Approximation Algorithms for Connected Graph Factor Problems

Marten Waanders
University of Twente

august 5, 2014

Abstract

Creating low cost networks that satisfy certain connectivity requirements is one of the main concerns within network design. Examples of this problem include VLSI design, vehicle routing and communication networks. This thesis describes two approximation algorithms for creating a low weight k-edge-connected d-regular subgraph under the assumption that edge weights satisfy the triangle inequality. These algorithms increase the connectivity of a d-regular graph until it is a k-edge-connected graph, without changing the degree of any vertex.

Acknowledgements

I would like to thank my supervisor Bodo Manthey for his advice and input. I have regarded our meetings as enjoyable and beneficial for the quality of my work.

Contents

Chapter 1

Introduction

1.1 Motivation

Creating a low cost network that satisfies some connectivity requirement is one of the main concerns within network design. Examples of this problem include VLSI design, vehicle routing and communication networks. These network design problems can easily be translated to graphs. For instance, in transportation networks one can make a complete graph where the various locations of interest are vertices and where weights on edges indicate the cost of connecting and maintaining a connection between two locations (e.g. the cost of maintaining certain roads). A common requirement is that the graph must be connected. However for some networks higher connectivity requirements need to be met, such that when a few connections break down the network can still function. An example of such systems is the telephone system where emergency numbers should be reachable at all times. For telecommunications networks, an important requirement is the resilience to link failures [15].

The practice of creating networks that satisfy stricter connectivity requirements is called survivable network design. Survivable network design is of importance when designing systems for which a lack of connectivity between parts of the network might be catastrophic. The goal of the survivable network problem is to find a graph that provides multiple routes between pairs of vertices. The connectivity between vertices is commonly defined using either edge-disjoint paths or vertex-disjoint paths. It is also possible but less common to consider a combination of these (note though that the number of vertex-disjoint paths is always at least as high as the number of edge-disjoint-paths). Research indicates that for telephone networks 2-connected topologies provide a high amount of survivability in a cost

effective manner [7].

More formally the survivable network design problem for edge connectivity can be stated as follows:

Deterministic survivable network design problem (NDP)

Input: An undirected graph $G = (V, E)$, edge costs w_e, and a $|V| \times |V|$ matrix r defining the edge connectivity requirements.
Output: A minimum cost set of edges $E' \subseteq E$ such that for all i, j with $i \neq j$, there exist at least r_{ij} edge disjoint path between vertices i and j.

For vertex connectivity the problem is analogously:

Deterministic survivable network design problem (vertex connectivity case) (NDP)

Input: An undirected graph $G = (V, E)$, edge costs w_e, and a $|V| \times |V|$ matrix r defining the vertex connectivity requirements.
Output: A minimum cost set of edges $E' \subseteq E$ such that for all i, j with $i \neq j$, there exist at least r_{ij} vertex disjoint path between vertices i and j.

In some cases it is also useful to add limits on the degree of vertices (the amount of direct connections to a single node in the network) or even fixing them. Upper bounds can be useful when vertices can only handle a certain amount of connections. For example, in applications of network design problems to multicasting, the degree constraint on a switch corresponds to the maximum number of multicast copies it can make in the network. Fixing the degree specification completely can also be useful though. For instance adding the specific degree specification that all vertices have a degree of 2 leads to the well known traveling salesman problem (TSP).

Traveling salesman problem (TSP)

input: An undirected graph $G = (V, E)$, edge costs w_e.
Output: A minimum weight Hamiltonian cycle of G. Note that a Hamiltonian cycle is a connected spanning subgraph where every vertex has a degree of 2.

In the Traveling Salesman Problem the original example application is that of a traveling salesman wanting to find the shortest possible route that visits each city he needs to travel too exactly once, while also returning to the city he started he came from. In this case the set of vertices is a list of cities and the edge costs are the distances between each pair of cities. The

Traveling Salesman Problem has many other applications, examples of which are designing the route a guard should follow and designing the most efficient ring topology that connects hundreds of computers.

Allowing one specific vertex to have a degree $2m$ while fixing the rest at degree two gives the vehicle routing problem with m vehicles, which is a generalization of TSP.

Vehicle routing problem with m vehicles (m-VRP)

input: An undirected graph $G = (V, E)$, edge costs w_e, a starting vertex v_d and a number m.

Output: A minimum weight set of m cycles of G. These cycles are disjoint with the exception that they all include vertex v_d, and every vertex in G is contained in at least one of these m cycles.

The most commonly mentioned application of the vehicle routing problem is designing a set of m least-cost vehicle routes in such a way that every city is visited exactly once by exactly one vehicle and all vehicle routes start and end at a specific vertex (v_d). This specific vertex is often called the depot.

1.2 Notation and Definitions

In this section we introduce some notation and definitions that we use throughout this document.

1.2.1 Graph Theory

A graph is a representation of a set of objects where some pairs of objects are connected by links. The objects are represented by mathematical abstractions called vertices, and the links are called edges. A graph is commonly depicted as a set of dots for the vertices, joined by lines or curves for the edges. Figure 1.2.1 depicts some example graphs. A simple graph is an undirected graph that does not contain loops (edges connected at both ends to the same vertex) and for any pair of vertices u and v it does not contain multiple edges from u to v. A simple graph is commonly defined as an ordered pair $G = (V, E)$, where V is the set of vertices and E is the set of edges. An edge is often described as a pair of vertices. Thus if an edge e connects the vertices u and v it can be written as $e := (u, v)$ (depending on the writer edges may instead be defined as $\{u, v\}$ or simply uv). Note that in undirected graphs for any edge $(u, v) \in E$ we will always have that $(u, v) = (v, u)$. A graph is finite if both the set of vertices and the set of edges are finite. In this case $|V|$ denotes the number of vertices and $|E|$ denotes the number of edges. A

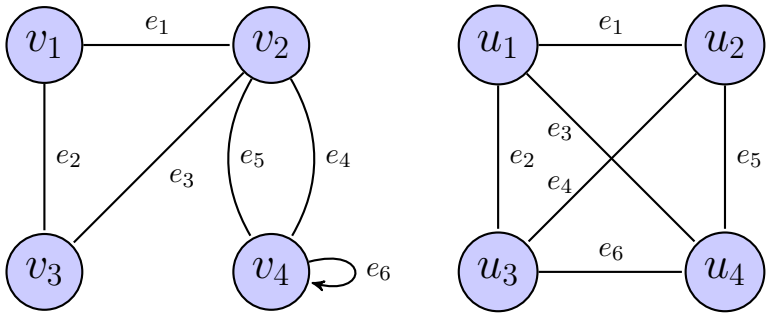

Figure 1.1: Example graphs: graph G_1 is on the left, while graph G_2 is on the right)

multigraph is a graph that does not contain loops, where it is possible for pairs of vertices u and v to have multiple edges from u to v. Note though that some authors also allow multigraphs to have loops, while others use the term pseudograph to describe multigraphs that are allowed to contain loops. Throughout this document we assume that all graphs are simple and finite unless mentioned otherwise.

In graph G_1 of figure 1.2.1 edge e_6 is a loop as it connects v_4 to itself. Also edges e_4 and e_5 both connect v_2 to v_4. Graph G_2 on the contrary does not contain loops and does not contain multiple edges connecting the same vertices. Therefore G_2 is a simple graph, while G_1 is not. Both graphs are finite graphs as both the set of vertices and the set of edges are finite. Any of the edges can be written as (u, v) with u and v specific vertices. For instance in graph G_1 the edge e_1 can be written as (v_1, v_2) and the edge e_6 can be written as (v_6, v_6). As long as a graph is simple, no confusion can arise over which edge is meant. Though as can be seen in the pseudograph G_1 (which as already mentioned is not a simple graph), both edges e_4 and e_5 can be written as (v_2, v_4).

Two edges of a graph are called adjacent if they share a common vertex. Similarly, two vertices are called adjacent if they share a common edge. When u is adjacent to v, the vertex u is called a neighbour of v. An edge e is incident to a vertex v when that edge connects vertex u to some other vertex v. The degree of a vertex is defined as the number of edges that are incident to that vertex. We denote the degree of a vertex v in G by $d(v; G)$. Note that we will drop the argument G when no confusion can arise as to which graph is referred. If you add the degrees of every vertex of a graph and divide by 2, you will attain the number of edges. Thus $|E| = \sum_{v \in V} d(v)/2$. For this reason the sum of the degrees must always be even. A graph is a regular graph if all vertices on that graph have the same degree. A graph where all

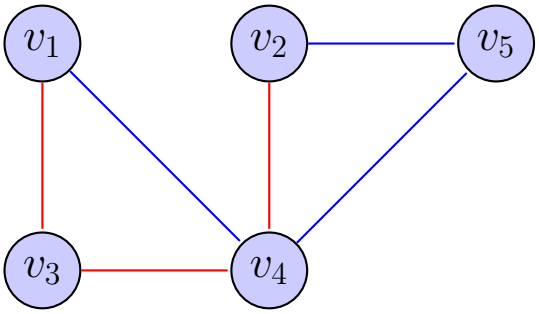

Figure 1.2: two edge disjoint paths that are not vertex disjoint)

vertices have a degree of k is called a *k-regular* graph. For $v \in V$ let $N(v; G)$ be the set of vertices adjacent to v in G.

In graph G_1 edge e_1 is adjacent to edge e_2, as they both have the vertex v_1 in common. The edge e_1 is incident to both v_1 and v_2. Thus vertex v_1 is adjacent to v_2. The degree of vertex v_1 is 2, as v_1 is incident to the two edges e_1 and e_2. Thus $d(v_1) = 2$. Similarly $d(v_2) = 4, d(v_3) = 2$ and $d(v_4) = 4$. Graph G_2 is a 3-regular graph, as all vertices of G_2 have a degree of 3.

A path in a graph is a finite or infinite sequence of edges which connect a sequence of vertices. A simple path is a path which does not repeat vertices. For example, two possible (simple) paths from v_2 to v_3 in G_1 are $\{e_3\}$ $\{e_1, e_2\}$. Two paths are (internally) vertex-disjoint (alternatively, vertex-independent) if they do not have any internal vertex in common. Similarly, two paths are edge-disjoint (or edge-independent) if they do not have any edge in common.

Lemma 1.1. *Two internally vertex-disjoint paths are edge-disjoint, but the converse is not necessarily true.*

Proof. Let P_1 and P_2 be internally vertex-disjoint paths in a graph G. Assume to the contrary that they are not edge disjoint. Then we must have that they both contain the same edge (u, v). However this means that both paths must contain vertices u and v. For P_1 and P_2 to be internally vertex-disjoint we must have that u and v are both endpoints of P_1 and P_2. However as G is a simple graph (remember that we assume all graphs are simple unless stated otherwise), it then follows that $P_1 = P_2$ and thus that P_1 and P_2 are not edge disjoint.

For the converse note that figure 1.2.1 contains a counterexample where the blue and the red edges are two edge-disjoint paths from v_1 to v_2, but they are not internally vertex-disjoint as they both contain the same internal vertex v_4. □

A circuit is a list of vertices starting and ending with the same vertex,

where each two consecutive vertices in the sequence are adjacent to each other in the graph. A circuit can also be defined as a path that starts and ends at the same vertex. A cycle is a circuit where no repetitions of vertices is allowed, other than the necessity of the starting vertex being the same as the ending vertex. A cycle can also be described by its set of edges, instead of a sequence of vertices. In graph G_1 an example of a cycle is $\{v_1, v_2, v_3, v_1\}$, or equivalently $\{e_1, e_2, e_3\}$.

Two vertices u and v are connected in a graph G, if G contains a path from u to v. If u and v are not connected in G they are called disconnected. We define a graph G to be connected if all vertex pairs within G are connected. An edge cut C is a group of edges whose removal disconnects the graph. Thus a set of edges $C \subseteq E$ is an edge cut if the graph $G' = (V, E \setminus C)$ is disconnected. We call $C \subseteq E$ a k-edge cut if C is an edge cut and $|C| = k$.

For $X, Y \subseteq V$ denote by $\mathrm{Cut}(X, Y; G)$ the set of edges of G with one endpoint in $X \setminus Y$ and one endpoint in $Y \setminus X$. Let $d(X, Y; G) := |\mathrm{Cut}(X, Y; G)|$. Also, for $X \subseteq V$ define $\mathrm{Cut}(X; G) := \mathrm{Cut}(X, V \setminus X; G)$ and $d(X; G) := |\mathrm{Cut}(X; G)|$. Note that the definition of the function $d(X, Y; G)$ here does not conflict with the degree function $d(v; G)$ defined on vertices, and can be seen as a generalization of it. This is because $d(\{v\}; G) = d(\{v\}, V \setminus \{v\}; G) = |\mathrm{Cut}(\{v\}, V \setminus \{v\}; G)|$ is the number of edges from v to the all other vertices in G, which is just the degree of v. Thus we have $d(\{v\}; G) = d(v; G)$ and we can use these notations interchangeably.

The edge connectivity of a graph G, for which we will use the notation $\lambda(G) \in \mathbb{R}$, is the size of the smallest edge cut in G. We denote by $\lambda(u, v; G)$ the local edge-connectivity of two vertices u, v in a graph G, which is the size of a smallest edge cut disconnecting u from v. Menger's theorem allows us to give an alternative interpretation to local edge-connectivity in terms of edge-independent paths rather than edge cuts: Let x and y be two distinct vertices. The size of the minimum edge cut disconnecting x and y is equal to the maximum number of pairwise edge-independent paths from x to y. Note that local k-edge-connectivity is a relation between vertices. It is in fact an equivalence relation as we prove in Lemma 1.2. Let $k \in \mathbb{Z}$, a graph G is called *k-edge-connected* if the edge connectivity of the graph is at least k, thus if $\lambda(G) \geq k$.

Lemma 1.2. *Local k-edge-connectivity an equivalence relation.*

Proof. To prove a relationship is an equivalence relation we need to prove it is reflexive, symmetric and transitive. Note that local edge-connectivity is reflexive as no edge cut can disconnect a vertex v from itself. As our graph is undirected, we automatically have that local edge-connectivity is symmetric.

