University of Twente

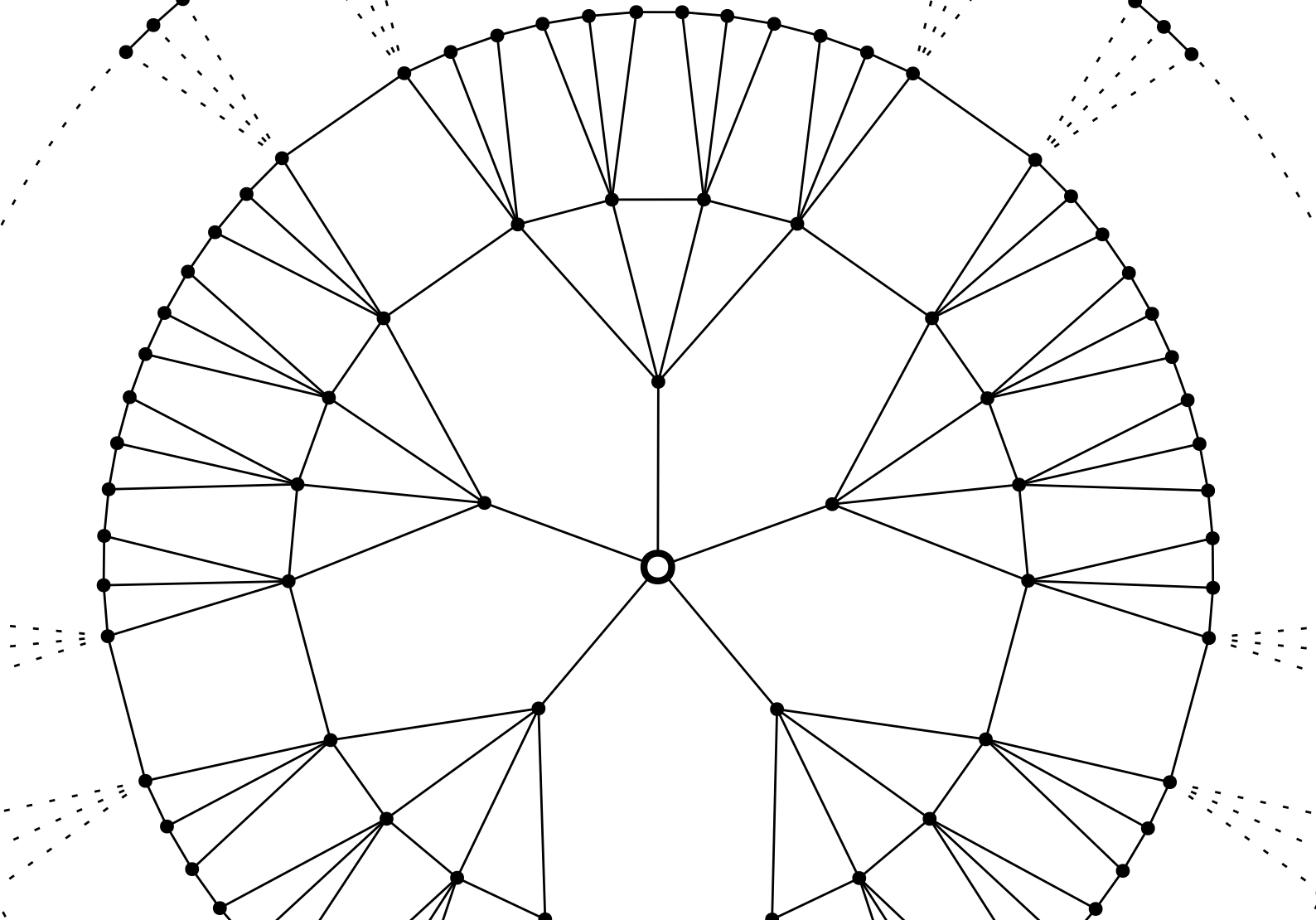
Master Thesis

Probabilistic Properties of Highly Connected Random Geometric Graphs

Victor M.J.J. Reijnders

Supervisor:
Dr. Bodo Manthey

August 18, 2016



Contents

Abstract

In this thesis we investigate how to obtain cheap reliable networks, and how the problem of finding them behaves. We study reliability in terms of k-edge-connectivity and k-vertex connectivity in graphs. In these graphs, we either charge per edge or we assign power to vertices and charge per vertex. When charging per vertex, the network can be seen as a wireless network and connections have to be symmetric there. The goal is to minimise the sum of edge lengths, or for wireless networks, the total power assignment. We define five problems such that they are Euclidean functionals.

We show some properties of these functionals, also using their boundary functionals, for arbitrary k, dimension d, and raising edge lengths to the pth power. With these properties we show complete convergence, as well as a bound on the longest edge in an optimal k-edge-connected graph with high probability. We show bounds on the rates of convergence of means for all our functionals. For the functionals associated with k-edge-connectedness we present two umbrella theorems as extensions on a limit theorem.

One of the reasons to prove our functionals have these properties is to use partitioning algorithms that rapidly compute near-optimal solutions on typical examples. To explain this performance, we apply smoothed analysis to obtain a smoothed approximation ratio.

Mixed integer linear programs are presented for all functionals, which we use for computing optimal solutions. Our computational results show the performance of the partitioning algorithms when we use various numbers of partitions for k-edge-connected graphs and power assignment graphs. These results show that solution times can be decreased greatly while only increasing the solution values slightly if the number of partitons is chosen in the right way.

As far as we are aware, k-edge-connectivity and k-vertex-connectivity have not been studied yet as functionals, and no partitioning algorithms have been presented for them.

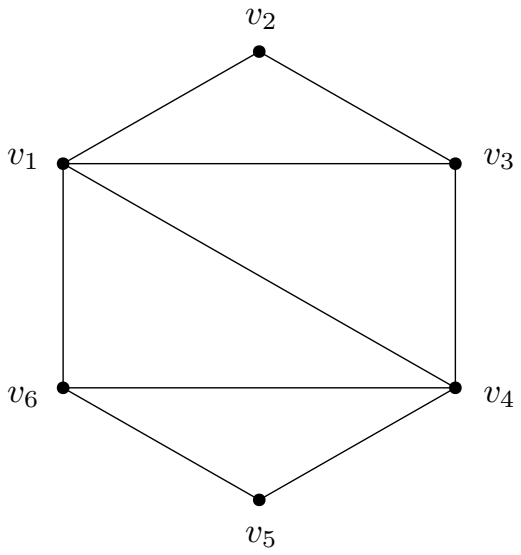
1 Introduction

The design of fault tolerant networks is an important issue in todays research, due to their numerous applications. The goal is to find cheap and reliable networks with some specific characteristics. Reliability is generally expressed in terms of the connectivity of a network. Different problems all ask for their own type of connectivity. For example, we might want to have multiple paths between each pair of nodes to account for possible failures in a link or even a failing node. Applications for these type of problems can be found in the design of reliable communication and transportation networks [3, 16, 17].

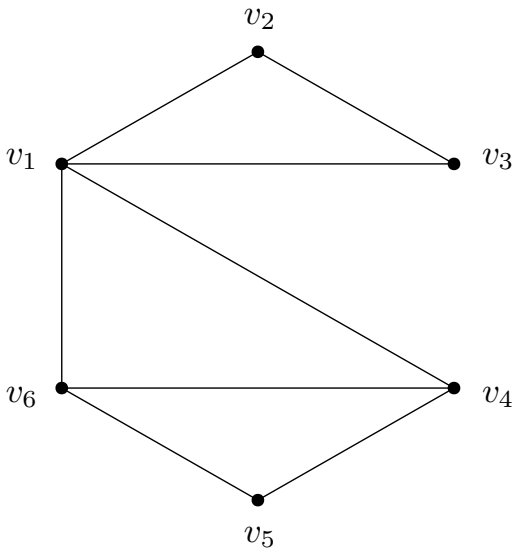
Wireless ad hoc networks have also received significant attention in recent studies [7, 12, 29]. Instead of direct connections between nodes, communication takes place through single-hop transmissions or by relaying through intermediate nodes. Here we assign a transmission power to each node. As the transmission range is directly related to the power used by a node, the goal is to find a fault tolerant network with minimal total power usage. Possible applications that are being considered by researches are environmental monitoring, emergency disaster relief where wiring is difficult, communication between mobile computers for conferencing and home networking, wireless sensor networks, multi-hop extensions of cellular telecommunication systems, and networks of vehicles [4, 12]. Metricom Inc's Ricochet network and the Army Near-Term Digital Radio network are examples of fully operational multi-hop wireless networks [29].

For both wired networks and wireless networks we study the problem of finding a cheapest k-edge-connected network, as well as a cheapest k-vertex-connected network. We also refer to wired networks as regular networks. The requirement for a network to be k-edge-connected is that the network is still connected when at most $k-1$ edges fail. Similarly, a network is k-vertex-connected if it is still connected when at most $k-1$ nodes fail. To illustrate these concepts, a spanning tree is 1-edge-connected and 1-vertex-connected, and a cycle is 2-edge-connected and 2-vertex-connected. In figure 1 we can see some other examples of k-edge-connected and k-vertex-connected graphs. In this thesis we want to find these k-edge-connected or k-vertex-connected

graphs of minimum costs, where costs are defined as the sum of all edge lengths for regular networks, and as the sum of the assigned power for wireless networks. We define five problems such that they are Euclidean functionals; Finding a minimal k-edge-connected graph, finding a minimal k-edge-connected multigraph, finding a minimal k-edge-connected power assignment graph, finding a minimal k-vertex-connected graph, and finding a minimal k-vertex-connected power assignment graph.



(a) *This graphs is 2-edge-connected, as removing edges (v_1, v_2) and (v_2, v_3) would leave a disconnected graph. Removing less edges does not disconnect the graph. It is 2-vertex-connected as well, since removing v_1 and v_4 would disconnect the graph. The graph cannot be disconnected by removing less than 2 vertices.*



(b) *After removing (v_3, v_4) the graphs is still 2-edge-connected, as removing only one edge will not disconnect the graph. It is only 1-vertex-connected though, as removing v_1 would disconnect the graph.*

Figure 1: *Example graphs showing the difference between k-edge-connectedness and k-vertex-connectedness.*

Finding a cheapest k-edge-connected, or k-vertex-connected network is NP-hard [15], and so is finding a minimal power wireless network [10]. As we still want to have reasonably good solutions in acceptable computation time, we need to find a good approximation algorithm. Partitioning algorithms have shown a lot of potential with similar problems [6]. In practice, partitioning algorithms are very fast. However, in the worst case these algorithms give solutions far from the optimum. To explain this performance, we use smoothed analysis [33]. Partitioning algorithms divide the whole problems into smaller cells and compute optimal solutions on these. Then these solutions are joined to obtain a solution for the whole problem. More detail can be found in Section 4.8.

Smoothed analysis is a hybrid of worst-case and average-case analysis. Only looking at worst-case performance often gives an analysis that is too pessimistic. Average-case analysis often exploits very specific properties that occur with overwhelming probablity, but it does not mean typical instances share these properties. That is why we use smoothed analysis. An adversary specifies an instance, and this instance is then slightly randomly perturbed. This often better explains the performance of algorithms convincingly. We refer to two recent surveys for a broader picture [24, 34].

We prove several properties for these problems needed for smoothed analysis. With these properties we are also able to prove several limit theorems as well as convergence theorems. We use smoothed analysis on a partitioning algorithm and implement this algorithm. Simulations are

done on randomly generated graphs with the partitioning algorithm and these are compared to optimal solutions acquired by solving linear programs. The simulations show promising results in terms of solving time.

