \cdot (|C| + |B|))$ time by a straightforward adaption of the dynamic programming for Maximum d-Leaves Subtree. The details can be found in the corresponding diploma thesis [20]. Since the block tree of a graph can be computed in linear time [19], we arrive at the following theorem.

Theorem 5.3. *In the case of a complete link set,* Partial Biconnectivity Augmentation *can be solved in* $O(k^3 \cdot |V| + |E|)$ *time.*

5.3 Partial Strong Connectivity Augmentation

Here, we show that Partial Strong Connectivity Augmentation (PSCA) is W[2]-hard, which is defined as follows: Given a directed graph $D = (V, A)$, a set of links $A' \subseteq V \times V$, and two non-negative integers Φ, k, find a subset A'' of A' with $|A''| \leq k$ such that $(V, A \cup A'')$ contains a strongly connected component with at least Φ vertices. We give a parameterized reduction from the W[2]-hard Set Cover problem [4], which is defined as follows: Given a set $S = \{s_1, s_2, \ldots, s_n\}$, a collection of subsets of S, that is, $C := \{S_1, S_2, \ldots, S_m\}$ with $S_i \subseteq S$ for all i, and a non-negative integer k, decide whether there is a subset C' of C with $|C'| \leq k$ such that $\bigcup_{S' \in C'} S' = S$.

We show first that, in the case that the link set A' is incomplete, that is, A' does not contain all possible arcs, PSCA is W[2]-hard. Then, we extend the proof to the case that A' is complete.

Theorem 5.4. Partial Strong Connectivity Augmentation *in the case of incomplete link sets is W[2]-hard with respect to k.*

Proof. Given a Set Cover-instance (S, C, k), we construct a PSCA-instance $(D = (V, A),\ A',\ \Phi,\ k')$ as follows (see Fig. 7 for an example): We add for each element $s \in S$ an element-vertex v_s and for each subset S_i in C a subset-vertex v_{S_i} to V. Moreover, a special vertex r is added to V. Then, we add n arcs to A, which connect all element-vertices to r. In addition, an arc from a subset-vertex v_{S_i} to an element-vertex v_s is added to A if $s \in S_i$. Then, we set $A' := \{(r, v_{S_i}) \mid S_i \in C\}$, $\Phi := k + n + 1$, and $k' := k$. Next, we prove that each Set Cover-solution $C' := \{S_{j_1}, S_{j_2}, \ldots, S_{j_k}\}$ with $1 \leq j_l \leq m$ for all $1 \leq l \leq k$ one-to-one corresponds to a PSCA-solution with $A'' := \{(r, v_{S_{j_1}}), (r, v_{S_{j_2}}), \ldots, (r, v_{S_{j_k}})\}$.

Let $C' := \{S_{j_1}, S_{j_2}, \ldots, S_{j_k}\}$ be a solution of the Set Cover-instance. Clearly, adding the arc $(r, v_{S_{j_l}})$ for $1 \leq l \leq k$ results in a strongly connected component consisting of r, $v_{S_{j_l}}$, and all element-vertices that correspond to the elements contained in S_{j_l}. Since $\bigcup_{S' \in C'} S' = S$, we get a strongly connected component with vertex r, subset-vertices $v_{S_{j_1}}, v_{S_{j_2}}, \ldots, v_{S_{j_k}}$, and all element-vertices. Altogether, this component has $1 + k + n$ vertices.

Consider a PSCA-solution $A'' := \{(r, v_{S_{j_1}}), (r, v_{S_{j_2}}), \ldots, (r, v_{S_{j_k}})\}$. Vertex r is in the resulting strongly connected component. Moreover, since we can without loss of generality assume that no subset in C is an empty set, the subset-vertices $v_{S_{j_1}}, v_{S_{j_2}}, \ldots, v_{S_{j_k}}$ are in this component. Note that no other subset-vertices can be in this connected component. Therefore, the remaining n vertices of this component are all element-vertices. Since an element-vertex v_s in D is only reachable from the subset-vertices that correspond to the subsets containing element s, the subcollection $C := \{S_{j_1}, S_{j_2}, \ldots, S_{j_k}\}$ covers all elements from S. □

Next, we show PSCA remains W[2]-hard even if A is complete.