For transitivity, we need that when $\lambda(u, v) \geq k$ and $\lambda(v, w) \geq k$, we will also have $\lambda(u, w) \geq k$. Take an arbitrary set of $k - 1$ edges $C \subsetneq E$. As $\lambda(u, v) \geq k$, u and v are still connected in $G - C$ (by Menger's theorem there are at least k edge disjoint paths from u to v (at least one of these paths still exists after removing the edges in C from the graph G)). The same holds for v and w. Clearly u and w are then also still connected in $G - C$. As C was arbitrary we see there exists no $k - 1$ edge cut that disconnects u from w. Thus $\lambda(u, w) \geq k$. □

A subgraph $G' = (V', E')$ of a graph $G = (V, G)$ is a graph such that $V' \subseteq V$, $E' \subseteq E$ and E' contains no edges incident to vertices in $V \setminus V'$. A subgraph G' is a spanning subgraph of G if G has the same vertex set as G. A subgraph G' of a graph G is said to be an induced subgraph if, for any pair of vertices $u, v \in G'$, the edge (x, y) is contained in G' if and only if (x, y) is contained in G. Thus an induced subgraph of a graph G has exactly the edges as G over its vertex set, where its vertex set is a subset of G. When an induced subgraph $G' = (V', E')$ of G is k-edge-connected we call G' a *k-edge-connected component.*

Let $X \subseteq V$ be a nonempty set of vertices. We call $X(G) \subseteq V$ a *locally k-edge-connected component* of G if for every two vertices $u, v \in X(G)$ we have that $\lambda(u, v; G) \geq k$. Clearly, if X is a k-edge-connected component it is also a locally k-edge-connected component. Though the reverse does not hold in general.

The definitions for vertex connectivity are similar to those of edge connectivity, except that they work with vertex cuts rather than edge cuts. A vertex cut C is a group of vertices whose removal disconnects the graph. Letting $E(C)$ be the sets of edges incident to the vertices C, we have that a set of vertices $C \subseteq V$ is an vertex cut if the graph $G' = (V \setminus C, E \setminus E(C))$ is disconnected. Or equivalently a set of vertices $C \subseteq V$ is an vertex cut if the graph induced by $V \setminus C$ is disconnected. We call $C \subseteq E$ a *k-vertex cut* if C is an vertex cut and $|C| = k$.

The vertex connectivity of a graph G, for which we will use the notation $\kappa(G) \in \mathbb{R}$, is the size of the smallest vertex cut in G. We denote by $\kappa(u, v; G)$ the local vertex-connectivity of two vertices u, v in a graph G, which is the size of a smallest vertex cut disconnecting u from v. Let $k \in \mathbb{Z}$, a graph G is called *k-vertex-connected* if the vertex connectivity of the graph is at least k, thus if $\kappa(G) \geq k$. A complete graph is a graph such that every vertex $u \in G$ is adjacent to every other vertex $v \in G$. Graph G_2 is an example of a compete graph. Note that a complete graph has no vertex cuts at all, but by convention its vertex connectivity is defined to be $|V| - 1$. When an induced subgraph $G' = (V', E')$ of G is k-vertex-connected, we

call G' a *k-vertex-connected component*. Let $X \subseteq V$ be a nonempty set of vertices. We call $X(G) \subseteq V$ a *locally k-vertex-connected component* of G if for every two vertices $u, v \in X(G)$ we have that $\kappa(u, v; G) \geq k$. Clearly if X is a k-vertex-connected component it is also an locally k-vertex-connected component, although the reverse does not hold in general. There is also a Menger's theorem for local vertex connectivity, which states that the size of the minimum vertex cut disconnecting x and y (the minimum number of vertices whose removal disconnects x and y) is equal to the maximum number of pairwise vertex-independent paths from x to y.

A tree is a graph that is connected and contains no cycles. There are alternative definition of a tree that can be proven to be equivalent. For instance a tree is a graph that contains no cycles where a cycle is formed if any edge is added to the graph, or a tree is a connected graph that is not connected if any single edge is removed from the tree. For finite graphs trees can also be defined as connected graph with $|V| - 1$ edges, or as a graph that does not contain cycles and has $|V| - 1$ edges.

A Hamiltonian cycle is a cycle that visits every vertex exactly once. A graph is Hamiltonian if it contains a Hamiltonian cycle. Graph G_2 is Hamiltonian, because it contains (amongst others) the Hamiltonian cycle e_1, e_2, e_6, e_5. An Eulerian circuit is a circuit that contains every edge of the graph exactly once. An Eulerian graph is a graph that contains a Eulerian circuit

A matching is a set of edges with no common vertices. A perfect matching is a matching which matches all vertices of the graph. For example in graph G_2 a perfect matching is $\{e_2, e_5\}$.

Let $b = (b_v : v \in V)$ be a vector containing positive integers. A b-matching is a graph such that every vertex v has a degree of at most $b(v)$. A perfect b-matching is a graph that satisfies the requirements that every vertex $v \in V$ has a degree equal to $b(v)$. Note that if $b(v) = 1$ for all vertices v, then a b-matching is a matching, and thus a b-matching can be seen as a generalization of a matching. For some choices of the vector b there may not exists a perfect b-matching because the problem is infeasible. However is is possible to decide whether this is the case by checking a few easy to check conditions. Lemma 1.3 and its proof state exactly which conditions need to hold.

Lemma 1.3 (Proof from Takuro Fukunaga and Hiroshi Nagamochi [6])**.** *Let V be a vertex set with $|V| \geq 2$ and $b : V \to Z^+$ be a degree specification. Then there exists a perfect b-matching if and only if $\sum_{v \in V} b(v)$ is even and $b(v) \leq \sum_{u \in (V \setminus \{v\})} b(u)$ for each $v \in V$.*

Proof. The necessity is trivial. We show the sufficiency by constructing a perfect b-matching. We let V $= \{v_1, \ldots, v_n\}$ and $B = \sum_{l=1}^{n} b(v_l)/2$. For

$j = 1, \ldots, B$, we define i_j as the minimum integer such that $\sum_{l=1}^{i_j} b(v_l) \geq j$, and i'_j as the minimum integer such that $\sum_{l=1}^{i'_j} b(v_l) \geq B + j$. Notice that $\sum_{l=1}^{i_j-1} b(v_l) < j$ holds by definition if $i_j \geq 2$. Then we can see that $i_j \neq i'_j$ since otherwise we would have $b(v_{i_j}) = \sum_{l=1}^{i_j} b(v_l) - \sum_{l=1}^{i_j-1} b(v_l) > (B+j)-j = B$ if $i_j \geq 2$ and $b(v_{i_j}) \geq B + j > B$ otherwise, which contradicts to the assumption.

Let $M = \{e_j = v_{i_j} v_{i'_j} | j = 1, \ldots, B\}$. Then M contains no loop by $i_j \neq i'_j$. Moreover G_M is a perfect b-matching since $|\{j | i_j = l \text{ or } i'_j = l\}| = b(v_i)$, as required. □

In this thesis the problem of finding a minimum cost perfect b-matching is of importance.

Minimum cost perfect b-matching

Input: An undirected graph $G = (V, E)$, edge costs w_e, and a vector $b = (b_v : v \in V)$ defining the degree requirements.
Output: A a minimum cost set of edges $E' \subseteq E$ such that for all vertices v, $d(v) = b(v)$

We call a vertex set X a k-special component in G if X is a locally k-edge-connected component satisfying $d(X) \leq k - 1$. Note that each vertex with a degree lower than k is trivially a k-special-component.

1.2.2 Metrics

A metric or distance function is a function that defines a distance between elements of a set. If a metric is defined on a set X it must be a function w of the form $w : X \times X \to \mathbb{R}$. Such a function $w(x, y)$ is a metric on a set X, if is satisfies the following conditions for all $x, y, z \in X$:

- $w(x, y) \geq 0$ (non-negativity, or separation axiom),
- $w(x, y) = 0$ if and only if $x = y$ (identity of indiscernibles, or coincidence axiom),
- $w(x, y) = w(y, x)$ (symmetry),
- $w(x, z) \leq w(x, y) + d(y, z)$ (subadditivity / triangle inequality).

1.3 Approximation Algorithms

For mathematical problems an instance of a problem is a possible input of that problem. For example, for many graph problems an instance of the problem is a specific graph. The set of solutions to a problem is the set of all possible outputs the algorithm may generate. The abstract problem can be seen as the relation that associates an instance of the problem to the correct answer.

Many real-world optimization problem are challenging from a computational standpoint. Large instances of a problem may not be solvable due to the amount of computation that would be needed to find the optimal solution.

An approximation algorithm is an algorithm that runs in polynomial time and finds a solution of provable quality in that its solution is at most a constant factor m larger than that of the optimal solution. Approximation algorithms are commonly used for problems that are NP-hard because, unless $P = NP$, it is impossible to find a deterministic algorithm that always finds the optimal solution of the instance in polynomial running time. They are also being used for problems where polynomial running time algorithms that solve the problem optimally are known, but where the running time of these algorithms is too slow for the problems at hand. The approximation ratio of such an algorithm is the bound on how much worse the algorithm's solution is compared to the optimal solution in the worst case.

An approximation algorithm with an approximation ratio m finds a solution to any instance the problem, such that the cost of the solution is at most $m \cdot w(OPT)$, where OPT is the optimal solution of that instance and $w(OPT)$ be the cost of this solution.

Such an algorithm is also called an m-approximation algorithm. The smaller the approximation ratio, the better the algorithm is at guaranteeing a low cost solution to the problem. For instance when an algorithm has an approximation ratio of 2, the solution to an instance can be at worst twice as expensive compared to the optimal, but when the approximation ratio is 1.05 the solution can be at most 1.05 times as large.

1.4 Problem Statement

As already mentioned, there are many real world applications to finding a low weight subgraph satisfying connectivity requirements and possibly some degree requirements. The connectivity requirements are commonly stated in term of edge connectivity or vertex connectivity. Between each pair of

vertices $\{u, v\}$, a prescribed value $r(u, v)$ is then given for the needed edge-connectivity (or vertex connectivity). The higher the value $r(u, v)$, the more important it is that vertex u stays connected to vertex v. Rather than specifying a value $r(u, v)$ for each edge (u, v), it may be sufficient to specify a value $r(u)$ for each vertex u. In this case $r(u)$ is a measure of how important it is to have the vertex u stay connected to the rest of the graph, and u should stay connected to the rest of the graph if less than $r(u)$ edges are deleted. Some vertices can be considered more important that others and will thus have a higher value for $r(u)$. For instance in a electricity system it is more important that a power plant stays connected than that a single home stays connected to the grid. When every vertex is considered of equal importance regarding connectivity, these connectivity requirements simplify to $r(u) = k$ for some integer k. In this case the problem becomes finding an k-edge-connected or k-vertex-connected graph. Note that the case of having connectivity requirements defined on vertices is a special case of the one where we define connectivity requirements for each edge. To see this, note that we can write $r(u, v) = \max(r(u), r(v))$ for all vertices u and v.

When assigning degree constraints the most commonly used constraints either completely fix the degrees of the vertices or bound them from above. These constraints are most commonly used as they tend to arise in various real world problems. As already mentioned, upper bounds are useful when vertices can only handle a certain amount of connections. Fixing the degree specification completely on the other hand can be useful as it can ensure creating a network with a specific structure, such as a ring topology for computer networks.

In recent years, much effort has been put into designing approximation algorithms for network design problems with additional degree constraints. Most of these network design problems are generalizations of TSP. TSP is an NP-hard problem and the problem remains NP-hard even for the case where the vertices are in a plane with Euclidean distances [18]. Removing the condition of letting each vertex have a degree of exactly 2 and allowing higher degrees does not remove the NP-hardness of TSP, as it is easily seen that in the planar case there is an optimal tour that visits each vertex only once (should a vertex be visited more than once, we can take a shortcut that skips a repeated visit without increasing the tour length). Unless P=NP, NP-hard problems have an execution time that is not limited by a polynomial in the input size. For this reason, as soon as larger instances of an NP-hard problem arise it is often impractical to find optimal solutions to these instances. However there are techniques that tend to give substantially faster algorithms, usually at the expense of not necessarily finding the optimal solution or by placing restrictions on the input. These techniques include

the following:

- Restriction: When restricting the structure of the input (e.g., requiring distances to be metric), it may be possible to find faster algorithms.
- Parameterization: At times there are fast algorithms when certain parameters of the input are fixed. Many problems have the following form: given an object x and an integer k, does x have some property that depends on k? For instance, when requiring a d-regular k-edge-connected graph, the parameter can be the either the number d or the number k. For some applications the parameter k may be small compared to the total input size. For such applications it is useful to have an algorithm whose computation time is exponential only in k.
- Randomized algorithms: Use randomness to get a faster average running time. Generally randomization algorithms attempt to give a good performance in the average case over all possible choices of random input. These algorithms are commonly allowed to fail with some small probability when used on large instances of NP-hard problems to achieve faster running times.
- Heuristic algorithms: The objective of a heuristic is to find a solution in a reasonable time frame that works reasonably well in most cases. For heuristic algorithms there is no proof that the algorithm is both always fast and always produces a good result.
- Approximation algorithms: Instead of searching for an optimal solution, search for a solution that is close to optimal. Unlike heuristics, the solution is required to have a provable solution quality and provable run-time bounds.

In this thesis we will in particular look at the problem of approximating a minimum weight k-edge connected d-regular graph and the minimum weight k-vertex connected d-regular graph.

minimum weight k-edge connected d-regular graph

Input: An undirected complete graph $G = (V, E)$, edge weights w and numbers $d, k \in N$ with $d \geq k$.
Output: A minimum weight k-edge-connected d-regular graph R of G.

minimum weight k-vertex connected d-regular graph

Input: An undirected complete graph $G = (V, E)$, edge weights w and numbers $d, k \in N$ with $d \geq k$.
Output: A minimum weight k-vertex-connected d-regular graph R of G.

Note that TSP can not be approximated at all when edge weights do not satisfy the triangle inequality [1, 2, 17], and for general TSP it is therefore impossible to find an approximation algorithm unless $P = NP$.