The rest of this thesis is organised as follows. In Section 2 we give all definitions used in the rest of this thesis. We summarise related work in Section 3. The theoretical results including their proofs are all in Section 4, followed by the linear programs of the problems in Section 5. In Section 6 we present a simulation study of the partitioning algorithm. Following that, we present our conclusions in Section 7, and finish with some recommendations for future work in Section 8.

2 Problem Description

In this section we will define all important concepts used in the rest of this thesis. Although many are defined identically in similar papers, there are some slight differences between different fields. We define all these concepts in this thesis to make the thesis self-contained and to avoid ambiguities caused by slightly different definitions used in the literature.

2.1 Functionals

All graphs in this paper are undirected, and we will both be considering simple graphs as well as multigraphs. Let $G = (V, E)$ be a graph. We assume $V \subset \mathbb{R}^d$, where d is a constant and V is finite. In the rest of the paper, $n = |V|$ is the number of vertices. In Section 1 we already mentioned edge- (vertex-)disjoint paths. We will now define these formally.

Definition 2.1 (Edge/Vertex-disjoint paths)**.** *Let $G = (V, E)$ be a graph. Consider two paths $C_1 = (u_1, \ldots, u_n)$ and $C_2 = (v_1, \ldots, v_m)$ with $u_i, v_j \in V$ and $(u_{i-1}, u_i), (v_{i-1}, v_i) \in E$. Then C_1 and C_2 are edge-disjoint, if they share no edge. The paths are vertex-disjoint, if they share no common vertex (start and end vertices excluded).*

We can use this definition to formally state if a graph is k-edge-connected by checking how many edge-disjoint paths it has between each pair of vertices.

Definition 2.2 (k-edge-connected)**.** *Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. G is k-edge-connected if the following holds:*

- *If $|V| \geq k + 1$ then there exist at least k edge-disjoint paths in E between u and v for all pairs of vertices $u, v \in V$.*
- *If $|V| \leq k$ then G is complete.*

We also present an alternative definition for k-edge-connectedness based on edge cuts instead of edge-disjoint paths. These definitions are equivalent.

Definition 2.3 (k-edge-connected (alternative))**.** *Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. Then G is k-edge-connected for $|V| \geq k + 1$ if for any edge set $C \subseteq E$ with $|C| \leq k - 1$, $(V, E \setminus C)$ is connected, and for $|V| \leq k$ if G is complete.*

We want to present these definitions for multigraphs as well. A multigraph in this paper can have multiple edges (u, v) between each pair of vertices, but no loops. N_{uv} indicates how often the edge (u, v) appears in the graph. For multigraphs the definitions are very similar, though the difference is that a edge (u, v) can appear in at most N_{uv} edge-disjoint paths.

Definition 2.4 (k-edge-connected (multigraph))**.** *Let $G = (V, E)$ be a multigraph and $k \in \mathbb{N}$. Then G is k-edge-connected if there exist at least k edge-disjoint paths in E between u and v for all pairs of vertices $u, v \in V$.*

In the same way as the alternative definition of k-edge-connectedness for simple graphs, we can also give this alternative definition for multigraphs.

Definition 2.5 (k-edge-connected (multigraph, alternative))**.** *Let $G = (V, E)$ be a multigraph and $k \in \mathbb{N}$. Then G is k-edge-connected if for any edge set $C \subseteq E$ with $|C| \leq k - 1$, $(V, E \setminus C)$ is connected.*

Definition 2.2 and 2.3 are equivalent, as well as definition 2.4 and 2.5. This follows from Menger's Theorem [26]. The intuition behind this is that when we have k-edge-disjoint paths from u to v, we can remove one edge from every path. This will disconnect the graph, while removing one from only $k-1$ paths will still leave one path from u to v. Menger's Theorem is also closely related to the Max-Flow Min-Cut Theorem, which is actually a generalisation of Menger's Theorem. Sometimes we will also need that only parts of a graph have k edge-disjoint paths between them. These parts are called locally k-edge-connected, as described in following definition.

Definition 2.6 (Locally k-edge-connected)**.** *Let $G = (V, E)$ be a graph, $u, v \in V$, and $k \in \mathbb{N}$. Then u and v are locally k-edge-connected in G if there exist at least k edge-disjoint paths in E between u and v. For $U \subset V$, we say that U is locally k-edge-connected in G if all pairs of $u, v \in U$ are locally k-edge-connected.*

It follows from Definition 2.6 that if V is locally k-edge-connected in a graph $G = (V, E)$, then G is k-edge-connected. This is in accordance with Definition 2.2. For k-vertex-connectedness we can use similar definitions, except that we need vertex-disjoint paths. Each pair of vertex-disjoint paths is automatically edge-disjoint paths, as having no common vertices means having no common edges. This shows that k-vertex-connectedness is a stronger requirement than k-vertex-connectedness.

Definition 2.7 (k-vertex-connected)**.** *Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. Then G is k-vertex-connected for $|V| \geq k + 1$ if there exist at least k vertex-disjoint paths in E between u and v for all pairs of vertices $u, v \in V$, and for $|V| \leq k$ if G is complete.*

Similar to the alternative definition we have for k-edge-connectedness, we can also define k-vertex-connectedness by looking at vertex cuts.

Definition 2.8 (k-vertex-connected (alternative))**.** *Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. Then G is k-vertex-connected if for any vertex set $K \subseteq V$ with $|K| \leq k - 1$, $(V \setminus K, E)$ is connected.*

Definition 2.7 and 2.8 are equivalent (by Menger's Theorem [26] and Whitney [36]). This can be made intuitive by the same type of reasoning as with k-edge-connectedness. Whitney presented this result based on Menger's Theorem. Definition 2.8 does not make the distinction of $|V| \leq k$ and $|V| \geq k + 1$ as it is already implied that G is complete if $|V| \leq k + 1$ (see Lemma 4.6). There is no use in defining k-vertex-connectedness on multigraphs, as we need vertex-disjoint paths and not edge-disjoint paths. If a graph is k-vertex-connected, it is automatically k-edge-connected. The converse does not hold. We also note that for $k = 1$ k-edge-connectedness will also be simply referred to as connectedness. Besides regular networks, we will also be looking at wireless networks. These are defined by assigning power to each vertex. Vertices are then connected to each other if their power if sufficiently high. The following definition makes this formal.

Definition 2.9 (Power assignment)**.** *A power assignment* PA *assigns a real, positive value to all vertices $v \in V$. The corresponding power assignment graph then contains all edges (u, v) for which $\mathrm{PA}(u), \mathrm{PA}(v) \geq |(u, v)|$, where $|(u, v)|$ denotes the Euclidean distance between u and v.*

Figure 2 shows an example of such a power assignment graph. Throughout this thesis we are searching for minimal k-edge-connected and k-vertex-connected (power assignment) graphs. For regular graphs, minimality is defined in terms of summed edge length. For power assignment graphs minimality is defined in terms of total power. For each set of points we can get the value of the minimal k-edge-connected and k-vertex-connected (power assignment) graph. Such a mapping can be captured in a functional.

Definition 2.10 (Functional)**.** *An function L is called a functional if it maps a set of points $V \subset W$ to a real value in $\mathbb{R}$, where W can be a space of functions. When the function L is linear,*

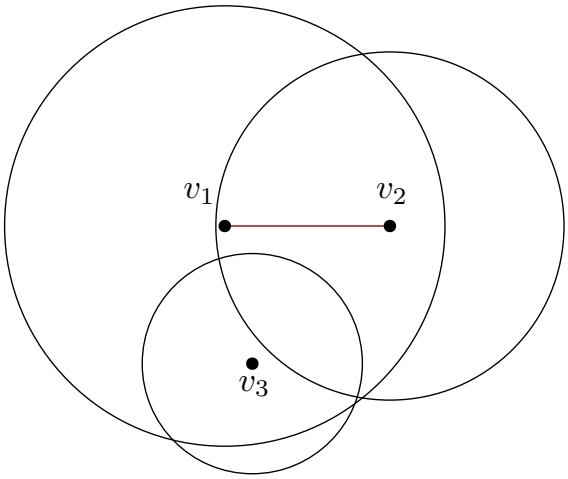


Figure 2: *Example of a power assignment graph with the circles indicating the transmitting power of each vertex. Only v_1 and v_2 reach each other, so only edge (v_1, v_2) is present in the graph.*

it is called a linear functional. We will write $L(V, R)$ for $L(V \cup R)$, and we can view L as a function defined on pairs (V, R) where V is a finite set and $R \in \mathcal{R}$ is a d-dimensional rectangle. Here $\mathcal{R} = \mathcal{R}(d)$ is the collection of d-dimensional rectangles. When L depends on a power p, we will write L^p instead of L.

Functionals can have some nice properties, for example being Euclidean. For the following definition we will assume $V = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$, $d > 1$.

Definition 2.11 (Euclidean functional,[35])**.** *A functional L is called a Euclidean Functional if the following properties hold:*

1. *Scaling, i.e. $L(\{\alpha x_1, \ldots, \alpha x_n\}, \alpha R) = \alpha L(\{x_1, \ldots, x_n\}, R)$ for all real $\alpha > 0$.*
2. *Translation Invariance, i.e. $L(\{x_1 + x, \ldots, x_n + x\}, R) = L(\{x_1, \ldots, x_n\}, R)$ for all $x \in \mathbb{R}^d$.*

The following are a few examples of Euclidean functionals:

- Mapping a finite point set V to the length of a shortest Hamiltonian cycle on V, i.e. the length of the optimal travelling salesman tour.
- Mapping a finite point set V to the length of a minimum-length perfect matching (leaving out one vertex if n is odd) on V.
- Mapping a finite point set V to the length of a minimum-length spanning tree on V.

We can now also define minimal k-edge-connected graphs in terms of summed edge length such that it is a functional. In this definition, we weight the Euclidean length of edges by taking it to the power of p. We will also refer to this as pth power-weighted edges.

Definition 2.12 (MkEE^p)**.** *Let $d \in \mathbb{N}$ be arbitrary and let $p > 0$. Then $\mathrm{MkEE}^p(V, R)$ is the length of a minimal k-edge-connected graph in terms of summed edge lenghts on V in d-dimensional rectangle R with pth power-weighted edges. Thus*

$$\mathrm{MkEE}^p(V, R) = \min_{X \in S} \sum_{e \in X} |e|^p, \tag{1}$$

where S is the set of k-edge-connected simple graphs on V and $|e|$ denotes the Euclidean length of an edge e.

The name MkEE stems from Minimal k-Edge-connected graph with Edge costs. We are also treating the same problem on multigraphs. For this functional we add an m to the name to distinguish it from the version restricted to simple graphs. In this way we get MkEEm. The main reason we also consider multigraphs is that we initially had trouble proving all useful properties for MkEE^p, and that using multigraphs would simplify these issues. Luckily, we managed to generalise all results for both simple and multigraphs. For completeness however, we decided to still include all results obtained for multigraphs. The definition for multigraphs is similar to Definition 2.12.

Definition 2.13 (MkEEm^p)**.** *For all $0 < p < d$, let $\text{MkEEm}^p(V, R)$ be the length of a minimal k-edge-connected multigraph in terms of summed edge lengths on V in d-dimensional rectangle R with pth power weighted edges. Thus*

$$\text{MkEEm}^p(V, R) = \min_{X \in S} \sum_{e \in X} |e|^p, \tag{2}$$

where S is the set of k-edge-connected multigraphs on V and $|e|$ denotes the Euclidean length of an edge e.

For power assignment graphs the objective changes a bit, as we want to minimise the total power usage. For this reason the name of the functional is based on Minimal k-Edge-connected Power assignment graph, MkEP, as each vertex has its own power.