Theorem 5.5. Partial Strong Connectivity Augmentation *in the case of complete link sets is W[2]-hard with respect to k.*

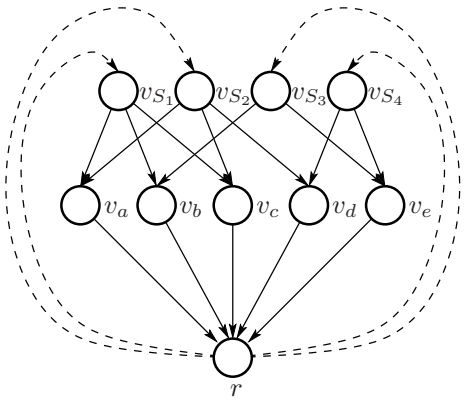

Figure 7: Example of the reduction from Set Cover to PSCA. Herein, $S_1 = \{a, b, c\}$, $S_2 = \{a, c, d\}$, $S_3 = \{b, e\}$, and $S_4 = \{d, e\}$. The arcs are drawn as solid lines and the links as dashed lines. In the case of an incomplete link set (Theorem 5.4), only the shown links are allowed to be part of the solution. In the case of a complete links set (Theorem 5.5), all possible links could be part of the solution. However, as shown in the proof of Theorem 5.5, we may assume that an optimal solution for the PSCA instance consists only of a subset of the shown links, too.

Proof. The construction of a PSCA-instance $(D = (V, A), A', \Phi, k')$ from a Set Cover-instance (S, C, k) is almost the same as in the proof of Theorem 5.4 (see Fig. 7 for an example). The only difference is that here $A' = V \times V$. Next we show the equivalence between the solutions. The direction that a Set Cover-solution corresponds to a PSCA-solution can be shown in the same way as in the proof of Theorem 5.4.

Given a PSCA-solution A'' with $|A''| \leq k$, we assume without loss of generality that A'' is minimal, that is, there is no proper subset of A'' whose addition to A results in a strongly connected component with at least Φ vertices. Let A_r be the set of links in A' that go from r to subset-vertices. We claim that $A'' \subseteq A_r$. To show this claim, we introduce the following notations. Let $r(u)$ for a vertex u be the set of vertices reachable from u in D including u. We say a link (u, v) is a *shadow* if there exists another link $(u', v') \in A'$ with $u' \in r(u)$ and $v \in r(v')$. We use A_p to denote the set of links (u, v) in A' with $v \in r(u)$ and A_s to denote the set of shadows. We can use the following argument to show that $A'' \cap (A_p \cup A_s) = \emptyset$: Suppose that there is some link in $A_p \cup A_s$ contained in A''. Let V' denote the vertex set of the resulting strongly connected component after adding A'' to A with $|V'| \geq \Phi$. Since A'' is minimal, there are two vertices $u, v \in V'$ such that the two paths between them contain a link (u, v) from $A_p \cup A_s$. In the case that $(u, v) \in A_p$, we can replace (u, v) by a path from u to v in D and remove (u, v) from A''. If $(u, v) \in A_s$, then we can find a link $(u', v') \in A'$ with $u' \in r(u)$ and $v \in r(v')$. Then, we can replace (u, v) by (u', v'). We can still find a path from u to v by passing through (u', v').

Thus, $A'' \cap (A_p \cup A_s) = \emptyset$.