Lemma 1.4 (proof from pages 30 and 31 of the book Approximation algorithms [17])**.** *For any polynomial time computable function $\alpha(n)$, TSP cannot be approximated within a factor of $\alpha(n)$, unless $\boldsymbol{P = NP}$.*

Proof. Assume, for a contradiction, that there is a factor $\alpha(n)$ polynomial time approximation algorithm, A, for the general TSP problem. We will show that A can be used for deciding the Hamiltonian cycle problem (which is **NP**-hard) in polynomial tine, thus implying $\mathbf{P} = \mathbf{NP}$.
The central idea is a reduction from the Hamiltonian cycle problem to the TSP, that transforms a graph G on n vertices to an edge-weighted complete graph G' on n vertices such that

- if G has an Hamiltonian cycle, then the cost of an optimal TSP tour in G' is n and
- if G does not have a Hamiltonian cycle, then an optimal TSP tour in G' is of cost $> \alpha(n) \cdot n$

Observe that when run on graph G', algorithm a must return a solution of cost $\leq \alpha(n) \cdot n$ in the first case, and a solution of cost $> \alpha(n) \cdot n$ in the second case. Thus, it can be used for deciding whether G contains a Hamiltonian cycle.
The reduction is simple. Assign a weight of 1 to edges of G, and a weight of $\alpha(n) \cdot n$ to nonedges, to obtain G'. Now, if G has a Hamiltonian cycle, then the corresponding tour in G' has a cost of n. On the other hand, if G, has no Hamiltonian cycle, any tour in G' must use an edge of cost $\alpha(n) \cdot n$, and therefore has cost $> \alpha(n) \cdot n$.

□

As most network design problems are generalizations of TSP it is natural to assume edge weights satisfy the triangle inequality when trying to find approximation algorithms. In this thesis we will also assume that we have non-negative weights. Non-negative weights tend to arise naturally in many applications.

1.5 Related Work

A lot of research has already been done on network design problems and a lot of approximation algorithms have already been created. Below we review some related work on this subject.

Note that when the connectivity requirements are removed, the problem becomes finding a minimum cost perfect b-matching. This problem can be solved in polynomial time [13, 16]. This is done by extending the graph using a construction that was first observed by Tutte and is described on pages 385 and 386 of the book Matching Theory [13]. This will construct a new graph G' which contains a perfect matching if and only if the original graph has a perfect b-matching. Afterwards a minimum weight matching is computed over G'. Note that a minimal weight matching can be found in polynomial time [9]. Note that when all vertices are given large degrees some weak connectivity constraints may become trivially satisfied. For instance, when every vertex has a degree of at least $|V|/2$, any perfect b-matching is automatically connected.

For the case where the degree requirements are removed we get the deterministic survivable network design problem (NDP) as mentioned in the motivation section. There does not exist a polynomial time algorithm for finding the optimal solution to this problem. However, there do exist approximation algorithms. Jain designed an algorithm for finding a 2-approximation for the edge-connectivity version of the problem [8]. The algorithm uses the ILP corresponding to the problem. The ILP is relaxed to an LP and is then solved with the ellipsoid algorithm. Then all solutions with value above 0.5 are rounded up to 1 and then these variables are fixed. Now the LP is again solved for the remaining variables and this process is continued until all variables are fixed (the paper proves that there is always at least one edge with a value over 0.5). This process of finding a solution by rounding up some variables and then solve the residual LP iteratively is called iterative rounding. For the vertex connectivity variant, Kortsarz and Nutov gave approximation algorithms for the special case of k-vertex connected graphs [10]. For arbitrary costs, they designed a k-approximation algorithm for undirected graphs and a $(k+1)$-approximation algorithm for directed graphs. For metric costs, they created a $(2+(k-1)/n)$-approximation algorithm for undirected graphs and a $(2+k/n)$-approximation algorithm for directed graphs. When the vertex connectivity requirements can be written as $r(u,v) = \max(r(u), r(v))$ where $r(u)$ is the connectivity requirement for vertex u, the best known approximation algorithm has an approximation ratio of $2(k-1)$, where $k = \max_{u \in V}(r(u))$ [14].

We now summarize some known results where both connectivity and de-

gree constraints are involved.

Recall that TSP asks for a connected graph where each vertex has degree 2. Such a graph is automatically 2-edge-connected as the conditions placed on the graph automatically ensure the graph is a cycle. For the case of TSP where edges satisfy the triangle inequality, Christofides' approximation algorithm has an approximation ratio of 3/2 [4]. The algorithm is described in Algorithm 1

input : undirected complete graph $G = (V, E)$, edge weights w
output: low weight Hamiltonian cycle H (2-edge-connected 2-regular subgraph) of G

1 Compute a minimum spanning tree T of G.
2 Let V_O be the set of vertices with odd degree in T. Compute a minimum weight perfect matching M in the complete graph over the vertices from V_O.
3 Combine the edges of M and T to form a multigraph H.
4 Form an Eulerian circuit C in H (H is Eulerian because it is connected, with only even-degree vertices).
5 Obtain a Himiltonian cycle H by skipping repeat visits to vertices of the circuit C (shortcutting).

Algorithm 1: Christofides algorithm for TSP

Cornelissen et al. [5] designed an approximation algorithm for the problem of finding a 2-edge-connected d-regular subgraph where edge weights satisfy the triangle inequality.

minimum weight 2-edge-connected d-regular spanning subgraph

Input: An undirected complete graph $G = (V, E)$, edge weights w and an integer k.

Output: A minimum weight spanning subgraph R of G that is 2-edge-connected and d-regular.

Their approximation ratio is 3 when d is odd and 2.5 for even d. The algorithm first computes a minimum cost d-regular graph, which is a special case of a minimum cost b-matching. Then the graph is transformed into a 2-edge-connected graph without changing the degree of any vertex.

Another way to find an approximation ratio for a problem is to find a bicriteria solution in which both the violation of some of the constraints as well as the final weight of the graph are constrained by bounds on how much worse they can be compared to the optimal solution. Examples in-

clude algorithns by Lau et al. [11, 12]. For instance, Lau et al. [11] designed an algorithm for approximating a minimum cost spanning subgraph which satisfies edge-connectivity requirements $r(u, v)$ between vertices and which also satisfies degree upper bounds $b^+(v)$ on the vertices. Their result is an $(2, 2b^+(v) + 3)$-approximation algorithm. This means that the cost of the graph is at most twice that of the optimal solution and the degree of each vertex v is at most $2b^+(v) + 3$.

Lau et al. [3] created an algorithm to construct a minimum cost k-edge-connected spanning subgraph under specific degree constraints.

minimum weight k-edge-connected b-matching

Input: An undirected complete graph $G = (V, E)$, edge weights w and an integer valued function b defined on V and an integer k.
Output: A minimum weight spanning k-edge-connected subgraph R of G such that $d(v; R) = b(v)$ for every vertex v.

They proved that any k-edge-connected graph G can be transformed into a graph with maximum degree $k + 1$ without increasing its cost. As we already noted there exists a 2-approximation algorithm for finding a minimum cost k-edge-connected spanning subgraph [8]. This thus translates into a 2-approximation algorithm for finding a minimum cost k-edge-connected spanning subgraph where each vertex has a maximum degree of $k+1$. To reduce the maximum degree down to k the cost of a minimum weight matching needs to be added to the approximation ratio. The cost of such a matching is proven to be at most $1/k$ times the cost of the minimum weight k-edge-connected subgraph. In total this gives a $(2+1/k)$-approximation algorithm for finding the minimum weight k-edge-connected k-regular subgraph. Their results can be generalized to the case of general connectivity requirements.

The deterministic survivable network design problem with degree constraints

Input: An undirected complete graph $G = (V, E)$, edge weights w and an integer valued function b defined on V and a $|V| \times |V|$ matrix r defining the edge connectivity requirements.
Output: A minimum cost set of edges $E' \subseteq E$ such that for all i, j with $i \neq j$, there exist at least r_{ij} edge disjoint path between vertices i and j. Furthermore for every vertex v, exactly $b(v)$ edges are incident to v.

Let $r_{\max} := \max_{u,v} r(u, v)$. Any graph G satisfying the general connectivity requirements can be transformed into a graph with maximum degree $\lceil r_{\max} \rceil$ while at most doubling its cost. The problem without degree constraints was 2-approximable [8]. Thus this results in a 4-approximation

algorithm when $r_{\max}$ is even. When $r_{\max}$ is odd this leads to a $(4, +1)$-approximation algorithm (where the $+1$ indicates the degree constraints can be off by one and the 4 is the approximation ratio on the cost function). Reducing the maximum degree to $r_{\max}$ in the case of odd $r_{\max}$ gives an 4.5-approximation ratio instead. For the case of vertex connectivity Lau et al. [3] also find a $(2 + \frac{k-1}{n} + \frac{1}{k})$-approximation algorithm for finding the minimum k-vertex connected k-regular graph.

One possible alteration to the problem is to allow multigraphs rather than simple graphs. Fukunaga and Nagamochi [6] give an approximation algorithm for finding a minimum cost k-edge-connected multigraph under the constraint that the degree of each vertex $v \in V$ is equal to a given value $b(v)$. As additional condition they require that $b(v) \geq 2$ for all vertices v. The problem admits an approximation algorithm with approximation ratio 2.5 if k is even and an approximation ratio $2.5 + 1.5/k$ if k is odd. The algorithm first creates a minimum cost perfect b-matching. Then if $|V| \leq 3$, this b-matching is the solution. Otherwise a Hamiltonian cycle G_h is computed using Christodes' algorithm and this cycle is copied $\lceil \frac{k}{2} \rceil$ times. The perfect b-matching and the $\lceil \frac{k}{2} \rceil$ copies of G_h are merged to a single graph. Then the algorithm uses operations that reduce the number of edges of the perfect b-matching, while ensuring that each vertex keeps a degree of at least $b(v)$, and without generating loops. The copies of the Hamiltonian cycle, which are not being modified yet, automatically ensure the graph is k-edge-connected. Afterwards the degrees of the vertices are further reduced to $b(v)$ by removing vertices from some of the copies of the Hamiltonian cycle, in such a way that no loops are generated. The degree of a vertex is reduced by 2 for each such cycle it is removed from. The algorithm ensures each cycle keeps at least 2 vertices to prevent loops. The algorithm also ensures k-edge-connectivity is not violated by carefully choosing which cycle to remove the vertices from. It is proven that this method successfully reduces the vertex degree of every vertex v to $d(v)$, thus achieving a graph with the desired properties.

Chapter 2

First Algorithm for Approximating d-Regular k-Edge-Connected Subgraphs

In this chapter we will generalize the algorithm of Cornelissen et al. [5]. The original algorithm is a 3-approximation algorithm for the problem of finding a d-regular 2-edge-connected graph. This algorithm start with a minimum weight d-regular graph. Given this graph G, it creates a tree $T(G)$ as follows: The tree has a vertex for every maximum 2-edge-connected subgraph (a 2-edge connected component) of G, and two such vertices are connected in $T(G)$ if the corresponding 2-edge connected components are connected in G. It is provable that every 2-edge connected component $L_i(G)$ of G contains an edge $e_i = (u_i, v_i)$ for which both u_i and v_i are not incident to any vertex outside $L_i(G)$.

Then the algorithm of Cornelissen et al. computes a minimum spanning tree and shortcuts it too a Hamiltonian cycle H. This cycle is shortcutted further to create a cycle H' that trough the vertices $\{u_1, \ldots, u_k\}$ where k is the number of 2-edge connected components. We now assume w.l.o.g. that H' traversed these vertices in the order $u_1, \ldots, u_k$. Now a d-regular 2-edge-connected graph is constructed by removing the edges (u_i, v_i) and adding the edges (u_i, v_{i+1}).

We will generalize this algorithm such that it will work on a i-edge-connected graph G and will create an $(i+1)$-edge-connected graph R for which every vertex has exactly the same degree as in G. The only constraint placed on the degrees of the original graph is that the minimum degree of the graph is at least $\lceil i+1 \rceil$. Note that this condition is trivial when $i+1$ is even, because we can not create a $(i+1)$-edge-connected graph if their exist vertices with a degree lower than $i+1$. This will allow us to find approxima-

tion for the minimum weight d-regular k-edge-connected spanning subgraph by repeatedly using this algorithm, although the approximation ratio does grow in the size of k.

2.1 Preliminaries

Let us first prove some lemmas on local k-edge-connectivity and on the structure of locally k-edge-connected components.

Lemma 2.1. *Let $X \subseteq V$ be a nonempty set of vertices satisfying $d(X) \leq k-1$. Assume every vertex in X has at least degree k. Then we must have $|X| \geq k$.*

Proof. Assume to the contrary that there exists a set of vertices $X \subseteq V$ with $d(X) \leq k-1$ having $|X| \leq k-1$. Each vertex $v \in X$ satisfies $d(v) \geq k$ and thus has at least k neighbours. Also note that each vertex v can have at most $|X|-1$ neighbours within X. Thus, it has at least $d(v)-(|X|-1) \geq k+1-|X|$ edges to vertices outside of X. Adding over all vertices in X this gives us $d(X) \geq (k+1-|X|)|X| = (k+1)|X| - |X|^2$ edges to vertices outside of X. Clearly for $|X| = 1$ this gives us $k+1-1 = k$ and for $|X| = k$ this also gives us $(k+1)k - k^2 = k$ edges to vertices outside of X. Noting that this function is a parabola we know that for $|X| \in (1, k)$ we have $d(X) > k$. We attained $d(X) \geq k$, which is a contradiction to $d(X) \leq k-1$. Thus our assumption of $|X| < k$ was wrong and we must have $|X| \geq k$. □

Corollary 2.2. *Assume every vertex in X has at least degree k. Then every k-special-component X satisfies $|X| \geq k$.*

Lemma 2.3. *Let G be a $(k-1)$-edge-connected graph. Every vertex set $X \subsetneq V$ either contains a k-special-component or satisfies $d(X) \geq k$.*

Proof. Assume to the contrary that we can find a set X that does not contain a k-special-component and satisfies $d(X) \leq k-1$.

Now let us take a minimal set X with this property. As X is minimal, there is no set of vertices $Y \subsetneq X$ that satisfies $d(Y) < k$ and does not contain a k-special-component. Clearly for any set of vertices $Y \subsetneq X$ we know Y does not contain a k-special-component as X does not contain such a component (If Y contains a k-special-component, X would also contain that k-special-component as $Y \subsetneq X$). Therefore, for each subset Y of X we must have $d(Y) \geq k$.

As we already have that $d(X) < k$, we know that if X were locally k-edge-connected, it would be a k-special-component. However, we also know that

X does not contain k-special-components and thus X cannot be a k-special-component. Thus, X is not locally k-edge-connected. As X is not locally k-edge-connected, we can find vertices $u, v \in X$ such that $\lambda(u, v) \leq k - 1$.

As $\lambda(u, v) \leq k - 1$ there exists an edge cut of size at most $k - 1$ that disconnects u from v. This cut must split up the graph in two sets U and $\overline{U}$ with $u \in U$, $v \in \overline{U}$, such that $d(U) \leq k - 1$.

Figure 2.1 illustrates how the graph is split up by these edge cuts. We can now see the graph G as being divided into the following four sections:

- $C_1 = X \cap U$,
- $C_2 = \overline{X} \cap U$,
- $C_3 = X \cap \overline{U}$,
- $C_4 = \overline{X} \cap \overline{U}$.