Definition 2.14 (MkEP^p)**.** *For all $0 < p < d$, let $\text{MkEP}^p(V, R)$ be the value of a minimal k-edge-connected power assignment graph in terms of summed power assignment on V in d-dimensional rectangle R with pth power-weighted edges. Thus*

$$\text{MkEP}^p(V, R) = \min_{\text{PA} \in S} \sum_{v \in V} \text{PA}(v)^p, \tag{3}$$

where S is the set of power assignments with k-edge-connected power assignment graphs on V and $\text{PA}(v)$ denotes the power assigned to vertex v.

We do not consider a multigraph variant of the power assignment version, as it is unclear how to obtain a larger number of edges between two nodes in a meaningful way using power assignments. We would have to find a specific function assiging a number of edges between a pair of vertices u and v depending on $\text{PA}(u)$ and $\text{PA}(v)$ in that case. This however is inconsistent with the applications of wireless networks, and would give a rather arbitrary functional. For these reason, we decided not to include power assignment multigraphs in the thesis.

we can define functionals for finding the minimal k-vertex-connected graphs. The name then logically changes to Minimal k-Vertex-connected graph with Edge costs, or MkVE.

Definition 2.15 (MkVE^p)**.** *For all $0 < p < d$, let $\text{MkVE}^p(V, R)$ be length of a minimal k-vertex-connected graph in terms of summed edge lengths on V in d-dimensional rectangle R with pth power-weighted edges. Thus*

$$\text{MkVE}^p(V, R) = \min_{X \in S} \sum_{e \in X} |e|^p, \tag{4}$$

where S is the set of k-vertex-connected simple graphs on V and $|e|$ denotes the Euclidean length of an edge e.

The last functional to define is for finding the k-vertex-connected power assignment graphs, and the name logically becomes MkVP.

Definition 2.16 (MkVP^p)**.** *For all $0 < p < d$, let $\text{MkVP}^p(V, R)$ be the value of a minimal k-vertex-connected power assignment graph in terms of summed power assignment on V in d-dimensional rectangle R with pth power-weighted edges. Thus*

$$\text{MkVP}^p(V, R) = \min_{\text{PA} \in S} \sum_{v \in V} \text{PA}(v)^p, \tag{5}$$

where S is the set of power assignments with k-vertex-connected power assignment graphs on V and $\mathrm{PA}(v)$ denotes the power assigned to vertex v.

As mentioned before, there is no use in defining k-vertex-connectedness on multigraphs, as we need vertex-disjoint paths and not edge-disjoint paths.

A lot of functionals in combinatorial optimisation have a nice structure, but do not possess all properties we want. By defining new functionals we can get functionals with the desired properties without deviating too much from the original functional. We will call these modified functionals boundary functionals, an idea articulated in Redmond's thesis [30]. Roughly speaking, in these functionals, the entire boundary of the rectangle is considered as one additional vertex that can be used. These functionals will be used for proving certain properties Section 4. First we will give the formal description of boundary functionals.

Definition 2.17 (Boundary functional)**.** *Let L^p be a functional with $p > 0$. We define its boundary functional L^p_B on pairs (V, R), where V is a finite set and $R \in \mathcal{R}$ is a d-dimensional rectangle. $L^p_B(V, R)$ treats the boundary of R as a single point so that all edges joined to the boundary are joined to one another. The boundary of R will also be refered to as ∂R.*

To distinguish between a functional and its boundary functional, we will refer to the functional as the original functional. To make sure solutions from the boundary functional are close enough to those of the original functional, we need a slightly different definition of k-edge-connectedness on boundary graphs.

Definition 2.18 (k-edge-connected boundary graph)**.** *Let $G = (V \cup \partial R, E)$ be a simple (power assignment) boundary graph and $k \in \mathbb{N}$. For determining if G is k-edge-connected, edges to the boundary count as up to k independent edges. For $|V| \leq k$, we allow both complete graphs, as well as graphs where all vertices have an edge to the boundary.*

In the other boundary functionals we will use the same definitions as for the original functionals. We can now define boundary functionals for each of the original functionals considered in this thesis.

Definition 2.19 (MkEE^p_B)**.** *For all $0 < p < d$, let $\mathrm{MkEE}^p_B(V, R)$ be the length of a minimal k-edge-connected boundary graph in terms of summed edge lengths on $V \cup \partial R$ in d-dimensional rectangle R with pth power-weighted edges. Thus*

$$\mathrm{MkEE}^p_B(V, R) = \min_{X \in S} \sum_{e \in X} |e|^p, \tag{6}$$

where S is the set of k-edge-connected boundary simple graphs on V where ∂R can be used, and $|e|$ denotes the Euclidean length of an edge e.

Definition 2.20 (MkEEm^p_B)**.** *For all $0 < p < d$, let $\mathrm{MkEEm}^p_B(V, R)$ be the length of a minimal k-edge-connected boundary multigraph in terms of summed edge lengths on $V \cup \partial R$s in d-dimensional rectangle R with pth power-weighted edge. Each vertex can have multiple edges to the boundary. Thus*

$$\mathrm{MkEEm}^p_B(V, R) = \min_{X \in S} \sum_{e \in X} |e|^p, \tag{7}$$

where S is the set of k-edge-connected boundary multigraphs on V where ∂R can be used, and $|e|$ denotes the Euclidean length of an edge e.

Definition 2.21 (MkEP^p_B)**.** *For all $0 < p < d$, let $\mathrm{MkEP}^p_B(V, R)$ be the value of a minimal k-edge-connected power assignment boundary graph in terms of summed power assignment on $V \cup \partial R$ in d-dimensional rectangle R with pth power-weighted edges. Here an edge to the boundary from a vertex v is present if $\mathrm{PA}(v) \geq \min_{w \in \partial R} |(u, w)|$. Thus*

$$\mathrm{MkEP}^p_B(V, R) = \min_{\mathrm{PA} \in S} \sum_{v \in V} \mathrm{PA}(v)^p, \tag{8}$$

where S is the set of power assignments with k-edge-connected power assignment boundary graphs on V where ∂R can be used, and $\mathrm{PA}(v)$ denotes the power assigned to vertex v.

Definition 2.22 (MkVE_B^p)**.** *For all $0 < p < d$, let $\mathrm{MkVE}_B^p(V, R)$ be the length of a minimal k-vertex-connected boundary graph in terms of summed edge lengths on $V \cup \partial R$ in d-dimensional rectangle R with pth power-weighted edges. Thus*

$$\mathrm{MkVE}_B^p(V, R) = \min_{X \in S} \sum_{e \in X} |e|^p, \tag{9}$$

where S is the set of k-vertex-connected boundary simple graphs on V where ∂R can be used, and $|e|$ denotes the Euclidean length of an edge e.

Definition 2.23 (MkVP_B^p)**.** *For all $0 < p < d$, let $\mathrm{MkVP}_B^p(V, R)$ be the value of a minimal k-vertex-connected power assignment boundary graph in terms of summed power assignment on $V \cup \partial R$ in d-dimensional rectangle R with pth power weighted edges. Here an edge to the boundary from a vertex v is present if $\mathrm{PA}(v) \geq \min_{w \in \partial R} |(u, w)|$. Thus*

$$\mathrm{MkVP}_B^p(V, R) = \min_{\mathrm{PA} \in S} \sum_{v \in V} \mathrm{PA}(v)^p, \tag{10}$$

where S is the set of power assignments with k-vertex-connected power assignment boundary graphs on V where ∂R can be used, and $\mathrm{PA}(v)$ denotes the power assigned to vertex v.

2.2 Properties

We have defined all functionals considered in this thesis in order to be able to look at certain properties these functionals have. Many functionals are (approximately) subadditive. This structure expresses the self-similarity properties of the graph and is based on geometry in d dimensions [37]. There are several ways of expressing this structure. First we introduce simple subadditivity. Roughly speaking, this shows that the function value of a whole set is not larger than the sum of function values of the sets in a partition of this set (with some error term).

Definition 2.24 (Simple Subadditivity)**.** *Let L^p be a functional with $p > 0$. Then L^p is simple subadditive if for all finite sets U and V in $[0, t]^d$ we have*

$$L^p(U \cup V, [0, t]^d) \leq L^p(U, [0, t]^d) + L^p(V, [0, t]^d) + C_1 t^p \tag{11}$$

where $C_1 = C_1(d, p)$ is a constant possibly depending on d and p.

Simple subadditivity is often stronger than needed for proving theorems. So we introduce a form of approximate subadditivity, called geometric subadditivity. Here the partitions of the whole set are limited to rectangles.

Definition 2.25 (Geometric Subadditivity)**.** *Let L^p be a functional with $p > 0$. Then L^p is geometric subadditive if for all finite sets V, all rectangles $R \in \mathcal{R}$ and all partitionings of R into rectangles R_1 and R_2 we have*

$$L^p(F, R) \leq L^p(F, R_1) + L^p(F, R_2) + C_1(\operatorname{diam} R)^p, \tag{12}$$

where $C_1 = C_1(d, p)$ is a finite constant.

If a functional is simple subadditive, it implies it is also geometric subadditive. We can also look at a property that works the other way around, called superadditivity. Roughly speaking, this shows that the function value of a whole set is not lower.

Definition 2.26 (Superadditivity)**.** *Let L^p be a functional with $p > 0$. Then L^p is superadditive if for all finite sets V, all rectangles $R \in \mathcal{R}$ and all partitionings of R into rectangles R_1 and R_2 we have*

$$L^p(F, R) \geq L^p(F, R_1) + L^p(F, R_2). \tag{13}$$

Note that (13) does not include an error term. If a functional would be both subadditive and superadditive, it would become nearly additive in the sense that

$$L^p(F, R) \approx L^p(F, R_1) + L^p(F, R_2). \tag{14}$$

This would be ideal to show that the global graph length can be approximated by the sum of the lengths of local components. Unfortunately most subadditive functionals do not satisfy superadditivity relation (13). Fortunately, boundary functionals are often superadditive, and this is the reason why we have introduced them. As we cannot directly show near additivity, we want to show that the original functional and the boundary functional are not too far from each other. This is made exact in the following definition.

Definition 2.27 (Pointwise Closeness)**.** *Fix $1 \leq p < d$. Let L^p be a functional and L^p_B its boundary functional. Then L^p and L^p_B are pointwise close if for all finite sets V we have*

$$|L^p(V, [0,1]^d) - L^p_B(V, [0,1]^d)| = o(|V|^{(d-p)/d}). \tag{15}$$

Pointwise closeness is usually sufficient for most approximation purposes, but there is also a second, weaker way to measure closeness. This is called closeness in mean and is sufficient for finding asymptotics and rates of convergence of means.

Definition 2.28 (Closeness in Mean)**.** *Fix $1 \leq p < d$. Let L^p be a functional and L^p_B its boundary functional. Let U_i, $i \geq 1$ be independent uniformly distributed variables with values in $[0,1]^d$. Then L^p and L^p_B are close if*

$$|\mathbb{E}[L^p(\{U_1, \ldots, U_n\}, [0,1]^d)] - \mathbb{E}[L^p_B(\{U_1, \ldots, U_n\}, [0,1]^d)]| = o(n^{(d-p)/d}). \tag{16}$$

The last important property of functionals we will address in this thesis is smoothness. This describes how strong the variations of a functional are if vertices are added or deleted. Smooth functionals behave a lot more predictable and therefore it plays an important role in many limit theories.

Definition 2.29 (Smoothness)**.** *Fix $0 < p < d$. Let L^p be a (boundary) functional. Then L^p is smooth if for all finite sets U and V we have*

$$|L^p(U \cup V, [0,1]^d) - L^p(U, [0,1]^d)| = O(|V|^{(d-p)/d}). \tag{17}$$

Similar to closeness in mean, we can also define smoothness in mean. Though we will not use it in this thesis, we define it here for completeness in the same way as in Yukich [37].