In order to show $A'' \subseteq A_r$, it remains to show $A' \setminus (A_s \cup A_p) = A_r$. Clearly, the links (v_s, r) and (v_{S_i}, r) for element-vertices v_s and subset-vertices v_{S_i} are all shadows. Clearly, a link (r, s) for an element-vertex v_s is a shadow of a link (r, v_{S_i}) for a subset-vertex v_{S_i} with $s \in S_i$. Moreover, for all $1 \leq i < j \leq m$, link (v_{S_i}, v_{S_j}) is a shadow of link (r, v_{S_j}), since $r \in r(v_{S_i})$. Analogously, for all $1 \leq i \leq n$ and $1 \leq j \leq m$, links (v_{s_i}, v_{S_j}) are shadows of links (r, v_{S_j}). It is also easy to see that links (v_{s_i}, v_{s_j}) for all $1 \leq i < j \leq n$ are shadows of links (r, v_{s_j}). In the case that $1 \leq i \leq n$ and $1 \leq j \leq m$, the links (v_{S_i}, v_{s_j}) are shadows of links (r, v_{s_j}). These are all possible links of $(V \times V) \setminus A_r$. Therefore, $A'' \subseteq A_r$.

With $A'' \subseteq A_r$, the proof of the corresponding Set Cover-solution follows analogously to the proof of Theorem 5.4. □

6 Open Problems

The most interesting open problem is to study the parameterized complexity of the Strong Connectivity Augmentation problem, where we are given a directed graph $D = (V, A)$, a set of links $A \subsetneq V \times V$, and a non-negative integer k and ask for a subset A'' of links such that graph $(V, A \cup A'')$ is strongly connected. Improving the size bounds of the problem kernels for Bridge-Connectivity Augmentation and Biconnectivity Augmentation to a linear function in k is another interesting open problem. Further opportunities for future work include investigating the approximability and fixed-parameter tractability of the Partial Bridge-Connectivity Augmentation and Partial Biconnectivity Augmentation problems for the case that the link set E' is incomplete.

Acknowledgement: We thank Rolf Niedermeier for inspiring discussion and helpful comments. We are grateful to an anonymous *Networks* referee for providing constructive feedback that helped to significantly improve the presentation of this work.

References

[1] J. Alber, N. Betzler, and R. Niedermeier. Experiments on data reduction for optimal domination in networks. *Annals of Operations Research*, 146(1):105–117, 2006.

[2] S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 2009. Accepted for publication.

[3] F. Dehne, M. A. Langston, X. Luo, S. Pitre, P. Shaw, and Y. Zhang. The cluster editing problem: Implementations and experiments. In *Proc. 2nd IWPEC*, volume 4196 of *LNCS*, pages 13–24. Springer, 2006.

[4] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[5] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.

[6] G. Even, J. Feldman, G. Kortsarz, and Z. Nutov. A 3/2-approximation algorithm for augmenting the edge-connectivity of a graph from 1 to 2 using a subset of a given edge set. In *Proc. 4th APPROX*, volume 2129 of *LNCS*, pages 90–101. Springer, 2001.

[7] G. Even, J. Feldman, G. Kortsarz, and Z. Nutov. A 1.8 approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Transactions on Algorithms*, 5(2), 2009.

[8] M. R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In *Proc. 2nd IWPEC*, volume 4169 of *LNCS*, pages 276–277. Springer, 2006.

[9] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[10] G. N. Frederickson and J. JáJá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

[11] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[12] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 51(1):7–25, 2008.

[13] S. Khuller and R. Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.

[14] G. Kortsarz, R. Krauthgamer, and J. R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM Journal on Computing*, 33(3):704–720, 2004.

[15] M. Marzban, Q.-P. Gu, and X. Jia. Computational study on dominating set problem of planar graphs. In *Proc. 2nd COCOA*, volume 5165 of *LNCS*, pages 89–102. Springer, 2008.

[16] H. Nagamochi. An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree. *Discrete Applied Mathematics*, 126:83–113, 2003.

[17] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[18] A. Rosenthal and A. Goldner. Smallest augmentations to biconnect a graph. *SIAM Journal on Computing*, 6(1):55–66, 1977.

[19] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[20] J. Uhlmann. *Parameterized Algorithmics for Connectivity Augmentation Problems in Networks.* Diploma thesis, WSI für Informatik, Universität Tübingen, Germany, 2007. URL: `http://theinf1.informatik.uni-jena.de/~uhlmann/diploma-thesis.pdf`.