We can also count the number of edges between these segments as follows:

- k_1 is the number of edges between C_1 and C_2 ($k_1 = d(C_1, C_2)$),
- k_2 is the number of edges between C_3 and C_4 ($k_2 = d(C_3, C_4)$),
- k_3 is the number of edges between C_1 and C_3 ($k_3 = d(C_1, C_3)$),
- k_4 is the number of edges between C_2 and C_4 ($k_4 = d(C_2, C_4)$),
- k_5 is the number of edges between C_1 and C_4 ($k_5 = d(C_1, C_4)$),
- k_6 is the number of edges between C_2 and C_3 ($k_6 = d(C_2, C_3)$).

In the following we specify constraints that the variables $\{k_i, i \in \{1, \ldots, 6\}\}$ must satisfy. Then we show that it is impossible to satisfy all these constraints and thus come to a contradiction. We know that by construction $C_1 \neq \emptyset$ and $C_3 \neq \emptyset$ (as they contain u and v respectfully). We also know $C_2 \cup C_4 \neq \emptyset$ as X is a proper subset of V. However we can have that either C_2 or C_4 is empty. Let us first assume $C_2, C_4 \neq \emptyset$.

Case 1: $C_2, C_4 \neq \emptyset$**.**

First of all we have $d(X) < k$. This translates into $k_1 + k_2 + k_5 + k_6 \leq k - 1$. We also have $d(U) \leq k - 1$ and thus $k_3 + k_4 + k_5 + k_6 \leq k - 1$.

Note that we have $C_1, C_3 \subsetneq X$ as by construction we have that $v \in X \cap U$ and $u \in X \cap \overline{U}$. We can translate this to constraints $k_1 + k_3 + k_5 \geq k$ and $k_2 + k_3 + k_6 \geq k$ respectively.

For our final constraint we need to use the fact that our graph is $(k-1)$-edge-connected. In particular, any edge cut that disconnects C_2 and C_4

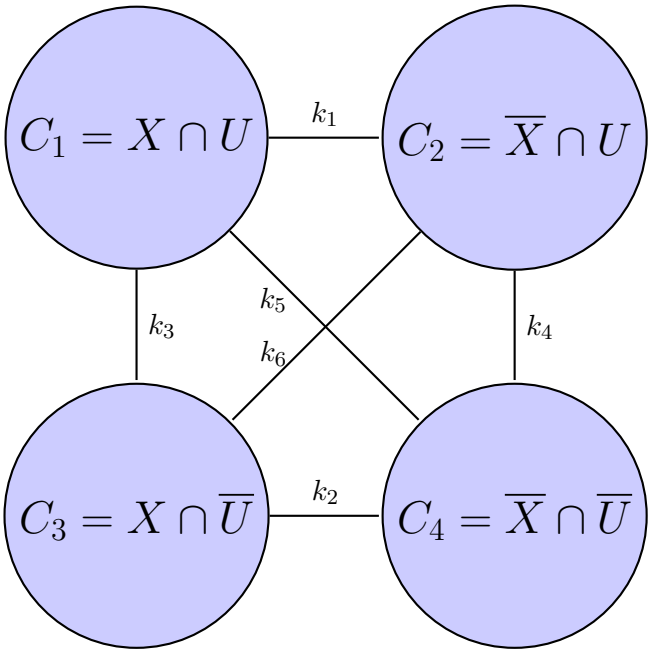

Figure 2.1: The graph divided by two edge cuts (one horizontal and the other vertical)

contains at least $k-1$ edges. This gives us the following bound on k_4: $k_4 \geq k - 1 - \min(k_1, k_2, k_3) - k_5 - k_6$. To see this, note that we want to find the maximum number of edge independent paths from C_2 to C_4 that the graph can possibly have. This total number of paths needs to be at least $k-1$. We know that we can have at most one such path for each of the k_4 direct edges and at most $\min(k_1, k_2, k_3)$ paths following a detour through C_2, C_1, C_3, C_4. There can then still be paths going trough C_2, C_3, C_4 and C_2, C_1, C_4, however we can upper bound those by the number of edges k_6 and k_5 respectfully.

Thus our system of equations becomes:

$$\begin{aligned}
&\text{Find } k_1, k_2, k_3, k_4, k_5, k_6 \in \mathbb{N} && (2.1.1)\\
&\text{s.t. } k_1 + k_2 + k_5 + k_6 \leq k - 1 && (2.1.2)\\
&\quad k_3 + k_4 + k_5 + k_6 \leq k - 1 && (2.1.3)\\
&\quad k_3 + k_1 + k_5 \geq k && (2.1.4)\\
&\quad k_3 + k_2 + k_6 \geq k && (2.1.5)\\
&\quad k_4 \geq k - 1 - \min(k_1, k_2, k_3) - k_5 - k_6 && (2.1.6)
\end{aligned}$$

We shall now show that this set of equations does not have a solution.

First note that the constraint 2.1.6 can be rewritten as the following three constraints:

- $k_4 \geq k - 1 - k_1 - k_5 - k_6$
- $k_4 \geq k - 1 - k_2 - k_5 - k_6$
- $k_4 \geq k - 1 - k_3 - k_5 - k_6$

We can rewrite constraint 2.1.3 as $k_4 \leq k - 1 - k_3 - k_5 - k_6$. From this constraint and the constraint $k_4 \geq k - 1 - k_3 - k_5 - k_6$ we see that we have $k_4 = k-1-k_3-k_5-k_6$. Filling this in into the equations $k_4 \geq k-1-k_1-k_5-k_6$ and $k_4 \geq k - 1 - k_2 - k_5 - k_6$, we get $-k_3 \geq -k_1$ and $-k_3 \geq -k_2$. Thus $k_3 \leq k_1$ and $k_3 \leq k_2$.

We also have $k_1 + k_3 + k_5 \geq k$ and $k_2 + k_3 + k_6 \geq k$ which we add giving us the constraint $k_1 + k_2 + 2k_3 + k_5 + k_6 \geq 2k$. We know $k_1 + k_2 + k_5 + k_6 \leq k - 1$ according to constraint 2.1.2, and we multiply this by two giving us $2k_1 + 2k_2 + 2k_5 + 2k_6 \leq 2k - 2$. We know that $k_3 \leq k_1$ and $k_3 \leq k_2$, thus we also have $k_1+k_2+2k_3+2k_5+2k_6 \leq 2k-2$. Finally we note that $0 \leq k_5+k_6$ as k_5 and k_6 can not be negative, and thus we get $k_1+k_2+2k_3+k_5+k_6 \leq 2k-2$. Clearly $k_1 + k_2 + 2k_3 + k_5 + k_6 \geq 2k$ and $k_1 + k_2 + 2k_3 + k_5 + k_6 \leq 2k - 2$ can not both be satisfied and thus we have found a contradiction.

Case 2: $\mathbf{C_2} = \emptyset$. Let us assume $C_2 = \emptyset$, we get the following equations (just a simplification of the equations for the general case):

Find $k_2, k_3, k_5 \in N$
s.t. $k_2 + k_5 \leq k - 1$
$k_3 + k_5 \leq k - 1$
$k_3 + k_5 \geq k$
$k_3 + k_2 \geq k$

Clearly we can not have both $k_3 + k_5 \geq k$ and $k_3 + k_5 \leq k - 1$ leading to an immediate contradiction.

Case 3: $\mathbf{C_4} = \emptyset$. We now get the following equations:

Find $k_1, k_3, k_6 \in N$
s.t. $k_1 + k_6 \leq k - 1$
$k_3 + k_6 \leq k - 1$
$k_3 + k_1 \geq k$
$k_3 + k_6 \geq k$

We find the equations of $k_3 + k_6 \geq k$ and $k_3 + k_6 \leq k - 1$, again leading to a contradiction.

Looking back we initially made the following assumption: Let X be the minimal set a vertices such that $d(X) < k$ and X does not contain a k-special-component. Thus we now know that this assumption must have been

incorrect. As there is no smallest set with this property there exists no set with this property and thus the result follows.

□

Lemma 2.4. *Let X and Y be locally k-edge-connected components in G. If there exists no $(k-1)$-edge-cut that disconnects X from Y in G, then $X \cup Y$ is a locally k-edge-connected components in G.*

Proof. Take an arbitrary $x \in X$ and $y \in Y$ and take an arbitrary set of $(k-1)$ edges C. As there exists no $(k-1)$-edge-cut that disconnects X from Y we know that for some $w_i \in X$ and some $w_j \in Y$, w_i is still connected to w_j in $G - C$. As X is a locally k-edge-connected component we know that in $G - C$ any vertex in X is still connected to every other vertex within this component (by Menger's theorem). The same holds for Y. Thus x is connected to w_i and y is connected to w_j. Now x is connected to y, as x is connected to w_i, which is connected to w_j, which is connected to y. As x, y and C were arbitrary we have that for every vertex $u \in X$ and every vertex $v \in Y$ we have $\lambda(u, v; G) \geq k$. Thus $X \cup Y$ is a locally k-edge-connected component of G. □

The following two lemmas show that we can find a unique list containing all k-special-components of a graph by showing that they are maximal locally k-edge-connected components and they do not overlap with each other.

Lemma 2.5. *Let X be a k-special-component of G, then X is a maximal locally k-edge-connected component of G.*

Proof. By definition of k-special-components we know that X is a locally k-edge-connected component and $d(X) < k$. As X is nonempty we can find a vertex $v \in X$. Let $u \notin X$. Every path from u to v requires at least one of the edges from $\text{Cut}(X)$. A set of i edge disjoint paths from u to v will require at least i of the edges from $\text{Cut}(X)$. As $d(X) < k$ there exist at most $k-1$ such edges and thus there exist at most $k-1$ edge disjoint paths from u to v. Thus $\lambda(u, v) \leq k - 1$. As v was arbitrary we see that no vertex $v \notin X$ is locally k-edge-connected to u and thus we can not add v to X while still keeping X a locally k-edge-connected component. Thus X is a maximal k-edge-connected component of G. □

Corollary 2.6. *If U and X are k-special-components of G, then either $U = X$ or $U \cap X = \emptyset$.*

Proof. Assume that $U \cap X \neq \emptyset$ and thus that there exists a vertex $u \in U \cap X$. As U and X are maximal locally k-edge-connected components of G they

both contain all vertices v for which $\lambda(u,v) \geq k$ and no vertex such that $\lambda(u,v) \leq k-1$. This follows from the fact that local edge connectivity is an equivalence relation. Thus, we must have $U = \{v \in V | \lambda(u,v) \geq k\} = X$. $\square$

The following lemmas give results on the local connectivity within k-special-components. For these lemma's let $m := \lfloor \frac{k}{2} \rfloor + 1$ and $m' := \lceil \frac{k}{2} \rceil + 1$.

Lemma 2.7. *Let G be a $(k-1)$-edge-connected graph and let X be a k-special-component of G. Then X is an m-edge-connected component.*

Proof. Let us take two arbitrary distinct vertices $u, v \in X$. As X is a k-special-component, X is a locally k-edge-connected component and $d(X) \leq k-1$. As u is locally k-edge-connected to v in G we can find k edge independent paths from u to v in G. As $u, v \in X$ we know that any of the paths from u to v that are not fully contained in X use at least two edges from X to $\overline{X}$. As we have at most $k-1$ edges going from X to $\overline{X}$, at most $\lfloor \frac{k-1}{2} \rfloor$ of the k edge-disjoint paths are not fully contained in X. Thus at least $k - \lfloor \frac{k-1}{2} \rfloor = \lfloor \frac{k}{2} \rfloor + 1 = m$ of these paths are fully contained in X.

As u and v were arbitrary, it follows that X is m-edge-connected component. $\square$

Lemma 2.8. *Let G be a $(k-1)$-edge-connected graph and let X be a k-special-component of G. Let k be odd and let every vertex of G have a degree of at least $k+1$. Then X contains an $(m+1)$-edge-connected component Z with $|Z| \geq k$.*

Proof. For $k = 1$ this follows from the fact that X is a finite connected component of G where every vertex has a degree of at least two. Assume to the contrary that X does not contain a 2-edge-connected component Z with $|Z| \geq 1$, thus X contains no cycle and therefore X is a tree. Thus looking at the graph induced by X, we must have $|V| = |E| + 1 > |E|$. However as each vertex has a degree of at least two we must have $|E| = \sum_{v \in V} d(v)/2 \geq \sum_{v \in V} 1 = |V|$. This contradicts $|V| > |E|$ and thus X contains an 2-edge-connected component Z with $|Z| \geq 1$.

Now assume $k \geq 2$. X is a m-edge-connected component by Lemma 2.7. Let G' be the graph induced by X. We examine the $(m+1)$-special-components of G'.

By Lemma 2.3 we know that every vertex set $Y \subsetneq X$ either contains a $(m+1)$-special-component in G' or satisfies $d(Y; G') \geq m+1$. Thus, if G' contains no $(m+1)$-special-components we have that $d(Y; G') \geq m+1$ for all $Y \subsetneq X$. From this it follows that X is an $(m+1)$-edge-connected component. The result follows from Lemma 2.1, $|X| \geq k$.

Now assume G' does contain $(m+1)$-special-components. Let Y be one of these $(m+1)$-special-components. We want to place a bound on $d(Y;G')$ by using the degree of its vertices. Just as in Lemma 2.1, we know that any vertex in Y can have at most $|Y|-1$ edges to other vertices in Y. Thus, each vertex has at least $d(v;G')+1-|Y|$ edges to vertices in $X \setminus Y$. We know that every vertex of G has a degree of at least $k+1$ in G. As $d(X)=k-1$, we know that G has at most $k-1$ edges incident to vertices of X going to vertices outside of X. These edges are not present in G'. Thus the total degree of a set of vertices $Y \subsetneq X$ in G' can be up to $k-1$ smaller than their degree in G. We get

$$\begin{aligned} d(Y;G') &\geq \sum_{v\in Y}(d(v;G')+1-|Y|) = (1-|Y|)|Y| + \sum_{v\in Y} d(v;G') \\ &\geq (1-|Y|)|Y| + |Y|(k+1) - (k-1) = -|Y|^2 + (k+2)|Y| - (k-1) \end{aligned}$$

For $|Y|=2$, we get $d(Y;G') \geq -4+2(k+2)-(k-1) = k+1$. For $|Y|=k$ we get $d(Y;G') \geq -k^2+k(k+2)-(k-1) = k+1$. As our function is a parabola in $|Y|$, we know that $d(Y;G') \geq k+1$ when $|Y| \in [2,k]$.