Definition 2.30 (Smoothess in Mean)**.** *Fix $p > 0$ and $d \geq 2$. Let L^p be a functional. Let U_i, $i \geq 1$ be independent uniformly distributed variables with values in $[0,1]^d$. Then L^p is smooth in mean if there exists a constant $\gamma < 1/2$ such that for all $n \geq 1$ and $0 \leq k \leq n/‘2$ we have*

$$|\mathbb{E}[L^p(\{U_1, \ldots, U_n\}, [0,1]^d) - L^p(\{U_1, \ldots, U_{n\pm k}\}, [0,1]^d)]| = Ckn^{-p/d+\gamma}, \tag{18}$$

where $C = C(p, d)$ is a constant.

2.3 Convergence

There is also a definition we need to indicate the convergence of our functionals. This is known as complete convergence. This form of convergence indicates that the function values are nicely concentrated around its mean, without jumps that are too large.

Definition 2.31 (Complete Convergence)**.** *Let X_n, $n \geq 1$ be a sequence of random variables. Then X_n converges completely (c.c.) to a constant C if and only if for all $\epsilon > 0$ we have*

$$\sum_{n=1}^{\infty} P[|X_n - C| > \epsilon] < \infty \tag{19}$$

3 Related Work

There is more need for reliable communication and transportation networks as well as reliable wireless sensor and radio networks [3, 4, 12, 16]. As it was known that MkEE^p, MkEEm^p, MkEP^p, MkVE^p and MkVP^p are NP-hard [10, 15], a lot of research has been focused on finding good approximation algorithms. Most algorithms only consider the case when $k = 1$ or $k = 2$, though some research has been done on finding more reliable networks ($k \geq 2$). To our knowledge, using partitioning algorithms and smoothed analysis on these functionals has never been attempted.

3.1 Work related to MkEE^p, MkEEm^p and MkEP^p

Grötschel et al. [18] studied 2-edge-connectivity as well as 2-vertex-connectivity by using a cutting plane algorithm. They also looked at k-edge-connectivity and k-vertex-connectivity for larger k mainly by investigating the polyhedra arising from these network design problems [16, 17, 19].

Mahjoub [22] looked at 2-edge-connectivity by studying the polytope of all possible 2-edge-connected graphs. Mahjoub also concluded that the polytope is completely described by the trivial constraints and cut constraints when the graph is series-parallel (does not contain a homeomorph of K_4 as a subgraph). Biha and Mahjoub [5] looked at series-parallel graphs as well. In this paper their previous work is extended to k-edge-connectivity. Mahjoub [23] studied the polytope of all 2-vertex-connected graphs and derived some useful results on the extreme points of this polytope.

Diarrassouba et al. [13] look at 2-vertex-connectivity when a maximum path length was imposed on the vertex-disjoint paths between set pairs of terminals. They derived inequalities that could be used to define facets of the polyhedron of solutions. They also used a Branch-and-Cut algorithm to find good feasible solutions when the path lenght was at most 3 edges. In a similar study Diarrassouba et al. [14] studied k-vertex-connectivity without a maximum path length for $k \geq 3$, again finding facets and using a Branch-and-Cut algorithm. They are not the only ones using this algorithm. Bendali et al. [3] also used a Branch-and-Cut algorithm for k-edge-connectedness.

Chan et al. [8] studied k-edge-connectivity and k-vertex-connectivity given a maximum degree for all vertices. They derived constant factor approximation algorithms when the cost function satisfied the triangle inequality, always returning solutions with the smallest possible maximum degree.

Chopra [9] did research on the polyhedron arising from k-edge-connected multigraphs, finding several facets of this polyhedron.

Cornelissen et al. [11] and Manthey and Waanders [25] both studied k-edge-connectivity and k-vertex-connectivity, albeit with degree constraints. They used approximation algorithms to find d-regular spanning subgraphs with some connectivity requirements.

3.2 Work related to MkVE^p and MkVP^p

Ramanathan and Rosales-Hain [29] have studied the problem of minimising the maximum power used by any vertex in the graph. They did this for connectivity and for 2-vertex-connectivity. They used two adaptive heuristics for mobile networks and two centralised algorithms for static networks.

Montemanni and Gambardella [28] looked at algorithms to get optimal solutions for connectedness in power assignment graphs. They also use linear programs and derive valid inequalities for the polytope of solutions.

Althaus et al. [1] use several approximation algorithms for connectivity in power assignment graphs, as well as a Branch-and-Cut algorithms based on a linear program for this problem.

Santi et al. [32] studied the connectivity in power assignment graph in case every vertex has the same amount of power. They derived lower and upper bounds on this power to have a high probability for a connected graph in 1-, 2- and 3-dimensional spaces.

De Graaf and Manthey [12] use a probabilistic approach to analyse connectivity in power assignment graphs. They obtain properties for this problem very similar to the ones we prove in

Section 4. They also show the expected approximation ratio of the simple spanning tree heuristic is strictly less than 2.

Calinescu and Wan [7] analysed several approximation algorithms for 2-edge-connectedness and 2-vertex-connectedness, as well as k-edge-connectedness for $k \geq 2$. They did this for both undirected graphs, as well as directed graphs (in which case arc $\{u, v\}$ in the power assignment graph if $\mathrm{PA}(u) \geq |\{u, v\}|$).

Lloyd et al. [21] describe a general approach for polynomial time algorithms for minimising the maximum power used by any vertex in a graph with monotone properties (k-edge-connectivity and k-vertex-connectivity are example of this). For minimising the total power, they prove the problem is NP-complete even for simple graph properties. They also use a approximation algorithm for 2-vertex-connectedness in power assignment graphs.

Bahramgiri et al. [2] designed an algorithm for k-vertex-connectedness in power assignment graphs where vertices in 2- and 3-dimensional spaces adjust their power by looking at nearby vertices.

Bettstetter [4] studied k-vertex-connected power assignment graphs when each vertex gets assigned the same power. He derived some bounds on this power to get an almost sure k-vertex-connected graph. In case transmission power is given, an algorithm is presented to calculate how many vertices are needed to cover an area with a k-vertex-connected graph.

Li and Hou [20] studied a problem closely related to k-vertex-connectivity in power assignment graphs. They look at an already k-vertex-connected graph and use their algorithm to reduce the maximum power used by a vertex while maintaining k-vertex-connectivity.

4 Theoretical Results

In this section we will treat the theoretical results we have obtained along with their proofs. For completeness, we have also included a few obvious results and counterexamples to hypotheses we had. After proving subaddivity, superaddivity, pointwise closeness and smoothness, we can use these properties. We show a bound on the longest edge with high probability for MkEE^p. We also use the properties to show complete convergence and some limit theories as wel as bounds on the rates of convergence of means. Finally we present two umbrella theorems for MkEE^p, MkEEm^p, and MkEP^p as extensions on the limit theorem. After this, some results related to the smoothed analysis of the partitioning algorithm are presented.

We have structured this section in the following way. We will start with a few simple results used throughout the rest of this section. After this, we will treat results related to subadditivity, followed by results related to superadditivity. Section 4.5 contains results related to pointwise closeness. In Section 4.6 we will prove results related to smoothness. After this some probabilistic results as well as limit theorems are presented and this section is concluded by the smoothed analysis of MkEE^p and MkEP^p. Throughout this section d, p and k are constants.

4.1 Basic Results

In Section 2 we gave a few examples of Euclidean functionals. The functionals we will treat in this thesis are also Euclidean.

Theorem 4.1. *For $p \geq 1$ and $k \in \mathbb{N}$, MkEE^p, MkEEm^p, MkEP^p, MkVE^p and MkVP^p are all Euclidean functionals.*

The proof of Theorem 4.1 is trivial, so we will omit it. We will however show a useful property of k-edge-connectedness that k-vertex-connectedness does not possess. Connecting two k-edge-connected parts creates a k-edge-connected graph. We call this transitivity. This property is made exact in Theorem 4.2.

Theorem 4.2. *k-edge-connectedness is transitive. This means that if graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are k-edge-connected and $V_1 \cap V_2 \neq \emptyset$, then the graph $H = (V_1 \cup V_2, E_1 \cup E_2)$ is k-edge-connected as well.*

Proof. For each pair of vertices $u, v \in V_1 \cup V_2$ we want to find k edge-disjoint paths between them. If both u and v are in either V_1 or V_2 these paths exists as both G_1 and G_2 are k-edge-connected. So w.l.o.g. $u \in V_1 \setminus V_2$ and $v \in V_2 \setminus V_1$. We consider a vertex $w \in V_1 \cap V_2$. We know that there exist k edge-disjoint paths P_i from u to w, and also k edge-disjoint paths Q_j from w to v with $1 \leq i, j \leq k$. We can combine these in the following way. We start with path P_i and follow it from u to w. As soon as we encounter a vertex t that is also in a path Q_j we stop, where $t = w$ is allowed. We can now create a path from u to v by taking P_i up to t and then completing it by adding Q_j from t to v. We then remove P_i and Q_j from the paths we have. As we have as many paths from u to w as from w to v, and as all of them have at least w in common, this process will terminate and we end up with k edge-disjoint paths from u to v. □

4.2 Subadditivity

We will prove that MkEE^p, MkEEm^p, MkEP^p, MkVE^p and MkVP^p all are geometric subadditive functionals. Although geometric subadditivity is weaker than simple subadditivity, most theorems described later in this section only require geometric subadditivity. On top of that geometric subadditivity is easier to prove. We will start with MkEE^p.

Theorem 4.3. *For $p \geq 1$, MkEE^p is a geometric subadditive functional.*

Proof. We will check that Definition 2.25 holds for $\mathrm{MkEE}^p(V, R)$.The partition into R_1 and R_2 results in two graphs which we will call (V_1, E_1) and (V_2, E_2). Here $V \cap R_1 = V_1$ and $V \cap R_2 = V_2$. We distinguish three cases. Here w.l.o.g. $|V_1| \geq |V_2|$.

1. $|V_1|, |V_2| \geq k + 1$. We have that (V_1, E_1) and (V_2, E_2) are k-edge-connected. By joining (V_1, E_1) and (V_2, E_2), we want to obtain a k-edge-connected graph on V. This means we need to add some edges. We add k vertex-disjoint edges $e_1, \ldots, e_k$, with $e_i = (u_i, v_i)$ and $u_i \in V_1$, $v_i \in V_2$. As (V_1, E_1) and (V_2, E_2) are k-edge-connected, removing less than k edges from any of them results in a connected graph. The only option then is to disconnect (V_1, E_1) and (V_2, E_2), but as there are k edges in between them, removing $k-1$ or fewer of them still results in a connected graph. So in total $(V, E_1 \cup E_2 \cup \{e_1, \ldots, e_k\})$ is a k-edge-connected graph and we can state that

$$\begin{aligned} \mathrm{MkEE}^p(V, R) &\leq \mathrm{MkEE}^p(V, R_1) + \mathrm{MkEE}^p(V, R_2) + \sum_{i=1}^{k} |e_i|^p && (20) \\ &\leq \mathrm{MkEE}^p(V, R_1) + \mathrm{MkEE}^p(V, R_2) + k d^{p/2} (\operatorname{diam} R)^p && (21) \end{aligned}$$

2. $|V_1| \geq k + 1$, $|V_2| \leq k$. We know that (V_2, E_2) is complete. For each vertex $v_i \in V_2$ we add k edges $e_{i_1}, \ldots e_{i_k}$ with endpoints in V_1. Each pair of vertices from V_1 has at least k edge-disjoint paths in E_1, and by adding e_{i_j} each pair in V has k edge-disjoint paths as well. At most k^2 edges have been added this way, so

$$\begin{aligned} \mathrm{MkEE}^p(V, R) &\leq \mathrm{MkEE}^p(V, R_1) + \mathrm{MkEE}^p(V, R_2) + \sum_{i=1}^{|V_2|} \sum_{j=1}^{k} |e_{i_j}|^p && (22) \\ &\leq \mathrm{MkEE}^p(F, R_1) + \mathrm{MkEE}^p(F, R_2) + k^2 d^{p/2} (\operatorname{diam} R)^p && (23) \end{aligned}$$

3. $|V_1|, |V_2| \leq k$. Both (V_1, E_1) and (V_2, E_2) are complete, and we know that a complete graph is always k-edge-connected. So if we make the combined graph complete as well, it is k-edge-connected. We need to add at most k^2 edges this way, so $\mathrm{MkEE}^p(V, R)$ will be bounded by a term similar to that in (22).