As Y is an $(m+1)$-special-component in G', we know that $d(Y;G') \leq m < k+1$ and thus we must have that $|Y| \notin [2,k]$. If $|Y| \geq k+1$, $|Y|$ is an $(m+1)$-edge-connected component with $|Y| \geq k$ and we are done. Thus we now only need to prove that $|Y| \neq 1$.

Assume to the contrary that $|Y|=1$. As Y is a $(m+1)$-special-component in G', we know $d(Y;G') \leq m$. We know $d(Y;G') = d(Y, X \setminus Y; G)$, and thus $d(Y, X \setminus Y; G) \leq m$. We know that $d(Y;G) = d(Y,\overline{Y};G) = d(Y,\overline{X};G) + d(Y, X \setminus Y; G)$ and thus $d(Y,\overline{X};G) \geq d(Y;G) - m$. We know $d(X;G) = k-1$ as X is a k-special-component of the $(k-1)$-edge-connected graph G. As $d(X;G) = d(X,\overline{X};G) = d(Y,\overline{X};G) + d(X \setminus Y, \overline{X};G)$, we have $d(X \setminus Y, \overline{X};G) = d(X;G) - d(Y,\overline{X};G) \leq k-1+m-d(Y;G)$.

Finally, we note

$$\begin{aligned} d(X \setminus Y;G) &= d(X \setminus Y, Y;G) + d(X \setminus Y, \overline{X};G) \\ &= d(Y, X \setminus Y;G) + d(X \setminus Y, \overline{X};G) \\ &\leq m + (k-1+m-d(Y;G)) = 2m+k-1-d(Y;G) \end{aligned}$$

Now as $|Y|=1$ we know $d(Y;G) \geq k+1$ and we get $d(X \setminus Y;G) \leq 2m-2 = 2\left\lfloor \frac{k}{2} \right\rfloor$. When k is odd this gives $d(X \setminus Y;G) \leq k-1$. However this leads to a contradiction as follows: As $Y \subsetneq X$ the vertex $v \in Y$ must be locally k-edge-connected to every other vertex in X. Thus v has at least k edge-disjoint paths to some other vertex $u \in X \setminus Y$ (as $|X| \geq k$ we always

have some vertex $u \in X \setminus Y$ as $k \geq 2$). However each of these paths uses at least one of the edges from $\text{Cut}(X \setminus Y)$ of which there are at most $k-1$ and thus there cannot be k such edge disjoint paths. Thus we have proven $|Y| \neq 1$.

□

Corollary 2.9. *Let G be a $(k-1)$ edge connected graph and let X be a k-special-component of G. Let every vertex of G have a degree of at least $2\left\lceil \frac{k}{2} \right\rceil$. Then X contains an m'-edge-connected component Z with $|Z| \geq k$.*

Lemma 2.10. *Assume every vertex in G has at least degree $2\left\lceil \frac{k}{2} \right\rceil$. Let X be a k-special-component and let $Z \subseteq X$ be an m'-edge-connected component, with $|Z| \geq k$. There exists a vertex $u \in Z$, such that $N(u) \subseteq X$.*

Proof. We know that X is a k-special-component. Thus, $d(X) \leq k-1$. By Corollary 2.9 the graph induced by X contains a m'-edge-connected component Z with $|Z| \geq k$. We have $k-1$ outgoing edges and Z contains at least k vertices. Therefore there exists a vertex $u \in Z$ that has all its neighbours within X. □

Lemma 2.11. *Assume that every vertex in G has at least degree $2\left\lceil \frac{k}{2} \right\rceil$. Let $L_1, \ldots, L_m$ be a set of k-special-components. There exist vertices $u_i, v_i \in L_i$ such that u_i and v_i satisfy the following properties:*

1. *u_i, v_i in L_i,*
2. *$(u_i, v_i) \in E$,*
3. *$(v_i, u_j) \notin E$ for all vertices u_i and v_j with $i \neq j$,*
4. *There exist m' edge-disjoint paths from u_i to v_i in the graph induced by L_i.*

Proof. To find these vertices we note that Lemma 2.10 gives us a vertex u_i with $N(u_i) \subseteq X$, for every $i \in \{1, \ldots, m\}$. We choose v_i to be another vertex within the m'-edge-connected component of Lemma 2.10 such that the edge (u_i, v_i) exists. Note that we can always find such an edge (u_i, v_i) as u_i has at least m' neighbours within the m'-edge-connected component. Each of these neighbours is a suitable candidate for v_i. Choosing u_i and v_i this way, all the above properties hold. Here properties 1 and 2 follow directly from our choices of u_i and v_i. Property 3 follows immediately from the fact that all the neighbours of u_i are containing within L_i. As k-special-components do not overlap we know that for any vertex $v_j \in L_j$ with $j \neq i$ we must have $v_j \notin L_i$ and thus $v_j \notin N(u_i)$. Property 4 follows from the fact that u_i and v_i are contained in a m'-edge-connected component. □

2.2 Algorithm and proof of correctness

Algorithm 2 is our proposed approximation algorithm for increasing a graphs edge connectivity while keeping the degree of every vertex the same. We can use this algorithm as a subroutine for creating a minimum weight k-edge-connected d-regular graph as is done in Algorithm 3. The correctness of Algorithm 3 is obvious provided that Algorithm 2 works correctly. Thus in this section we prove Algorithm 2 successfully increases the edge connectivity of a graph by 1. Afterwards we analyze the approximation ratio of this algorithm. First, we prove a simple lemma that Algorithm 2 does not change the degree of any vertex. In this chapter we let $L_1^k(G), \ldots, L_m^k(G)$ denote the k-special-components of a graph G, where $m = m(G)$ denotes the number of k-special-components of G. Similarly we let u_i^k and v_i^k for $i = \{1, \ldots, m\}$ denote the vertices with the properties as in Lemma 2.11.

input : undirected complete graph $G = (V, E)$, edge weights w, $d, k \in N$ with $d \geq 2\left\lceil \frac{k}{2} \right\rceil$, a $(k-1)$-edge-connected simple subgraph G' of G satisfying $d(v; G') \geq d$ for all vertices $v \in V$
output: k-edge-connected simple subgraph R of G with $d(v; R) = d(v; G')$ for all vertices $v \in V$

1 Find $L_1^k(G'), \ldots, L_m^k(G')$
2 $m \leftarrow m(G')$
3 for $i = 1, \ldots, m$ find vertices u_i^p and v_i^p
4 compute MST of G'
5 Duplicate each edge of MST and take shortcuts to obtain a Hamiltonian cycle H
6 Take shortcuts to obtain from H a Hamiltonian cycle H' trough $u_1^k, \ldots, u_m^k$, assume w.l.o.g. that H' traverses the vertices in the order $u_1^k, u_2^k, \ldots, u_m^k, u_1^k$
7 $Q \leftarrow \{e_1^k, e_2^k, \ldots, e_m^k\}$ with $e_i^k = (u_i^k, v_i^k)$
8 $S \leftarrow \{(u_i^k, v_{i+1}^k) | i \in \{1, \ldots, m\}, m+1 = 1\}$
9 $R = (G' \setminus Q) \cup S$

Algorithm 2: Increasing edge connectivity of a graph while preserving vertex degrees

Lemma 2.12. *R as computed by Algorithm 2 is a simple graph with $d(v; R) = d(v; G')$ for all vertices $v \in V$*

Proof. By our choice of the vertices v_i, we do not add any edge to G' that is already contained in G. This is because each v_i is chosen such that it does not

input : undirected complete graph $G = (V, E)$, edge weights w, $d, k \in N$ with $d \geq 2\left\lceil \frac{k}{2} \right\rceil$
output: k-edge-connected d-factor R of G

Compute a minimum-weight d-factor d-F of G
Compute a 2 edge connected graph d-factor G' of G using the algorithm of [5].
for $p = 2 \dots k-1$ **do**
    Apply algorithm 2 to create a $p+1$ edge connected graph d-factor G'' of G'
    $G' \leftarrow G''$
end

Algorithm 3: Creating a k-edge-connected d-regular graph with bounded weight

already have an edge to the vertex u_{i-1}, which is the edge the algorithm adds in replacement of (u_i, v_i). As G' was a simple graph it did not have duplicate edges, we also did not add any new ones, and thus R is a simple graph. We obtain R from G' by removing one edge incident to each u_i and v_i, and then adding one edge incident to each u_i and v_i. Therefore $d(v; R) = d(v; G')$ is satisfied for all vertices $v \in V$. Thus all the degrees of vertices in R are the same as those of G'. □

Lemma 2.13. *Let the vertices u_i and v_i be chosen by Algorithm 2. In $G-Q$, there does not exist a set of $k-1$ edges that disconnects both u_i from v_i and u_j from v_j when $i \neq j$.*

Proof. Assume that we can find a set of $k-1$ edges C disconnecting both u_i^k from v_i^k and u_j^k from v_j^k in $G-Q$. According to Lemma 2.8 we have that we can find at least $\left\lceil \frac{k}{2} \right\rceil + 1$ edge disjoint paths from u_i to v_i within L_i in G. After removing the edge (u_i, v_i) we still have at least $\left\lceil \frac{k}{2} \right\rceil$ disjoint paths. Thus our edge cut contains at least $\left\lceil \frac{k}{2} \right\rceil$ edges within L_i in G'. The exact same thing can be said of L_j. Thus for an edge cut C to disconnect both u_i from v_i and u_j from v_j we need to have that $|C| \geq 2\left\lceil \frac{k}{2} \right\rceil \geq k$. As C was an arbitrary cut that disconnects both u_i from v_i and u_j from v_j, we know that any such cut needs to have at least k edges, and thus is not a $k-1$ edge cut. □

Now we prove that when applying Algorithm 2 to a $(k-1)$-edge-connected graph the result is k-edge-connected in order to prove the correctness of Algorithm 2.

First of we prove that the graph remains $(k-1)$-edge-connected after removing Q as we do not remove edges between components and we remove at most one edge per k-edge-connected component of our previous result. Thus our new graph is $(k-1)$-edge-connected even without adding the edges (u_i, v_{i+1}).

Lemma 2.14. *Let G be a $(k-1)$-edge-connected k-regular graph graph, $G-Q$ with Q chosen as in Algorithm 2 is $(k-1)$-edge-connected.*

Proof. Assume we remove only one edge (u_i, v_i) in Q from some locally k-edge-connected-component. In G we have $d(X) \geq k-1$ for every vertex set X. After removal of the edge (u_i, v_i), the only sets X for which $d(X)$ changes are the ones with $u_i \in X$ and $v_i \in \overline{X}$ or vica versa. However as u_i is locally k-edge-connected to v_i in G we have $d(X) \geq k$ for every such X, and thus now have $d(X) \geq k-1$ as we only removed one edge from $\mathrm{Cut}(X)$. Thus after removing an arbitrary (u_i, v_i) edge in Q from G, we have $d(X) \geq k-1$ for every vertex set X and thus the graph is still $(k-1)$-edge-connected.

Now we just need to prove that removal of other edges (u_j, v_j) of Q in components L_j does not destroy the local k-edge-connectedness property of other components L_i. To be more precise we need that u_i is locally k-edge-connected to v_i in $G-L+(u_i, v_i)$ where $L \subseteq Q$. This is equivalent to showing that there is no k-edge-cut disconnecting u_i from v_i in $G-L+(u_i, v_i)$. This follows from Lemma 2.13.

Thus u_i is still locally k-edge-connected to v_i in $G-L+(u_i, v_i)$. Thus we can remove the edge (u_i, v_i) from $G-L+(u_i, v_i)$ while keeping the graph $(k-1)$ edge connected. We can now iteratively remove edges (u_i, v_i) from G while keeping the graph $(k-1)$-edge-connected thus eventually attaining the graph $G-Q$ which is then also proven $(k-1)$-edge-connected.

□

Note that by using Lemma 2.13 to prove Lemma 2.14 we implicitly used the assumption that $d \geq 2\left\lceil \frac{k}{2} \right\rceil$. It is in fact possible to prove Lemma 2.14 without this assumption using Lemma 2.7. For this we need to prove that there is no k-edge-cut disconnecting u_i from v_i in $G-L+(u_i, v_i)$.

Lemma 2.15. *Let u_i, v_i, $G-Q$ and L be defined as in Lemma 2.14. There is no k-edge-cut disconnecting u_i from v_i in $G-L+(u_i, v_i)$ even when the assumption that $d \geq 2\left\lceil \frac{k}{2} \right\rceil$ is dropped.*

Proof. By Lemma 2.7 we know that for any i, the graph induced by L_i is $(\left\lfloor \frac{k}{2} \right\rfloor + 1)$-edge-connected. After removal of an edge (u_i, v_i) the graph induced by L_i is still be at least $\left\lfloor \frac{k}{2} \right\rfloor$-edge-connected.

Now assume to the contrary that we can find a $k-1$ edge cut C disconnecting u_i from v_i in $G - L + (u_i, v_i)$. As u_i is locally k-edge-connected to v_i in G we can find k edge disjoint paths from u_i to v_i in G.

According to Lemma 2.7 we have that we can find at least $\lfloor \frac{k}{2} \rfloor + 1$ edge disjoint paths from u_i to v_i within L_i. Thus our edge cut contains at least $\lfloor \frac{k}{2} \rfloor + 1$ edges within L_i. As each other component L_j with $j \neq i$ has at least $\lfloor \frac{k}{2} \rfloor$ edge disjoint paths from u_j to v_j within L_j we need to remove at least $\lfloor \frac{k}{2} \rfloor$ edges within L_j to disconnect u_j to v_j. However as $\lfloor \frac{k}{2} \rfloor + 1 + \lfloor \frac{k}{2} \rfloor \geq k$ we can not find an $k-1$ edge cut that disconnects both u_i from v_i and u_j from v_j in $G - Q + (u_i, v_i)$. Thus we have that u_j and v_j must be connected in $G - L + (u_i, v_i) - C$.

We know u_i and v_i are locally k-edge-connected in G. Thus our $k-1$ edge cut C disconnecting u_i from v_i in $G - L + (u_i, v_i)$ does not disconnect u_i from v_i in G. Thus u_i and v_i are connected in $G - C$ and there must be a path P from u_i to v_i in $G - C$. Now if this path uses an edge (u_j, v_j) we know that u_j and v_j must be connected in $G - L + (u_i, v_i) - C$ and thus we can find a path from u_j to v_j in $G - Q + (u_i, v_i) - C$. Thus for each edge $(u_j, v_j) \in P$ we can remove that edge and add a path from (u_j, v_j) containing only edges from $G - L + (u_i, v_i) - C$. As the edges (u_j, v_j) are the only edges in P that are contained in $G - C$ but not contained in $G - L + (u_i, v_i) - C$, we have that this procedure creates a path P' from u_i to v_i in $G - Q + (u_i, v_i) - C$. Thus we have found a contradiction, as C is not disconnecting u_i from v_i in $G - L + (u_i, v_i)$ as we initially assumed.

□

Lemma 2.16. *In $R = G + S - Q$ as created by Algorithm 2 we have that u_i and v_{i+1} are locally k-edge-connected.*

Proof. In $G - Q$ we have that u_i and v_{i+1} are still $(k-1)$-edge-connected by Lemma 2.14. Thus there are $k-1$ edge independent paths from u_i to v_{i+1} in $G - Q$. In R we have an additional edge (u_i, v_{i+1}) that is not contained in $G - Q$ and is thus a k^{th} edge path from u_i to v_{i+1} that is edge-disjoint to the other $k-1$ paths. □

Lemma 2.17. *In $R = G + S - Q$ as created by Algorithm 2 we have that u_i and u_j are locally k-edge-connected for all $i, j \in \{1, \ldots, m\}$.*

Proof. Assume to the contrary that there exists a $k-1$ edge cut C disconnecting some vertex u_i from some vertex u_j in R. As $G - Q$ is still $k-1$ edge connected we know none of the $k-1$ removed edges are newly introduces edges of S. We want to find a contradiction by using the new paths $P_1 = u_{i+1}, u_{i+2}, \ldots, u_{j-2}, u_{j-1}$ and $P_2 = u_{i-1}, u_{i-2}, \ldots, u_{j+2}, u_{j+1}$ to get from

u_i to u_j. From Lemma 2.13 we know that if a $k-1$ edge cut disconnects u_k from v_k then it is impossible for that cut to also disconnects u_l from v_l for any $l \neq k$. Thus there can be at most one pair (u_k, v_k) that is disconnected by the $k-1$ edge cut. However this means that either path P_1 or path P_2 connects u_i to u_j. This follows from the fact that we have edges going from each u_i to each v_{i+1} and for only a single k we can not find a path from v_k to u_k to connect these (u_i, v_{i+1}) edges into a path (only one path P_1 or P_2 requires a path from v_k to u_k, thus the other path is a valid path contained in $R-C$). We have now proven that u_i is connected to u_j in $G-C$ which is a contradiction. Thus we can not find a $k-1$ edge cut disconnecting some vertices u_i and u_j from each other. Thus for all $i, j \in \{1, \ldots, m\}$ we have that u_i and u_j are locally k-edge-connected with each other in the new graph. □

Lemma 2.18. *In $R = G + S - Q$ as created by Algorithm 2 we have that L_i and L_j are locally k-edge-connected for all $i, j \in \{1, \ldots, m\}$.*

Proof. To prove this lemma we want to show that every vertex $v \in L_i$ is locally k-edge-connected to every vertex $u \in L_j$ for arbitrary i and j. Lemma 2.17 already gives us this result if we choose $v = u_i$ and $u = u_j$. We first extend this by adding the option of choosing vertices v_i. We know that for all i, u_{i-1} and v_i are locally k-edge-connected. We also already know u_{i-1} is locally k-edge-connected to any u_j. Thus using the fact that k-edge-connectivity is transitive, we find that all vertices v_i are locally k-edge-connected to all u_i and all other v_j.

Now take some vertex $v \in L_i$ with $v \neq u_i$ and $v \neq v_i$. We need to prove that it is locally k-edge-connected to u_i. Let us assume to the contrary that v is not locally k-edge-connected to u_i and thus that we can find a $k-1$ edge cut C disconnecting v from u_i. As $G-Q$ is $(k-1)$-edge-connected we know all $k-1$ of the edges in C are not contained in S. Thus we can also take this cut in the original graph G. We know v and u_i are locally k-edge-connected in G. Thus there is still a path from v to u_i in $G-C$. Necessarily it uses one or more edges $(u_j, v_j) \in Q$, as they are the only edges contained in R that are not contained in $G-Q$ (otherwise the path would exist in $R-C$ and thus C would not be an edge cut disconnecting v from u_i). Let us write the path from v to u_i as $P_1(u_j, v_j)P_2$. As we already showed each u_j and v_j are locally k-edge-connected in $G-Q+S$, we know that u_j and v_j are still connected in $G-Q+S-C$. Thus we can find a path P_3 contained in $G-Q+S-C$, that connects u_i and v_i. Thus we have a path $P_1P_3P_2$ going from v to u_i. Note that P_3 contains no edges $(u_i, v_i) \in Q$ and thus this new path $P_1P_3P_2$ has a lower number of edges contained in Q compared to the old path $P_1(u_j, v_j)P_2$. We can repeat this procedure for any other edge $(u_j, v_j) \in Q$ occurring in the new path $P_1P_3P_2$ until the entire path from v to

u_i only uses edges within $G+S-Q$. Thus contradicting our assumption that a $k-1$ edge cut C disconnecting v from u_i exists. Thus we have that that any vertex $v \in L_i$ is locally k-edge-connected to u_i. We have by transitivity of k-edge-connectivity that every vertex $v \in L_i$ is locally k-edge-connected to every vertex $u \in L_j$ for arbitrary i and j. □

Theorem 2.19. *The graph $R = G + S - Q$ as created by Algorithm 2 is k-edge-connected.*

Proof. Our previous Lemma already gave us the result that L_i and L_j are locally k-edge-connected for all $i, j \in \{1, \ldots, m\}$.

Thus we are only left to look at the vertices not contained in any components L_i. Let v be a vertex not contained in one of the components and assume that we have a $k-1$ edge cut C disconnecting it from a part of the graph. Take the largest set X such that $v \in X$ and every vertex in X is connected to v in $G-C$. We now know $d(X; R-C) = 0$ as otherwise we would have taken a larger set X. As $|C| \leq k-1$ we have $d(X; R) \leq k-1$. From Lemma 2.3 we know that $d(X) \geq k$ for any set X that does not contain an k-special-component (as we already know G is $k-1$ edge connected). Thus X has to contain some k-special-component L_i. Thus, after the edge cut, v is still connected to a vertex u within one of the components L_i.

As C was arbitrary we find that there exists no $k-1$ edge cut such that v is disconnected from all vertices of $\bigcup_i L_i$ in G. As to create R from G the only edges we removed were edges connecting two vertices within $\bigcup_i L_i$ we know that this property also holds for the graph $R = G + S - Q$.

We have already proven that $\bigcup_i L_i$ is a locally k-edge-connected component. A single vertex is a trivial locally k-edge-connected component. Thus we can use Lemma 2.4 to find that v is locally k-edge-connected to all vertices in $\bigcup_i L_i$. Finally as local k-edge-connectivity is transitive and v was arbitrary we find that the entire vertex set X is a locally k-edge-connected component in R. From this it immaterially follows that R is a k-edge-connected graph. □

Lemma 2.20. *Assume that the graph R is computed as in Algorithm 2 and let G be the input graph of Algorithm 2. Let MST be the minimum weight spanning tree. We have $w(R) \leq w(G) + 2w(MST)$.*

Proof. Note that by the triangle inequality we have $w(u_i, v_{i+1}) \leq w(u_i, u_{i+1}) +$

$w(u_{i+1}, v_{i+1})$. We have

$$\begin{aligned} w(R) = w(G + S - Q) &= w(G) + \sum_{i=1}^{m} w(u_i, v_{i+1}) - \sum_{i=1}^{m} w(u_{i+1}, v_{i+1}) \\ &\leq w(G) + \sum_{i=1}^{m} w(u_i, u_{i+1}) + w(u_{i+1}, v_{i+1}) - w(u_{i+1}, v_{i+1}) \\ &= w(G) + w(H') \leq w(G) + w(H) \end{aligned}$$

Finally recall that H was created using a MST such that $w(H) \leq 2w(MST)$.

□

Corollary 2.21. *Assume that the graph R is computed as in Algorithm 2 and let G be the input graph of Algorithm 2. If G is a connected graph (that is, if $k \geq 2$), then $w(R) \leq 3w(G)$.*

Proof. Let MST be the minimum weight spanning tree. Note that $w(MST) \leq w(G)$ as G is a connected graph and MST is the minimum weight connected graph (due to having non-negative edge costs, the minimum weight connected graph must be a tree). The result now easily follows from Lemma 2.20 as $w(R) \leq w(G) + 2w(MST) \leq 3w(G)$. □

Theorem 2.22. *Let $d \geq k \geq 2$. Let R be the k-edge-connected d-factor computed as in Algorithm 3 and let $G_{k,d}^{OPT}$ be the minimum weight k-edge-connected d-factor. We have $w(R) \leq (2k-1)w(G_{k,d}^{OPT})$.*

Proof. Let $G_{-,d}$ be a minimum weight d-factor and let MST be the minimum weight spanning tree. Note that the initial 2-edge-connected d-factor $G_{2,d}$ satisfies $w(G_{2,d}) \leq w(G_{-,d}) + 2w(MST)$ [5]. Note that the algorithm will have a total of $k-2$ iterations, and note that by Lemma 2.20 each iterations adds a weight of at most $2w(MST)$ to the graph from the previous iteration. Thus we get $w(R) \leq w(G_{-,d}) + 2w(MST) + (k-2)2w(MST) = w(G_{-,d}) + (2k-2)w(MST)$. We have that $w(MST) \leq w(G_{k,d}^{OPT})$ as $G_{k,d}^{OPT}$ is a connected graph and MST is a minimum weight connected graph. Also $G_{k,d}^{OPT}$ is a d-factor, while $G_{-,d}$ is a minimum weight d-factor and thus $w(G_{-,d}) \leq w(G_{k,d}^{OPT})$. From these facts the result now follows. □

2.3 Generalization

Algorithm 3 can easily be generalized to create k-edge-connected graphs that satisfy somewhat arbitrary degree requirements giving us Algorithm 4. The

correctness of this algorithm is now trivial as there exist algorithms for creating a minimum cost perfect b-matching in polynomial time, and we have already proven that 2 works correctly in increasing the edge connectivity of a graph without changing the degree of any vertex.

input : undirected complete graph $G = (V, E)$, edge weights w, $k \in N$, and a vector $b = (b_v : v \in V)$ defining the degree requirements with the restriction that $b_v \geq 2\left\lceil \frac{k}{2} \right\rceil$ for all $v \in V$
output: k-edge-connected spanning subgraph R of G, satisfying $d(v; R) = b_v$ for all vertices $v \in V$

```
Compute a minimum cost perfect b-matching G' of G
for p = 1 ... k do
    Apply algorithm 2 to create a p-edge-connected graph G'' of G'
    satisfying d(v; R) = b_v for all vertices v ∈ V
    G' ← G''
end
```

Algorithm 4: Creating a k-edge-connected spanning subgraph, satisfying degree constraints, with bounded weight

Theorem 2.23. *Let R be the k-edge-connected spanning subgraph, satisfying degree constraints, as was created with Algorithm 4 and let $G_{k,b}^{OPT}$ be the minimum weight k-edge-connected b-matching. We have $w(R) \leq (2k + 1)w(G_{k,b}^{OPT})$.*

Proof. Let $G_{-,b}$ be a minimum weight b-matching and let MST be the minimum weight spanning tree. Note that the algorithm will have a total of k iterations, and note that by Lemma 2.20 each iterations adds a weight of at most $2w(MST)$ to the graph from the previous iteration. Thus we get $w(R) \leq w(G_{-,b}) + 2kw(MST)$. We have that $w(MST) \leq w(G_{k,b}^{OPT})$ as $G_{k,b}^{OPT}$ is a connected graph and MST is a minimum weight connected graph. Also $G_{k,b}^{OPT}$ is a b-matching, while $G_{-,b}$ is a minimum weight b-matching and thus $w(G_{-,b}) \leq w(G_{k,b}^{OPT})$. From these facts the result now follows. □

Chapter 3

Second Algorithm for Approximating d-Regular k-Edge-Connected Subgraph

Assume we have a k-edge-connected graph $G_{k,k} = (E_{k,k}, V_{k,k})$ where each vertex has maximum degree k and a d-regular graph $G_{-,d} = (E_{-,d}, V_{-,d})$. We wish to create a k-edge-connected d-regular graph $G_{k,d}$. We do so by starting with $G_{-,d}$, which is d-regular, and adding edges of $G_{k,k}$ without changing the degree of any vertex (by removing other edges) until the new graph $G_{k,d}$ is also k-edge-connected. (It is impossible for $G_{k,d}$ not to become k-edge-connected in this process, as after adding all edges of $G_{k,k}$ we would have that $G_{k,k}$ is a subgraph of $G_{k,d}$ thus immaterially implying that $G_{k,d}$ is k-edge-connected.) First note that if $d = k$ then $G_{k,k}$ is d-regular as each vertex has maximum degree k and for k-edge-connectedness each vertex has to have at least degree k. Thus we can take $G_{k,d} = G_{k,k}$. Thus we can assume $d > k$ which implies that for each vertex $v \in G_{-,d}$ there is at least one edge $e = (v, u) \in G_{-,d}$ with $e \notin G_{k,k}$. Also note that as long as $G_{k,k}$ is not a subgraph of $G_{-,d}$ (at which point we are surely done) there always exists some edge $e \in G_{k,k}$ with $e \notin G_{-,d}$.