Since the case distinction is exhaustive, MkEE^p is subadditive. □

As we never explicitly used that G was a simple graph in the proof of Theorem 4.3, we can easily copy this result for MkEEm^p.

Theorem 4.4. *For $p \geq 1$, MkEEm^p is a geometric subadditive functional.*

Proof. The proof is similar to that of Theorem 4.3. □

The proof of subadditivity of MkEP^p is similar, though we need to account for the change of power while adding edges.

Theorem 4.5. *For $p \geq 1$, MkEP^p is a geometric subadditive functional.*

Proof. We follow the proof of Theorem 4.3. We need to increase $\mathrm{PA}(v)$ accordingly for the edges that need to be added in the power assignment graph. We define $d_{uv} = \max\{0, |(u,v)| - \mathrm{PA}(u)\}$. This denotes the increase in power needed for u to reach v. Then we can add an edge (u,v) by a change of power equal to $(\mathrm{PA}(u) + d_{uv})^p + (\mathrm{PA}(v) + d_{vu})^p - \mathrm{PA}(u)^p - \mathrm{PA}(v)^p \leq 2(\operatorname{diam} R)^p$. The rest of the proof follows. □

For proving subadditivity of MkVE^p and MkVP^p we first need some other results.

Lemma 4.6. *Given $k \in \mathbb{N}$, consider a graph $G = (V, E)$ with $|E| \leq k+1$. The G is k-vertex-connected if and only if it is complete.*

Proof. Suppose graph G is not complete and let (u,v) be an edge that is not in E. Now consider the cut $X = V \setminus \{u, v\}$ and notice that $|X| \leq k-1$. This cut will disconnect vertices u and v so G cannot be k-edge-connected. Conversely, let G be a complete graph and consider any cut X with $|X| \leq k-1$. Then, as G is complete, $(V \setminus X, E)$ will be connected as the remaining vertices still have edges in between them, or it will leave the null graph, which is connected by definition. □

Lemma 4.7. *Consider a complete graph $G = (V, E)$. Then G is k-vertex-connected for all $k \in \mathbb{N}$.*

Proof. See proof Lemma 4.6 for the case where $|E| \leq k+1$. When $|E| > k+1$, any cut $X \subseteq V$ with $|X| \leq k-1$ will leave $(V \setminus X, E)$ connected as the remaining vertices still have edges in between them. □

With these lemmas, we can use a case distinction similar to the one in the proof of Theorem 4.3 to prove subadditivity for MkVE^p.

Theorem 4.8. *For $p \geq 1$, MkVE^p is a geometric subadditive functional.*

Proof. We will check that Definition 2.25 holds for $\mathrm{MkVE}^p(V, R)$. Note that the partition into R_1 and R_2 results in two graphs which we will call (V_1, E_1) and (V_2, E_2). We distinguish between three cases. Here w.l.o.g. $|V_1| \geq |V_2|$.

1. $|V_1|, |V_2| \geq k+1$. We have that (V_1, E_1) and (V_2, E_2) are k-vertex-connected. By joining (V_1, E_1) and (V_2, E_2), we want to obtain a k-vertex-connected graph on V. This means we need to add some edges. We add k vertex-disjoint edges $e_1, \ldots, e_k$, with $e_i = (u_i, v_i)$ and $u_i \in V_1$, $v_i \in V_2$. As (V_1, E_1) and (V_2, E_2) are k-vertex-connected, removing less than k vertices from any of them results in a connected graph. The only option then is to disconnect (V_1, E_1) and (V_2, E_2), but as there are k vertex-disjoint edges in between them, removing $k-1$ or fewer vertices still results in a connected graph. So in total $(V, E_1 \cup E_2 \cup \{e_1, \ldots, e_k\})$ is a k-vertex-connected graph and we can state that

$$\begin{aligned} \mathrm{MkVE}^p(V,R) &\leq \mathrm{MkVE}^p(V,R_1) + \mathrm{MkVE}^p(V,R_2) + \sum_{i=1}^{k} |e_i|^p && (24) \\ &\leq \mathrm{MkVE}^p(V,R_1) + \mathrm{MkVE}^p(V,R_2) + k d^{p/2} (\operatorname{diam} R)^p && (25) \end{aligned}$$

2. $|V_1| \geq k+1$, $|V_2| \leq k$. We know that (V_2, E_2) is complete. For each vertex $v_i \in V_2$ we add k edges $e_{i_1}, \ldots e_{i_k}$ with endpoints in V_1. As (V_1, E_1) and (V_2, E_2) are k-vertex-connected, removing less than k vertices from any of them will have a connected graph as a result. The only option then is to disconnect (V_1, E_1) and (V_2, E_2). Since each vertex in V_2 is connected

to k different vertices in V_1, we would have to remove at least k vertices to disconnect them. This means we have a k-vertex-connected graph on V. At most k^2 edges have been added this way, so

$$\begin{aligned} \mathrm{MkVE}^p(V,R) &\leq \mathrm{MkVE}^p(V,R_1) + \mathrm{MkVE}^p(V,R_2) + \sum_{i=1}^{|V_2|}\sum_{j=1}^{k} |e_{i_j}|^p && (26)\\ &\leq \mathrm{MkVE}^p(F,R_1) + \mathrm{MkVE}^p(F,R_2) + k^2 d^{p/2} (\operatorname{diam} R)^p && (27) \end{aligned}$$

3. $|V_1|, |V_2| \leq k$. As both (V_1, E_1) and (V_2, E_2) are k-vertex-connected, according to Lemma 4.6 they are complete. If we make a complete graph on V as well, we know that it is k-vertex-connected by Lemma 4.6. As $|V_1| + |V_2| < 2k$, we have to add fewer than k^2 edges to make a complete graph, so $\mathrm{MkVE}^p(V,R)$ will be bounded by a term similar to that in (26).

Since the case distinction is exhaustive, MkVE^p is subadditive. □

We can prove subadditivity for MkVP^p by combining the proofs of MkVE^p and MkEP^p.

Theorem 4.9. *For $p \geq 1$, MkVP^p is a geometric subadditive functional.*

Proof. Combining the proofs of Theorem 4.5 and 4.8, this theorem follows. □

4.3 Superadditivity

In the previous section, we have proved subaddivity for our original functionals. Unfortunately, as mentioned in Section 2, the original functionals that we consider here are not superadditive. Luckily their boundary functionals (Definition 2.17) do have this property. Again we start with the proof of superadditivity of MkEE_B^p.

Theorem 4.10. *For $p \geq 1$, MkEE_B^p is a superadditive functional.*

Proof. We will check that Definition 2.26 holds for MkEE_B^p, so we are given a k-edge-connected graph $G = (V, E)$ and a partitioning of R into rectangles R_1 and R_2. For each vertex in $V_1 = V \cap R_1$ that has an edge to a vertex in $V_2 = V \cap R_2$ we add an edge to the boundary between R_1 and R_2, and we do the same for vertices in V_2. If we replace the edges lost by the partition in this way, we keep the k-edge-connectedness in both parts, as both V_1 and V_2 (using the same notation as before) are locally k-edge-connected in E. Now V_1 and V_2 are locally k-edge-connected in E_1 and E_2 respectively. This holds as the k edge-disjoint paths from $u \in V_1$ to $v \in V_1$ that went trough V_2 now travel along the boundary (as an edge rooted to the boundary can count as up to k independent edges). We get an equivalent result for V_2. This means both (V_1, E_1) and (V_2, E_2) have k edge-disjoint paths between pairs of vertices in them. If we can show that the costs do not increase by this construction, we have that MkEE_B^p is superadditive.

We have to replace an edge c by two edges a and b to the new boundary. As these new edges are be perpendicular to the boundary we have that $|a| + |b| \leq |c|$, so also $|c|^p \geq ||a| + |b||^p \geq |a|^p + |b|^p$ for $p \geq 1$, where $|e|$ denotes the Euclidean length of an edge e. This means that the costs are not increasing by the construction, so

$$\mathrm{MkEE}_B^p(V,R) \geq \mathrm{MkEE}_B^p(V,R_1) + \mathrm{MkEE}_B^p(V,R_2) \qquad (28)$$

and therefore MkEE_B^p is superadditive for $p \geq 1$. □

As the definition of k-edge-connected simple boundary graphs and boundary multigraphs is slightly different (see Definition 2.18), we cannot literally copy the result of Theorem 4.10 for multigraphs.

Theorem 4.11. *For $p \geq 1$, MkEEm_B^p is a superadditive functional.*

Proof. We can use the same reasoning as in the proof of Theorem 4.10, but we replace edges lost by the partition by an equal amount of edges to the boundary (as we have a multigraph). We also do not increase the costs here as each added edge is still cheaper than the edge crossing it. So MkEEm_B^p is superadditive for $p \geq 1$. □

Proving superadditivity of MkEP_B^p is even easier than proving it of MkEE_B^p as the power of vertices that lose a connection is already large enough to reach to boundary.

Theorem 4.12. *For $p \geq 1$, MkEP_B^p is a superadditive functional.*

Proof. We apply the same construction as in the proof of Theorem 4.10. Notice that since $|a| \leq |c|$ and $|b| \leq |c|$, the edges to the boundary are already included without changing the power assignment. This means that $\mathrm{MkEP}_B^p(V, R) \geq \mathrm{MkEP}_B^p(V, R_1) + \mathrm{MkEP}_B^p(V, R_2)$ and therefore MkEP_B^p is superadditive for $p \geq 1$. □

For k-vertex-connected boundary graphs superadditivity is obtained in same way it was done for k-edge-connected boundary graphs.

Theorem 4.13. *For $p \geq 1$, MkVE_B^p is a superadditive functional.*

Proof. As in the proof of Theorem 4.10, we replace all edges lost by the partition by edges to the boundary and show that both parts still have k-vertex-connectedness. There are k vertex-disjoints paths in E between each pair of vertices in V_1, some of which used vertices in V_2. The path that used vertices in V_2 can now be redirected along the boundary and are therefore still vertex-disjoint (as the boundary does not count as a vertex). This means that (V_1, E_1) is also k-vertex-connected. We get an equivalent result for V_2. We showed in the proof of Theorem 4.10 already that we do not increase the costs by this construction, so we can conclude that MkVE_B^p is superadditive for $p \geq 1$. □

Again MkVP_B^p can easily be proved by combining the proofs of MkEP_B^p and MkVE_B^p.

Theorem 4.14. *For $p \geq 1$, MkVP_B^p is a superadditive functional.*

Proof. Combining the proofs of Theorem 4.12 and 4.13, this theorem follows. □

4.4 Growth Bounds

The previous two sections showed that the original functionals are subadditive, and their boundary functionals are superadditive. If we can also prove that each functional is close to their boundary functional, our functionals show a form of near additivity. We first need another lemma to prove closeness that appeared in Yukich [37, Lemma 3.3]. The proof can also be found there.

Lemma 4.15 (Yukich [37], growth bound)**.** *Let $0 < p < d$ and L^p be a subadditive Euclidean functional. Then there exists a constant $C = C(d, p)$ such that for all cubes $R \subset \mathbb{R}^d$ and all $V \subset R$ we have*

$$L^p(V, R) \leq C(\operatorname{diam} R)^p |V|^{(d-p)/d} \tag{29}$$

As the original functionals we consider in this thesis are all subadditive and Euclidean, we can get the following result.

Lemma 4.16. *For $p \geq 1$, MkEE^p, MkEEm^p, MkEP^p, MkVE^p, and MkVP^p all satisfy the growth bound from Lemma 4.15.*

Proof. This lemma follows directly by combining Theorem 4.1 and Lemma 4.15. □

4.5 Pointwise Closeness

With the results from Section 4.4, we can prove pointwise closeness of MkEE^p to MkEE^p_B.

Theorem 4.17. *For $1 \leq p < d$, MkEE^p is pointwise close to MkEE^p_B.*

Proof. We will check that Definition 2.27 holds, so we are a given an arbitrary finite set of points V. By definition we have $\mathrm{MkEE}^p(V,[0,1]^d) \geq \mathrm{MkEE}^p_B(V,[0,1]^d)$, we only need to check $\mathrm{MkEE}^p(V,[0,1]^d) \leq \mathrm{MkEE}^p_B(V,[0,1]^d) + \mathrm{o}(|V|^{(d-p)/p})$. We distinguish between the cases $|V| \leq k$ and $|V| \geq k+1$.

For $|V| \leq k$ we know that the corresponding graph in the original functional will be complete (per definition). In the boundary functional, it will have an edge to the boundary for every vertex in V, or it is complete. So we need to add at most $k(k-1)$ edges. This only costs $O(1)$ as k is constant, so we have $\mathrm{MkEE}^p(V,[0,1]^d) \leq \mathrm{MkEE}^p_B(V,[0,1]^d) + \mathrm{o}(|V|^{(d-p)/p})$.

For $|V| \geq k+1$ we need a different approach. We will first prove our variant of Theorem 4.18, following the proof of Lemma 3.8 from [37].

Lemma 4.18. *Let $V \subset [0,1]^d$, $|V| = n$, and $1 \leq p < d$. Consider a graph $G = (V,E)$ realising the optimal solution of $\mathrm{MkEE}^p_B(V,[0,1]^d)$. The sum of the p-th powers of the lengths of the edges connecting vertices in V with the boundary of $[0,1]^d$ is bounded by $O(n^{(d-p-1)/(d-1)})$.*

Proof. The proof depends upon a dyadic subdivision of $[0,1]^d$. This idea comes from Yukich [37]. Let Q_0 by the cube of edge length $1/3$ and centered within $[0,1]^d$. Let Q_1 be the cube of edge length $2/3$, also centered within $[0,1]^d$. Partition $Q_1 - Q_0$ into subcubes of edge length $1/6$; it is easy to verify that the number of such subcubes is bounded by $C6^{d-1}$ for some constant $C = C(d)$.

Continue with this subdivision recursively, so that at the jth stage we define cube Q_j of edge length $1 - 2(3 \cdot 2^j)^{-1}$ and partition $Q_j - Q_{j-1}$ into subcubes of edge length $(3 \cdot 2^j)^{-1}$. The number of such subcubes isbounded from above by $C3^{d-1}2^{j(d-1)}$. Carry out this recursion until the ℓth stage, where ℓ is the unique integer satisfying

$$2^{(\ell-1)(d-1)} \leq n \leq 2^{\ell(d-1)}. \tag{30}$$

This procedure produces nested cubes $Q_1 \subset Q_2 \subset \cdots \subset Q_\ell$. It produces a dyadic covering of the cube until the moat $[0,1]^d - Q_\ell$ has a width of $O(n^{-1/(d-1)})$. We use these properties to prove Theorem 4.18 as follows.

This dyadic subdivision partitions the largest cube Q_ℓ into at most

$$\sum_{j=0}^{k} C3^{d-1}2^{j(d-1)} \leq Cn \tag{31}$$

subsubes, each with an edge length equal to the distance between the subcube and the boundary of $[0,1]^d$. Furthermore, by partitioning each subcube into $(k2^y)^d$ congruent subcubes, where y is the least integer satisfying $2^y \geq d^{1/2}$, we obtain a partition $\mathcal{P}$ of Q_ℓ consisting of at most Cn subcubes with the property that the k times the diameter of each subcube is less than the distance to the boundary.

Observe that in an optimal boundary solution on V, each subcube Q in $\mathcal{P}$ contains at most k points in V which are rooted to the boundary. Indeed, if there were more than k point in $V \cap Q$ rooted to the boundary, we can do the following. We know that these points will not have edges directly between them as they already have k edge-disjoint paths between them, and edges between them would then not be optimal. So we can take one of them and connect them to all the other points rooted to the boundary (which are at least k), while removing the connection to the boundary. As the diameter of each subcube is less than the $1/k$ times the distance to the boundary, this relinking gives us a cheaper solution, contradicting the optimality of the solution.

The sum of the p-th powers of the lengths of the edges connecting vertices in $V \cap (Q_j - Q_{j-1})$ with the boundary is thus bounded by the product of the number of subcubes in $Q_j - Q_{j-1}$ and the p-th power of the common diameter of the subcubes, namely

$$C3^{d-1}2^{j(d-1)} \cdot (3 \cdot 2^j)^{-p}. \tag{32}$$

Summing over all $1 \leq j \leq \ell$ gives a bound for the sum of the p-th power of the lengths of the edges connecting points to the boundary in $V \cap Q_\ell$:

$$\sum_{j=1}^{\ell} C3^{d-1}2^{j(d-1)} \cdot (3 \cdot 2^j)^{-p} \leq \begin{cases} C \max\{n^{d-p-1}, \log n\} & \text{if } 1 \leq p \leq d-1 \\ C & \text{if } d-1 < p < d. \end{cases} \tag{33}$$

The $\log n$ term is needed to cover the case $p = d - 1$. The sum of the p-th powers of the lengths of the edges connecting vertices in $V \cap ([0,1]^d - Q_\ell)$ with the boundary is at most the product of $n = |V|$ and the p-th power of the width of the moat $[0,1]^d - Q_\ell$, i.e. at most

$$n \cdot Cn^{-p/(d-1)} = Cn^{(d-p-1)/(d-1)}. \tag{34}$$

Combining (33) and (34) establishes Theorem 4.18 for MkEE^p_B. □

We can now continue with the proof of Theorem 4.17. Consider $U \subset V$ the set of all vertices connected to the boundary, and $\mathcal{B} \subset \partial[0,1]^d$ the set of points on the boundary to which vertices are connected. Then $|U| \geq |\mathcal{B}|$. As we want to remove all edges to the boundary to get to a solution for our normal functional, we will need to add new edges as well. We even need a k-edge-connected graph on U, as the edged rooted to the boundary could be counting as k edges for these points. As used in the proof of Theorem 4.18, we know there are no edges between the points of U.

To get a good bound on the increase of costs by adding the edges to get a k-edge-connected graph on U, we will first prove the following lemma.

Lemma 4.19. *Fix $1 \leq p < d$ and let $G = (V, E)$ be a k-edge-connected graph realising the optimal solution for $\mathrm{MkEE}^p(V, [0,1]^d)$. Then there exists a constant $c = c(k, d)$ such that the degree of every vertex $v \in V$ is bounded by c.*

Proof. Let us assume to the contrary that there exists a vertex $v \in V$ for which the degree is not bounded by c. We then divide $[0,1]^d$ into cones originating from v and with the property that for every two points x and y in a cone we have $\angle(x, y, v) < \pi/3$. In this way, the number of cones we create is finite (as d is constant) and every point is covered by a cone. We consider a cone C with an unbounded number of points connected to v and look at the point y that is furthest from v and connected to v. Such a cone has to exist as the number of cones is bounded. Let us consider two cases.

In the first case, the degree of y is greater than c as well, and y is connected to all vertices in C that are also connected to v. This means both are connected to more than k vertices in C. In that case we can remove the edge between v and y as we would still have more than k edge-disjoint paths between v and y (and other points will not be affected). Removing an edge would only lower the cost for the graph, so this would contradict the optimality of the solution.

In the other case, we can find a vertex to which y is not connected, but v is. Let us call this vertex z and consider the triangle $\Delta(v, z, y)$. As we know that $\angle(z, v, y) < \pi/3$, either $\angle(z, y, v) > \pi/3$ or $\angle(v, z,)y > \pi/3$ (or both). Using that $|(v, y)| \geq |(v, z)|$, we can see that $|(y, z)| < |(v, y)|$. If we then replace the edge from v to y by the edge from y to z, the number of edge-disjoint paths from y to z or v cannot decrease. But as $|(y, z)| < |(v, y)|$, we also have $|(y, z)|^p < |(v, y)|^p$, meaning we lowered the cost, and still have a k-edge-connected graph. This again would contradict the optimality of G, so we have to conclude no such vertex v can exist. □

With this lemma we can continue the proof of Theorem 4.17. We can also get a bound on the length of each edge we need to add. By using the triangle inequality for $p = 1$ we get $|(u, v)| \leq |(u, u_B)| + |(u_B, v_B)| + |(v_B, v)|$, where u_B and v_B are the points where u and v respectively are rooted to the boundary. If we look at M, the set of edges used create a k-edge-connected graph on U, and use the relaxed triangle inequality $|(a, c)|^p \leq 2^{p-1}(|(a, b)|^p + |(b, c)|^p)$, we have:

$$\begin{aligned}
\sum_{(u,v)\in M} |(u,v)|^p &\leq 2^{p-1}\Big(\sum_{(u,\cdot)\in M} |(u,u_B)|^p + \sum_{(u,v)\in M} |(u_B,v)|^p\Big) && (35)\\
&\leq 2^{p-1}\Big(\sum_{(u,\cdot)\in M} |(u,u_B)|^p + 2^{p-1}\Big(\sum_{(u,v)\in M} |(u_B,v_B)|^p + \sum_{(\cdot,v)\in M} |(v_B,v)|^p\Big)\Big) && (36)\\
&\leq 4^{p-1}\Big(\sum_{(u,\cdot)\in M} |(u,u_B)|^p + \sum_{(u,v)\in M} |(u_B,v_B)|^p + \sum_{(\cdot,v)\in M} |(v_B,v)|^p\Big) && (37)\\
&\leq 4^p \sum_{(u,\cdot)\in M} |(u,u_B)|^p + 4^{p-1}\,\mathrm{MkEE}^p(\mathcal{B},[0,1]^d) && (38)\\
&\leq 4^p C_1 \sum_{u\in U} |(u,u_B)|^p + 4^{p-1}C_2|V|^{(d-p-1)/(d-1)} \quad \text{using Lemma 4.16} && (39)\\
&\leq 4^p C_1 C_3 |V|^{(d-p-1)/(d-1)} + \mathrm{o}(|V|^{(d-p)/d}) \quad \text{using Lemma 4.18} && (40)\\
&\leq \mathrm{o}(|V|^{(d-p)/d}) && (41)
\end{aligned}$$

Now we changed a graphs achieving the optimal solution for $\mathrm{MkEE}^p_B(V, [0, 1]^p)$ into a k-edge-connected graph. This means it is an upper bound for $\mathrm{MkEE}^p(V, [0, 1]^p)$. As we increased the solution value by at most $\mathrm{o}(|V|^{(d-p)/d})$, we have $\mathrm{MkEE}^p(V, [0, 1]^d) \leq \mathrm{MkEE}^p_B(V, [0, 1]^d) + \mathrm{o}(|V|^{(d-p)/p})$. This proves that MkEE^p_B and MkEE^p are pointwise close. □

Changing this proof to hold for multigraphs is rather simple.