Algorithm 5 is our approximation algorithm for creating a k-edge-connected d-regular graph for the case where $d \geq 2k - 1$. Each loop we add an edge of $G_{k,k}$ to $G_{k,d}$. Then to keep $G_{k,d}$ a d-regular graph we remove two edges in $G_{k,d}$ incident to the newly added edge and add the edge connecting the two endpoints u_2 and v_2 of these newly added edges. To keep the graph simple we require that none of the two newly added edges is already contained in $G_{k,d}$. We shall later show that as long as $G_{k,d}$ is not k-edge-connected and $d \geq 2k - 1$, we can always find edges which satisfy the conditions of this algorithm.

input : undirected complete graph G, edge weights w, $d, k \in N$ satisfying $d \geq 2k-1$, k-edge-connected subgraph $G_{k,k}$ of G where each vertex has maximum degree k, a d-regular subgraph $G_{-,d}$ of G
output: k-edge-connected d-factor $G_{k,d}$ of G

1 $G_{k,d} \leftarrow G_{-,d}$
2 **while** $G_{k,d}$ *is not k-edge-connected* **do**
3 Take an edge $(u_1, v_1) \in G_{k,k}$ with $(u_1, v_1) \notin G_{k,d}$ and add it to $G_{k,d}$
4 Remove some edges $(v_1, u_2), (u_1, v_2)$ from $G_{k,d}$ that are not contained in $G_{k,k}$.
5 Add an edge $(u_2, v_2) \notin G_{k,d}$ to $G_{k,d}$ and go to step 2
6 **end**

Algorithm 5: Second algorithm for creating a k-edge-connected d-regular graph with bounded weight

Lemma 3.1. *The graph $G_{k,d}$ created by algorithm 5 is a d-regular graph.*

Proof. First note that at the start of the algorithm $G_{k,d}$ is d-regular as $G_{-,d}$ is d-regular. Thus we only need to prove that after every iteration of the loop the graph remains d-regular. In this algorithm we add edges (u_1, v_1) and (u_2, v_2) and we remove edges (u_1, v_2) and (u_2, v_1). Thus for for all four of these vertices we remove one adjacent edge and add an adjacent edge, leaving their degree the same. The degree of all other vertices does not change at all during the loop. At the end of the loop every vertex still has the same degree as at the beginning of the loop and thus $G_{k,d}$ is still d-regular. □

Lemma 3.2. *The graph $G_{k,d}$ created by by algorithm 5 is a simple graph.*

Proof. Initially $G_{k,d}$ is a simple graph as $G_{-,d}$ is a simple graph. Any edge we add during the algorithm is an edge that at that time is not already contained within $G_{k,d}$. Thus at any point in the algorithm, $G_{k,d}$ remains a simple graph. □

Lemma 3.3. *The graph $G_{k,d}$ created by by algorithm 5 is k-edge-connected.*

Proof. If the algorithm terminates, this is trivial. □

Lemma 3.4. *Assume $d \geq 2k-1$, if $G_{k,d}$ is not k-edge-connected we can find two edges $e_1 = (u_1, v_1), e_2 = (u_2, v_2)$ satisfying the following conditions:*

1. $e_1, e_2 \notin G_{k,d}$

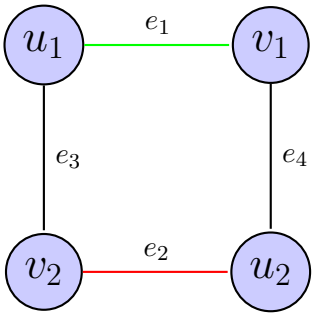

Figure 3.1: Picture for the proof of Lemma 3.4. The cycle we wish to construct in Lemma 3.4.

2. $e_1 \in G_{k,k}$

3. *There exist edges* e_3 *and* e_4*, adjacent to both* e_1 *and* e_2*, satisfying* $e_3, e_4 \in G_{k,d}$ *and* $e_3, e_4 \notin G_{k,k}$ *(w.l.o.g. we can assume* $e_3 = (u_1, v_2)$ *and* $e_4 = (v_1, u_2)$*)*

Proof. If $G_{k,d}$ is not k-edge-connected than we know there exists a vertex set X with $d(X) \leq k-1$ in $G_{k,d}$. As $G_{k,k}$ is k-edge-connected we know $d(X) \geq k$ in $G_{k,k}$. Thus there exists at least one edge $e_1 = (u_1, v_1) \in G_{k,k}$ with $e_1 \notin G_{k,d}$ going from X to $\overline{X}$.

As $G_{k,k}$ has maximum degree k we know that u_1 and v_1 have at most $k-1$ incident edges that are contained in both $G_{k,d}$ and $G_{k,k}$. As $G_{k,d}$ is d-regular we thus have that u_1 and v_1 have at least $d-k+1$ other edges in $G_{k,d}$, all of which are not contained in $G_{k,k}$. Thus we have $d-k+1$ choices for u_2 and v_2 such that $e_3, e_4 \in G_{k,d}$ and $e_3, e_4 \notin G_{k,k}$. We are done if it is possible to choose u_2 and v_2 such that the edge $e_2 = (u_2, v_2)$ is not contained in $G_{k,d}$. As then we have that $e_1, e_2 \notin G_{k,d}$, $e_1 \in G_{k,k}$ and e_1 and e_2 are incident to two edges e_3 and e_4 satisfying $e_3, e_4 \in G_{k,d}$ and $e_3, e_4 \notin G_{k,k}$ thus satisfying all conditions of the lemma.

Our result follows after we have proven that we can find such an edge $e_2 = (u_2, v_2)$.

Thus assume to the contrary that all choices for u_2 and v_2 lead to either the existence of an edge (u_2, v_2) in $G_{k,d}$ or to $u_2 = v_2$. We can find a contradiction by proving that $d(X) \geq k$. Refer to figure 3 as an illustration of the following construction.

Let u_i^* with $i \in \{1, \ldots, 1+d-k\}$ be the possible choices for u_2 and v_i^* with $i \in \{1, \ldots, 1+d-k\}$ be the possible choices for v_2. For any i and j where $u_i^* = v_j^*$ we have the following path from u_1 to v_1: $(u_1, u_i^* = v_j^*, v_1)$. Now remove the vertices for which $u_i^* = v_j^*$ for some i and j from our sets

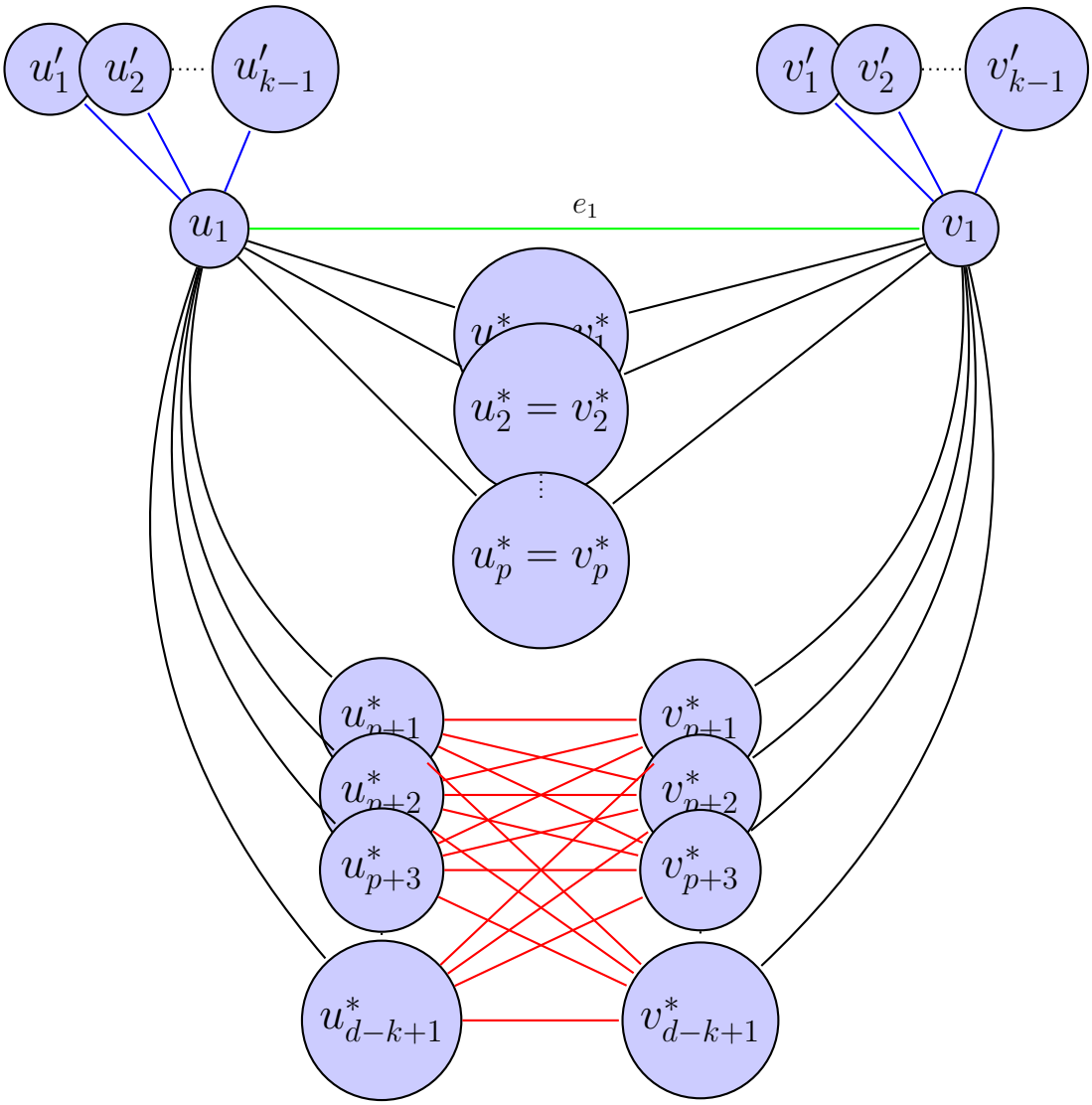

Figure 3.2: Picture for the proof of Lemma 3.4. Blue edges are a part of $G_{d,d}$ and possibly also of $G_{k,d}$, the green edge only exists in $G_{d,d}$ and we prove that at least one of the red edges does not exist in $G_{k,d}$. (There are possibly more than $d-k+1$ vertices connected to u_i or v_i via black edges, although we need only look at the depicted vertices.)

$\{u_i^*|i \in \{1,\ldots,1+d-k\}\}$ and $\{v_j^*|j \in \{1,\ldots,1+d-k\}\}$. Clearly if we found p paths $(u_1, u_i^* = v_j^*, v_1)$ our sets now still have $d-k+1-p$ vertices. Relabel the vertices to $\{u_i^*|i \in \{1,\ldots,1+d-k-p\}\}$ and $\{v_j^*|j \in \{1,\ldots,1+d-k-p\}\}$. Now for these vertices we have the following path from u_1 to v_1: (u_1, u_i^*, v_i^*, v_1) for $i = 1,\ldots,d-k+1-p$ in $G_{k,d}$. We are sure these paths exist as $e_2 = (u_2, v_2) \in G_{k,d}$ for all choices for u_2 and v_2 where $u_2 \neq v_2$ and we already eliminated the possibility of $u_2 = v_2$.

We have thus created a total of $d-k+1$ paths from u_1 to v_1 in $G_{k,d}$. Note that each path uses a single vertex u_i^* and a single vertex u_j^* (though possibly $u_i^* = v_j^*$) and none of these paths have overlapping vertices outside of u_1 and v_1. Thus we have created $d-k+1$ vertex disjoint paths from u_1 to v_1. Clearly these paths are also edge disjoint. As $d \geq 2k-1$ we have $d-k+1 \geq k$ and thus u_1 and v_1 are locally k-edge-connected in $G_{k,d}$.

However as u_1 and v_1 are locally k-edge-connected we must have that any set $Y \subseteq V$ with $u_1 \in Y$ and $v_1 \notin Y$ satisfies $d(Y) \geq k$ in $G_{k,d}$. However this contradicts our initial choice of e_1 as we know the existence of a set $X \subseteq V$ with $u_1 \in X$, $v_1 \notin X$ and $d(X) \leq k-1$ in $G_{k,d}$.

Thus there must exists an edge $e_2 = (u_2, v_2) \notin G_{k,d}$ with $u_2 \neq v_2$. □

Lemma 3.5. *The algorithm terminates in polynomial time.*

Proof. The algorithm eventually terminates, because if we we add an edge from $G_{k,k}$ to $G_{k,d}$, we have increased the number of edges of $G_{k,k}$ that $G_{k,d}$ contains by 1. This is because all the edges we removes from $G_{k,d}$ are not edges contained in $G_{k,k}$. As $G_{k,k}$ has a finite amount of edges we can only do this step a finite number of times ($O(kn)$). Each time we change $G_{k,d}$ we may have to reconsider some of the edges $(u, v) \in G_{k,k}$ that we previously did not need to add to $G_{k,d}$. However as we change $G_{k,d}$ at most once for each edge in $G_{k,k}$, the number of edges to reconsider is also bounded by the number of edges in $G_{k,k}$. Thus we need to check the same edge at most $O(kn)$ times and there are at most $O(kn)$ edges we need to check, each check requires us to check less than $3d$ other edges, thus we can construct $G_{k,d}$ in $O(3dk^2n^2)$ time. (Though with good book keeping, the algorithm's worse case running time can likely be decreased.) □

Lemma 3.6. *Assume $d \geq 2k-1$ and edge weights satisfying the triangle inequality. Then there exists a $(5+\frac{2}{k})$-approximation algorithm for the minimum weight k-edge-connected d-regular graph problem.*

Proof. We already showed that the algorithm runs in polynomial time and that it creates a k-edge-connected d-regular graph in previous lemmas. We are left with proving the approximation ratio. Now for any loop in the

algorithm we have a cycle consisting of four edges (e_1, e_2, e_3, e_4), where $e_1 \in G_{k,k}$ and e_3 are added, while $e_2, e_4 \in G_{-,d}$ are removed from $G_{k,d}$. By the triangle inequality we know that $w(e_3) \leq w(e_4) + w(e_1) + w(e_2)$. Thus we know the weight we added to $G_{k,d}$ equals $w(e_1) + w(e_3) - w(e_2) - w(e_4) \leq 2w(e_1)$. As we alter $G_{k,d}$ at most once for every edge $e_1 \in G_{k,k}$ and at the start of the algorithm we have $G_{k,d} = G_{-,d}$, we have $w(G_{k,d}) \leq w(G_{-,d}) + 2w(G_{k,k})$. Let $G_{k,d}^{OPT}$ be the minimal weight k-edge-connected d-regular graph. For $G_{-,d}$ we can find the minimum weight d-regular graph. Clearly we have $w(G_{-,d}) \leq w(G_{k,d}^{OPT})$ as $G_{k,d}^{OPT}$ is also a d-regular graph and $G_{-,d}$ has minimum weight amongst the d-regular graphs. For $G_{k,k}$ we can find a low weight k-regular k-edge-connected graph that is at most $2 + \frac{1}{k}$ times as expensive as the minimum weight k-edge connected graph. Thus as $G_{k,d}^{OPT}$ is also a k-edge connected graph we have $w(G_{k,k}) \leq (2 + \frac{1}{k})w(G_{k,d}^{OPT})$. Putting it all together we find that $w(G_{k,d}) \leq w(G_{-,d}) + 2w(G_{k,k}) \leq (5 + \frac{2}{k})w(G)$ □

3.1 Vertex Connectivity Case

Algorithm 5 does in fact also work when we replace edge connectivity with vertex connectivity resulting in Algorithm 6.

input : undirected complete graph G, edge weights w, $d, k \in N$ satisfying $d \geq 2k - 1$, k-vertex-connected subgraph $G_{k,k}$ of G where each vertex has maximum degree k, a d-regular subgraph $G_{-,d}$ of G
output: k-vertex-connected d-factor $G_{k,d}$ of G

1 $G_{k,d} \leftarrow G_{-,d}$
2 **while** $G_{k,d}$ *is not k-vertex-connected* **do**
3 Take an edge $(u_1, v_1) \in G_{k,k}$ with $(u_1, v_1) \notin G_{k,d}$ and add it to $G_{k,d}$
4 Remove some edges $(v_1, u_2), (u_1, v_2)$ from $G_{k,d}$ that are not contained in $G_{k,k}$.
5 Add an edge $(u_2, v_2) \notin G_{k,d}$ to $G_{k,d}$ and go to step 2
6 **end**

Algorithm 6: Algorithm for creating a k-vertex-connected d-regular graph with bounded weight

This time we assume we have a k-vertex-connected graph $G_{k,k} = (E_{k,k}, V_{k,k})$ where each vertex has maximum degree k, rather than a k-edge-connected graph. We still have our d-regular graph $G_{-,d} = (E_{-,d}, V_{-,d})$ and we now wish to create a k-vertex-connected d-regular graph $G_{k,d}$. Our algorithm is

exactly the same as before except now checking if $G_{k,d}$ is k-vertex-connected rather then checking for edge connectivity:

The only Lemmas that used the fact that $G_{k,k}$ is k-edge-connected are Lemma 3.4 and Lemma 3.6.