Theorem 4.20. *For $1 \leq p < d$, MkEEm^p is pointwise close to MkEEm^p_B.*

Proof. The proof for the multigraph is the same as the proof of Theorem 4.17, except that we create a k-edge-connected multigraph on U. □

Proving pointwise closeness of MkEP^p and MkEP^p_B is a bit more difficult as not all theorems we used in the proof of Theorem 4.17 hold for these functionals.

Theorem 4.21. *For $1 \leq p < d$, MkEP^p is pointwise close to MkEP^p_B.*

Proof. The proof for power assignments is the same as the proof of Theorem 4.17, except that Theorem 4.18 does not hold directly for MkEP^p_B. Luckily, a similar theorem also holds for the power assignment functional MkEP^p_B:

Theorem 4.22. *Let $V \subset [0, 1]^d$, $|V| = n$, and $1 \leq p < d$. Consider a graph realising the optimal solution of $\mathrm{MkEP}^p_B(V, [0, 1]^d)$. The sum of the p-th powers of the lengths of the edges connecting vertices in V with the boundary of $[0, 1]^d$ is bounded by $O(n^{(d-p-1)/(d-1)})$.*

Proof. The proof is similar to that of Theorem 4.18, except that for power assignment, in case more than k points are rooted to the boundary, all of them are also connected to each other (as the diameter of the subcube is less than the distance to the boundary). Without changing the solution, we can remove one of the roots to the boundary. This also gives us that we only have to take into account at most k points rooted to the boundary in each subcube. The rest of the proof follows. □

As we know the degree of vertices in an optimal power assignment graph can be unbounded [12], we cannot show Lemma 4.19 for MkEP^p as well. We do however have the following lemma:

Lemma 4.23. *Fix $1 \leq p < d$, let $V \subset [0,1]^d$ be V is a finite subset and let $R \in \mathcal{R}$ be a d-dimensional rectangle. Then we have that* $\mathrm{MkEP}^p(V,R) \leq 2\,\mathrm{MkEE}^p(V,R)$.

Proof. Consider a graph $G = (V,E)$ achieving the optimal solution for $\mathrm{MkEE}^p(V,R)$. Then for each vertex $v \in V$ we take the longest edge from v and we set power $\mathrm{PA}(v)$ to the length of this edge. This means that all edges that were in E, will also be in the graph resulting from power assignment PA. So the power assignment graph resulting from power assignment PA will also be k-edge-connected, and has costs no more than twice $\mathrm{MkEE}^p(V,R)$. An optimal solution for $\mathrm{MkEP}^p(V,R)$ can only have costs lower or equal to that of PA, so $\mathrm{MkEP}^p(V,R) \leq 2\,\mathrm{MkEE}^p(V,R)$ follows. □

We can use Lemma 4.23 and equation (35) to get the following result for a k-edge-connected power assignment graph on U

$$\mathrm{MkEP}^p(U,[0,1]^d) \leq 2\,\mathrm{MkEE}^p(U,[0,1]^d) = 2\sum_{(u,v)\in M} \mathrm{d}(u,v)^p \leq \mathrm{o}(|V|^{(d-p)/d}), \tag{42}$$

where M is the set of edges used create a k-edge-connected graph on U. This gives us for MkEP^p that we have $\mathrm{MkEP}^p(V,[0,1]^d) \leq \mathrm{MkEP}^p_B(V,[0,1]^d) + \mathrm{o}(|V|^{(d-p)/p})$, and therefores MkEP^p_B and MkEP^p are pointwise close. □

To prove pointwise closeness for MkVE^p and MkVP^p is straightforward now.

Theorem 4.24. *For $1 \leq p < d$,* MkVE^p *is pointwise close to* MkVE^p_B.

Proof. The proof for k-vertex-connectedness is the same as the proof of Theorem 4.17, except that we create a k-vertex-connected graph on U. □

Theorem 4.25. *For $1 \leq p < d$,* MkVP^p *is pointwise close to* MkVP^p_B.

Proof. The proof for k-vertex-connectedness with power assignments is the same as the proof of Theorem 4.21, except that we create a k-vertex-connected power assignment graph on U. □

The following is a weaker version of pointwise closeness, as it only holds in mean. It is included for completeness nonetheless.

Remark 4.26. *For $1 \leq p < d$,* MkEE^p, MkEEm^p, MkEP^p, MkVE^p, *and* MkVP^p, *are all close in mean to their corresponding boundary functional.*

4.6 Smoothness

We have shown geometric subadditivity, superadditivity and pointwise closeness, creating a powerful set of properties for other results. These properties are more useful for other obtaining other results when functional also is smooth. Unfortunately, we have not been able to prove smoothness for MkVE^p and MkVP^p. One of the problems is that k-vertex-connectedness is not a transitive property as k-edge-connectedness is. We do have proofs for MkEE^p, MkEEm^p and MkEP^p and their respective boundary functionals. At the end of this section we will also show a counterexample of a different approach we tried to use for proving smoothness of MkEE^p.

Theorem 4.27. *For $1 \leq p < d$,* MkEE^p *is smooth.*

Proof. We will check that Definition 2.29 holds, so we start with finite sets $U, V \subset \mathbb{R}^d$. As MkEE^p is a subadditive Euclidean functional, we can first use subadditivity to get

$$\mathrm{MkEE}^p(U \cup V,[0,1]^d) \leq \mathrm{MkEE}^p(U,[0,1]^d) + \mathrm{MkEE}^p(V,[0,1]^d) + c_1 \tag{43}$$

for some constant $c_1 = c_1(d,p)$. Using Lemma 4.16 we get

$$\begin{aligned} \mathrm{MkEE}^p(U \cup V,[0,1]^d) &\leq \mathrm{MkEE}^p(U,[0,1]^d) + c|V|^{(d-p)/d} + c_1 \\ &= \mathrm{MkEE}^p(U,[0,1]^d) + O(|V|^{(d-p)/d}). \end{aligned} \tag{44}$$

Now to prove smoothness we will only need to show

$$\mathrm{MkEE}^p(U,[0,1]^d) - \mathrm{MkEE}^p(U \cup V,[0,1]^d) \leq O(|V|^{(d-p)/d}). \tag{45}$$

We first consider the case $k = 2$. We will start with a graph $G = (U \cup V, E)$ achieving the optimal solution for $\mathrm{MkEE}^p(U \cup V,[0,1]^d)$. After removing the set V we will show that we can alter G to obtain a k-edge-connected graph on U by increasing the cost at most $O(|V|^{(d-p)/d})$. We consider the set of neighbours N_V containing all vertices that are direct neighbours of a vertex in V, but not vertex in V itself. By Lemma 4.19 we know that $|N_V| \leq c|V|$ for some constant c, that only depends on k and d.

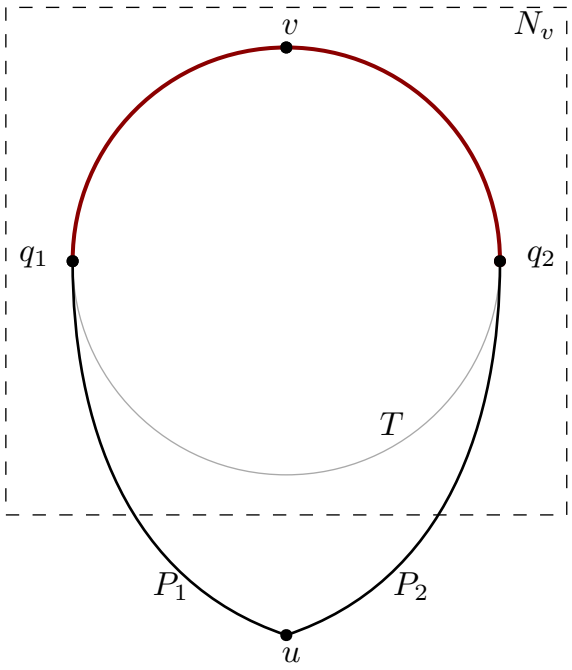


Figure 3: *Explanation for the proof of Theorem 4.27. Paths P_1 and P_2 meet the first vertex in N_v, being q_1 and q_2 respectively. As both q_1 and q_2 are on tour T, we can easily complete paths P_1 and P_2 on T to reach v without having to use the same edge anywhere.*

We now compute a minimal length TSP tour T on N_V. As mapping a finite point set N_V to the length of the optimal travelling salesman tour on N_V is an Euclidean functional, we can use Lemma 4.16. By this growth bound, we know the weight $w(T)$ will be no more than $O(|N_V|^{(d-p)/d}) = O(|V|^{(d-p)/d})$. Let F be edge set left from E after we remove V from G. We combine T with F, only adding edges to F that are not in it yet. Consider the graph $(U, F \cup T)$. We need to prove that it is 2-edge-connected. Consider two vertices $u, v \in U$, we need to prove that there exist two edge-disjoint paths between u and v, given that there are at least k edge-disjoint paths between u and v in G. We distinguish three cases.

1. Both u and v are in N_V. As T is a tour, there are two edge-disjoint paths from u to v.
2. Exactly one of u, v is in N_V and one is not. W.l.o.g. $v \in N_V$ and $u \in U \setminus N_V$. Consider the two edge-disjoint (u, v)-paths P_1 and P_2 that existed in $(U \cup V, E)$. If these paths did not use a vertex from $V \cup N_V$, nothing is changed. In case a path did use a vertex from $V \cup N_V$, we know it has to enter and exit V through N_V as it contains all neighbours of V. Following path P_i from u, let q_i be the first vertex in N_V. From q_i we can continue the path along T to v. When both paths use $V \cup N_V$, we know q_1 and q_2 are on a tour, so they can both reach v using T without crossing each other. This means we still have two edge-disjoint paths from u to v. Figure 3 illustrates this.

3. Both u and v are not in N_V. Take a vertex $x \in N_V$, then by Item 2 we know there are two edge-disjoint paths from u to x and from x to v. By transitivity of k-edge-connectedness (Theorem 4.2), we know that there are also two edge-disjoint paths from u to v.

In all cases there are two edge-disjoint paths. Thus $(U, F \cup T)$ is 2-edge-connected. So we found a k-edge-connected graph on U by increasing the cost no more than $O(|V|^{(d-p)/d})$ from a graph achieving the optimal solution for $\text{MkEE}^p(U \cup V, [0,1]^d)$. Then it follows that $\text{MkEE}^p(U \cup V, [0,1]^d) + O(|V|^{(d-p)/d}) \geq \text{MkEE}^p(U, [0,1]^d)$. This means MkEE^p is smooth for $k = 2$.

For $k \geq 3$, we apply a similar argument. Instead of creating a TSP tour on N_V, we create a k-edge-connected graph T on N_V. Again this will increase the cost by no more than $O(|N_V|^{(d-p)/d}) = O(|V|^{(d-p)/d})$ as we have Lemma 4.16. For the graph $(U, F \cup T)$ we will prove it is k-edge-connected. Consider two vertices $u, v \in U$. We need to prove that there exist k edge-disjoint paths between u and v, given that there are at least k edge-disjoint paths between u and v in G. We distinguish between three cases.