For Lemma 3.4 we note that we found k vertex disjoint paths rather than just k edge disjoint paths from u_1 to v_1. Additionally we now start out with a u_i and a v_i that are disconnected by a $k-1$ vertex cut rather than by a $k-1$ edge cut. However outside of these details the proof remains the same. For completeness we have added the proof of Lemma 3.4 for the k-vertex connected case below.

Lemma 3.7. *Assume $d \geq 2k-1$, if $G_{k,d}$ is not k-vertex-connected we can find two edges $e_1 = (u_1, v_1), e_2 = (u_2, v_2)$s satisfying the following conditions:*

1. $e_1, e_2 \notin G_{k,d}$
2. $e_1 \in G_{k,k}$
3. *e_1 and e_2 are incident to two edges e_3 and e_4 satisfying $e_3, e_4 \in G_{k,d}$ and $e_3, e_4 \notin G_{k,k}$ (w.l.o.g. we can assume $e_3 = (u_1, v_2)$ and $e_3 = (v_1, u_2)$*

Proof. As $G_{k,d}$ is not k-vertex-connected there must be some edges $e_1 = (u_1, v_1)$ contained in $G_{k,k}$ that are not contained in $G_{k,d}$.

Recall that $G_{k,d}$ is k-vertex-connected if $|V| > k$ (which should always be true due to the minimum degree of each vertex) and $|N(X)| \geq k$ for all nonempty $X \subseteq V$ with $|X| \leq |V| - k$. As $G_{k,d}$ is not k-vertex-connected we can thus find a set $X \subseteq V$ with $|X| \leq |V| - k$ and $|N(X)| \leq k-1$. As $G_{k,k}$ is k-vertex-connected we also know $|N(X)| \geq k$ in $G_{k,k}$. Thus there exists some edge going from X to $\overline{X}$ contained in $G_{k,k}$ that is not contained in $G_{k,d}$. Choose one such edge to be $e_1 = (u_1, v_1)$ (Thus we ensured $e_1 \in G_{k,k}$ and $e_1 \notin G_{k,d}$.) We can assume w.l.o.g. that $u_1 \in X$ and $v_1 \notin X$.

As $G_{k,k}$ has maximum degree k we know that u_1 and v_1 have at most $k-1$ incident edges that are contained in both $G_{k,d}$ and $G_{k,k}$. As $G_{k,d}$ is d-regular we thus have that u_1 and v_1 have at least $d-k+1$ other edges in $G_{k,d}$, all of which are not contained in $G_{k,k}$. Thus we have $d-k+1$ choices for u_2 and v_2 such that $e_3, e_4 \in G_{k,d}$ and $e_3, e_4 \notin G_{k,k}$. We are done if it is possible to choose u_2 and v_2 such that the edge $e_2 = (u_2, v_2)$ is not contained in $G_{k,d}$ and $u_2 \neq v_2$ (this latter condition is required as $G_{k,d}$ must remain a simple graph). As then we have that $e_1, e_2 \notin G_{k,d}$, $e_1 \in G_{k,k}$ and e_1 and e_2 are incident to two edges e_3 and e_4 satisfying $e_3, e_4 \in G_{k,d}$ and $e_3, e_4 \notin G_{k,k}$ thus satisfying all conditions of the lemma.

Our result follows after we have proven that we can find such an edge $e_2 = (u_2, v_2) \notin G_{k,d}$. Thus assume to the contrary that all choices for u_2 and v_2 lead to either the existence of an edge (u_2, v_2) in $G_{k,d}$ or to $u_2 = v_2$.

Let u_i^* with $i \in \{1, \ldots, 1 + d - k\}$ be the possible choices for u_2 and v_i^* with $i \in \{1, \ldots, 1 + d - k\}$ be the possible choices for v_2. For any i and j where $u_i^* = v_j^*$ we have the following path from u_1 to v_1: $(u_1, u_i^* = v_j^*, v_1)$. Now remove the vertices for which $u_i^* = v_j^*$ for some i and j from our sets $\{u_i^* | i \in \{1, \ldots, 1 + d - k\}\}$ and $\{v_j^* | j \in \{1, \ldots, 1 + d - k\}\}$. Clearly if we found p paths $(u_1, u_i^* = v_j^*, v_1)$ our sets now still have $d - k + 1 - p$ vertices. Relabel the vertices to $\{u_i^* | i \in \{1, \ldots, 1 + d - k - p\}\}$ and $\{v_j^* | j \in \{1, \ldots, 1 + d - k - p\}\}$. Now for these vertices we have the following path from u_1 to v_1: (u_1, u_i^*, v_i^*, v_1) for $i = 1, \ldots, d - k + 1 - p$ in $G_{k,d}$. We are sure these paths exist as $e_2 = (u_2, v_2) \in G_{k,d}$ for all choices for u_2 and v_2 where $u_2 \neq v_2$ and we already eliminated the possibility of $u_2 = v_2$.

We have thus created a total of $d-k+1$ paths from u_1 to v_1 in $G_{k,d}$. Note that each path uses a single vertex u_i^* and a single vertex u_j^* (though possibly $u_i^* = v_j^*$) and none of these paths have overlapping vertices outside of u_1 and v_1. Thus we have created $d - k + 1$ vertex disjoint paths from u_1 to v_1. As $d \geq 2k - 1$ we have $d - k + 1 \geq k$ and thus u_1 and v_1 are k-vertex-connected in $G_{k,d}$.

However as u_1 and v_1 are k-vertex-connected we must have that any set $Y \subseteq V$ with $|Y| \leq |V| - k$, $u_1 \in Y$ and $v_1 \notin Y$ satisfies $|N(Y)| \geq k$ in $G_{k,d}$. However this contradicts our initial choice of e_1 as we know $X \subseteq V$ with $|X| \leq |V| - k$, $u_1 \in X$, $v_1 \notin X$ and $|N(X)| \leq k - 1$.

Thus there must exists an edge $e_2 = (u_2, v_2) \notin G_{k,d}$. □

For Lemma 3.6 the only change is that the statement "For $G_{k,k}$ we can find minimum weight k-regular k-edge-connected graph that is at most $2 + \frac{1}{k}$ times as expensive as the minimum k-edge connected graph." no longer holds. However we do know that when $|V| \geq 2k$ there is a $(2 + \frac{k-1}{n} + \frac{1}{k})$-approximation of the minimum k-vertex connected graph. From our requirement that $d \geq 2k - 1$, we automatically have that $|V| \geq 2k$ is satisfied and thus this we can always use this $(2 + \frac{k-1}{n} + \frac{1}{k})$-approximation of the minimum k-vertex connected graph. Leaving the rest of the proof the same we thus end up with a $(5 + \frac{2(k-1)}{n} + \frac{2}{k})$-approximation algorithm for the minimum weight k-vertex-connected d-regular graph problem when $d \geq 2k - 1$.

Chapter 4

Summary and future work

4.1 Summary

In this thesis we have described various problems involving the creation of low cost networks that satisfy certain connectivity requirements, and summarized some of the work that has been done in this area. We then described two approximation algorithms for creating a low weight k-edge-connected d-regular subgraph under the assumption that edge weights satisfy the triangle inequality and proved their correctness.

The first of these algorithms is Algorithm 3, which gives a $(2k-1)$ approximation ratio under the condition that $d \geq 2\left\lceil \frac{k}{2} \right\rceil$. This algorithm can also be generalized to work with arbitrary degree requirements b_v, where each vertex v is to have a degree of b_v. This results in Algorithm 4, which has an approximation ratio of $(2k+1)$ and requires that $b_v \geq 2\left\lceil \frac{k}{2} \right\rceil$ for all vertices v.

The second algorithm is Algorithm 5, which gives a $(5+\frac{2}{k})$ approximation ratio, but requires the restriction that $d \geq 2k-1$. The same type of algorithm can also be used for creating a low weight k-vertex-connected d-regular subgraph. This results in Algorithm 6 which is an $(5+\frac{2(k-1)}{n}+\frac{2}{k})$-approximation algorithm for the minimum weight k-vertex-connected d-regular graph problem. This algorithm also requires the restriction that $d \geq 2k-1$.

4.2 Future work

Algorithm 3 and Algorithm 4 require that every vertex needs to have a degree of at least $2\left\lceil \frac{k}{2} \right\rceil$. This condition only has an effect when k is odd and we require that some vertices have a degree of exactly k. It might be possible to extend this algorithm such that it will work without this condition. This

could for instance be useful in real world problems where one wants to create a connected graph and some vertices are to have a degree of one. For this case one can use algorithm Algorithm 4, but there will be some connected components that do not contain a 2-edge-connected components, which the current algorithm requires for its proof of correctness. One can still apply the algorithm on the components that do have 2-edge-connected components, but after the algorithm ends there will still be some components $T_1, ..., T_c$ (which will be trees) that are not connected to the rest of the graph R. If the original degree specification b_v is feasible, it is possible to connect these trees to the rest of the graph. One option to do so would be to remove an edge contained in a cycle of R and an edge contained in some tree T_i and to subsequently connect the incident vertices of the tree, with those of the cycle. Doing this for every tree T_i would create a connected graph with the degree specification b_v. It is still uncertain if such an approach can be proven to have a solution that is at most a constant factor larger than that of the optimal solution.

Additionally it may be possible to improve the approximation ratio of Algorithm 3 and Algorithm 4 by transforming a i-edge-connected graph immediately into a $(i + 2)$-edge-connected graph. An indication that this may be possible is the paper of Cornelissen et al. [5] which uses a similar approach to transform a 0-edge-connected graph into a 2-edge-connected graph at the same cost Algorithm 4 requires to transform an i-edge-connected graph into a $(i + 1)$-edge-connected graph.

Bibliography

[1] Markus Bläser, *A new approximation algorithm for the asymmetric tsp with triangle inequality*, ACM Transactions on Algorithms **4** (2008), no. 4.

[2] Markus Bläser, Bodo Manthey, and Jiri Sgall, *An improved approximation algorithm for the asymmetric TSP with strengthened triangle inequality*, J. Discrete Algorithms **4** (2006), no. 4, 623–632.

[3] Yuk Hei Chan, Wai Shing Fung, Lap Chi Lau, and Chun Kong Yung, *Degree bounded network design with metric costs*, SIAM J. Comput. **40** (2011), no. 4, 953–980.

[4] N. Christofides, *Worst-case analysis of a new heuristic for the traveling salesman problem*, (1976).

[5] Kamiel Cornelissen, Ruben Hoeksma, Bodo Manthey, N. S. Narayanaswamy, and C. S. Rahul, *Approximability of connected factors*, WAOA, 2013, pp. 120–131.

[6] Takuro Fukunaga and Hiroshi Nagamochi, *Network design with edge-connectivity and degree constraints*, Theory Comput. Syst. **45** (2009), no. 3, 512–532.

[7] Martin Grötschel and Clyde L. Monma, *Integer polyhedra arising from certain network design problems with connectivity constraints*, SIAM J. Discret. Math. **3** (1990), no. 4, 502–523.

[8] Kamal Jain, *A factor 2 approximation algorithm for the generalized steiner network problem*, Combinatorica **21** (2001), no. 1, 39–60.

[9] Vladimir Kolmogorov, *Blossom V: a new implementation of a minimum cost perfect matching algorithm.*, Math. Program. Comput. **1** (2009), no. 1, 43–67.

[10] Guy Kortsarz and Zeev Nutov, *Approximating node connectivity problems via set covers*, Algorithmica **37** (2003), no. 2, 75–92.

[11] Lap Chi Lau, Joseph Naor, Mohammad R. Salavatipour, and Mohit Singh, *Survivable network design with degree or order constraints*, SIAM J. Comput. **39** (2009), no. 3, 1062–1087.

[12] Lap Chi Lau and Mohit Singh, *Additive approximation for bounded degree survivable network design*, SIAM J. Comput. **42** (2013), no. 6, 2217–2242.

[13] Laszlo Lovasz and Michael D. Plummer, *Matching Theory*, American Mathematical Society, August 1986.

[14] Zeev Nutov, *Approximating multiroot 3-outconnected subgraphs*, Networks **36** (2000), no. 3, 172–179.

[15] M. Resende and P. (eds.) Pardalos, *Handbook of Optimization in Telecommunications*, 1 ed., Springer, March 2006.

[16] Alexander Schrijver, *Combinatorial Optimization : Polyhedra and Efficiency*, Algorithms and Combinatorics, Springer, July 2004.

[17] Vijay V. Vazirani, *Approximation algorithms*, Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[18] Leonardo Zambito, *The traveling salesman problem: A comprehensive survey*, (2006), `http://www.cse.yorku.ca/~aaw/Zambito/TSP_Survey.pdf`.