1. Both u and v are in N_V. As T is a k-edge-connected graph, there are k edge-disjoint paths from u to v.

2. Exactly one of u, v is in N_V and one is not. W.l.o.g. $v \in N_V$ and $u \in U \setminus N_V$. Consider k edge-disjoint paths $P_1, \ldots, P_k$ from u to v in $(U \cup V, E)$. If path P_i does not use a vertex from $V \cup N_V$, we keep it. If P_i does use a vertex from $V \cup N_V$, let q_i be first vertex in N_V when following path P_i from u. For all these paths P_i we only keep them up to q_i. Now we have at most k paths temporarily halted at a vertex in N_V. These paths all need to be completed to reach v. Say we add a vertex x to our graph and we connect it to all q_i, adding some edges multiple times to make sure x has exactly degree k. $N_V \cup \{x\}$ is also k-edge-connected as we do not have a cut with less than k edges (T already was k-edge-connected and x is connected to T with k edges). Using Definition 2.2, we know there are k edge-disjoint paths between x and v, and that all these paths will have to use the vertices q_i. So from every q_i we can find an edge-disjoint path in T to v. If we combine all halted P_i with these paths, we in total have k edge-disjoint (u, v)-paths again.

3. Both u and v are not in N_V. Take a $x \in N_V$, then by Item 2 we know there are k edge-disjoint paths from u to x and from x to v. By transitivity of k-edge-connectedness (Theorem 4.2), we know there are also k edge-disjoint paths from u to v.

In all cases there are k edge-disjoint paths. Thus $(U, F \cup T)$ is k-edge-connected. So we found a k-edge-connected graph on U by increasing the cost no more than $O(|V|^{(d-p)/d})$ from a graph achieving the optimal solution for $\text{MkEE}^p(U \cup V, [0,1]^d)$. Then it follows that $\text{MkEE}^p(U \cup V, [0,1]^d) + O(|V|^{(d-p)/d}) \geq \text{MkEE}^p(U, [0,1]^d)$. This means MkEE^p is smooth for $k \geq 3$. □

This theorem is easily extended to hold for MkEEm^p as well.

Theorem 4.28. *For $1 \leq p < d$, MkEEm^p is smooth.*

Proof. The proof is similar to that of Theorem 4.27, except for the fact that we create a k-edge-connected multigraph on N_V. □

If we try to extend the proof of Theorem 4.27 to power assignments, we get into trouble with the possible unbounded degree of power assignment. So instead of trying to bound the number of vertices connected to one vertex, we will bound the number of k-edge-connected components connected to one vertex. To do this, we first need another lemma. We omit the proof, which can be found in De Graaf and Manthey [12, Lemma 3.2].

Lemma 4.29. *Let $x, y \in V$, let $v \in [0,1]^d$, and assume that x and y have power $\text{PA}(x) \geq |(x,v)|^p$ and $\text{PA}(y) \geq |(y,v)|^p$, respectively. Assume further that $|(x,v)| \leq |(y,v)|$ and that $\angle(x,v,y) \leq \alpha$ with $\alpha \leq \pi/3$. Then the following holds:*

(a) $\mathrm{PA}(y) \geq |(x,y)|^p$*, i.e., y has sufficient power to reach x.*

(b) If x and y are not connected (i.e., $\mathrm{PA}(x) < |(x,y)|^p$*), then* $|(y,v)| > 2\cos(\alpha) \cdot |(x,v)|$.

With this lemma, we can prove smoothness for MkEP^p. This proof is based on the proof of Lemma 3.6 in De Graaf and Manthey [12].

Theorem 4.30. *For* $1 \leq p < d$, MkEP^p *is smooth.*

Proof. To prove smoothness, we will follow the proof of Lemma 3.6 in De Graaf and Manthey [12]. Consider a graph $G = (U \cup V, E)$ achieving the optimal solution for $\mathrm{MkEP}^p(U \cup V, R)$. The problem for smoothness is that the degree $\deg_G(v)$ of vertices $v \in V$ can be unbounded. The idea is to exploit the fact that removing $v \in V$ also frees some power. Roughly speaking, we proceed as follows: Let $v \in V$ be a vertex of possibly large degree. We add the power of v to some vertices close to v. The graph obtained from removing v and distributing its energy has only a constant number of k-edge-connected components (either seperate vertices, or components with $k+1$ or more vertices). To prove this, Lemma 4.29 is crucial. We consider cones rooted at v with the following properties:

- The cones have a small angle α with v; for all cones C and for all $x, y \in C$, we have $\angle(x,v,y) < \alpha = \pi/6$.
- Every point in $[0,1]^d$ is covered by some cone.
- The number of cones m is finite (as d is a constant).

Let $C_1, \ldots, C_m$ be these cones. By abusing notation, let C_i also denote all points $C_i \cap (U \cup V \setminus \{x\})$ that are adjacent to v in G. Let $x_{i_1}, \ldots, x_{i_k}$ be the k points in C_i closest to v and let y_i be farthest from v. (We ignore C_i if $C_i \cap U = \emptyset$). Let $\ell_i = |(y_i, v)|$ be the maximum distance of a vertex in C_i to v and let $\ell = \max_i \ell_i$. We note that $\ell \leq \sqrt[p]{\mathrm{PA}(v)}$.

We increase the prower of the k closest points in each C_i by $\ell^p/(mk)$. Since the power of v is at least ℓ^p and we have m cones, we can account for this increase by the removal of v. As $\alpha = \pi/6$, and as $x_{i_1}, \ldots, x_{i_k}$ are the closest points to v, any point in C_i is closer to $x_{i_1}, \ldots, x_{i_k}$ than to v. According to Lemma 4.29(a), every point in $C_i \setminus \{x_{i_1}, \ldots, x_{i_k}\}$ has sufficient power to reach all vertices in $x_{i_1}, \ldots, x_{i_k}$. There will now be an edge between $x_{i_1}, \ldots, x_{i_k}$ and every point $z \in C_i$ that has a distance of at most $\ell/\sqrt[p]{mk}$. Now let z_1 be the first vertex not connected to all $x_{i_1}, \ldots, x_{i_k}$ as it has too little power. We can use Lemma 4.29(b) which implies that if x and y are not connected, then $|(y,v)| > 2\cos(\alpha)|(x,v)| = \sqrt{3}|(x,v)|$. We section the cone in pieces such that if x and y are in one piece, we have $|(y,v)| \leq \sqrt{3}|(x,v)|$. In this way, adjacent pieces are scaled by a factor $\sqrt{3}$. We cover all vertices in the cone in this way, starting with the piece containing all $x_{i_1}, \ldots, x_{i_k}$ up to z_1. Let us denote the number of pieces we need to cover the cone by h. We can state that $\ell \geq \ell_i = |(y_i,v)| \geq \sqrt{3}^{h-1}|(z_1,v)| \geq \ell/\sqrt[p]{mk}$. This implies that $h \leq \log_{\sqrt{3}}(\sqrt[p]{mk}) + 1$.

For each piece, we have two options.

1. The number of vertices in there is smaller or equal to k. In this case we just count all of them as separate k-edge-connected components.
2. The number of vertices in there is larger or equal to $k+1$. As $|(y,v)| \leq \sqrt{3}|(x,v)|$ for all x and y in one piece, we know all vertices in this piece are connected to each other by Lemma 4.29. A complete graph on $k+1$ vertices is k-edge-connected, so we can count this whole piece as one k-edge-connected component.

This means that the number of k-edge-connected components per cone after removing k and redistributing the power is bounded by $h \cdot k = k \log_{\sqrt{3}}(\sqrt[p]{mk}) + k$, which is a constant number. As the number of cones is finite as well, the total number of k-edge-connected components is bounded

by a constant as well. If we remove $|V|$ points, the graph fall into at most $O(|V|)$ k-edge-connected components. Creating a k-edge-connected graph on these components will increase the costs no more than $O(|V|^{(d-p)/d})$, meaning we can follow the proof from Theorem 4.27. This then proves that MkEP^p is also smooth for $k \geq 2$. □

Smoothness for the boundary functionals now follows almost automatically.

Theorem 4.31. *For $1 \leq p < d$, MkEE^p_B, MkEEm^p_B and MkEP^p_B are smooth.*

Proof. The proofs are similar to those of Theorem 4.27, 4.28 and 4.30, respectively. For removing V, we ignore the possible connections to the boundary and create a boundary k-edge-connected graph on N_V for MkEE^p_B, a boundary k-edge-connected multigraph on N_V for MkEEm^p_B, and a boundary k-edge-connected power assignment graph on N_V for MkEP^p_B. □

Before we figured out that we could simply create a k-edge-connected graph on N_v, we tried a different approach. We expected that removing one vertex from a k-edge-connected graph would give a constant number of locally k-edge-components, but this was not the case. We will show a counterexample of our hypothesis.

Lemma 4.32. *For $1 \leq p < d$, $k \in \mathbb{N}$ and for infinitely many n, there exist instances of $|V| = n$ vertices in $[0,1]^d$ such that removing one vertex from the graph achieving the optimal solution of $\text{MkEE}^p(V, [0,1]^d)$ leaves a graph with $O(n)$ k-edge-connected components.*

Proof. We will prove this by giving an example of graphs achieving this behavior. Figure 4 shows this example for $k = 5$. We will explain the idea behind this graph $G = (V, E)$. We start of with a vertex v, which will be the vertex we remove. v is connected to k vertices on layer 1 (indicated by the number 1 in Figure 4). Each of these vertices has connections to $k-1$ vertices on the next layer. These vertices in layer 2 are connected to each other such that they form a circle with one edge missing. Each of these vertices has edges to $k-2$ vertices on layer 3. As with layer 2, the vertices in layer 3 again form a circle with one edge missing.

The structure between layer 2 and 3 can now by copied an arbitrary number of times. So, vertices on layer i would form a circle with one edge missing, and each of them has edges to $k-2$ vertices in layer $i+1$. The total amount of layers is denoted by m. Then on layer $m-1$ the vertices again form a circle with one edge missing. We call this missing edge $(u_{m-1,1}, u_{m-1,s})$ where s is the number of vertices on layer $m-1$. Then, to make sure G is k-edge-connected, we add an extra edge from $u_{m-1,1}$ and $u_{m-1,s}$ to their neighbouring vertex on layer $m-1$. For layer m we form a cirle with edge $(u_{m,1}, u_{m,t})$ missing, where t is the number of vertices on layer m. To ensure k-edge-connectedness, we need $\lceil \frac{k}{2} \rceil$ edges between neighbouring vertices on layer m instead of just 1 edge. To account for the missing edge $u_{m,1}$ has $k-1$ edges $u_{m,2}$, $u_{m,2}$ has $k-2$ to $u_{m,3}$, and so on, until $k-i$ would be lower than $\lceil \frac{k}{2} \rceil$. This works similarly for $u_{m,t}$.

We can verify this multigraph is k-edge-connected, as no removal of $k-1$ edges will disconnect the graph. However, we were looking for a simple graph. Luckily, we can alter this quite easily. By transitivity of k-edge-connectedness, G would also be k-edge-connected if all vertices (with exception of v) would be k-edge-connected components. So by replacing all vertices with multiple edges to another vertex by k-edge-connected components, we can turn G into a simple graph. For convenience of argumentation, we will still refer to each k-edge-connected component as a vertex.

What happens when we remove v? All vertices in layer 1 only have $k-1$ edges, so they all belong to seperate k-edge-connected components. At that point in layer 2, the vertices that miss the edge from the circle only have $k-1$ to other vertices that are not a seperate component yet, so they become seperate components as as well. By repeating this line of reasoning again and again, eventually all vertices will be in seperate k-edge-connected components. This means by removal of v, a total of

$$k + k(k-1) + \sum_{i=1}^{m-2} k(k-1)(k-2)^i = O(n) \tag{46}$$

k-edge-connected components were formed. This proves the lemma for $k \geq 3$. In case $k = 2$ we can use a simple cycle to prove this lemma. For $k = 1$ we can use a star and remove the central vertex.

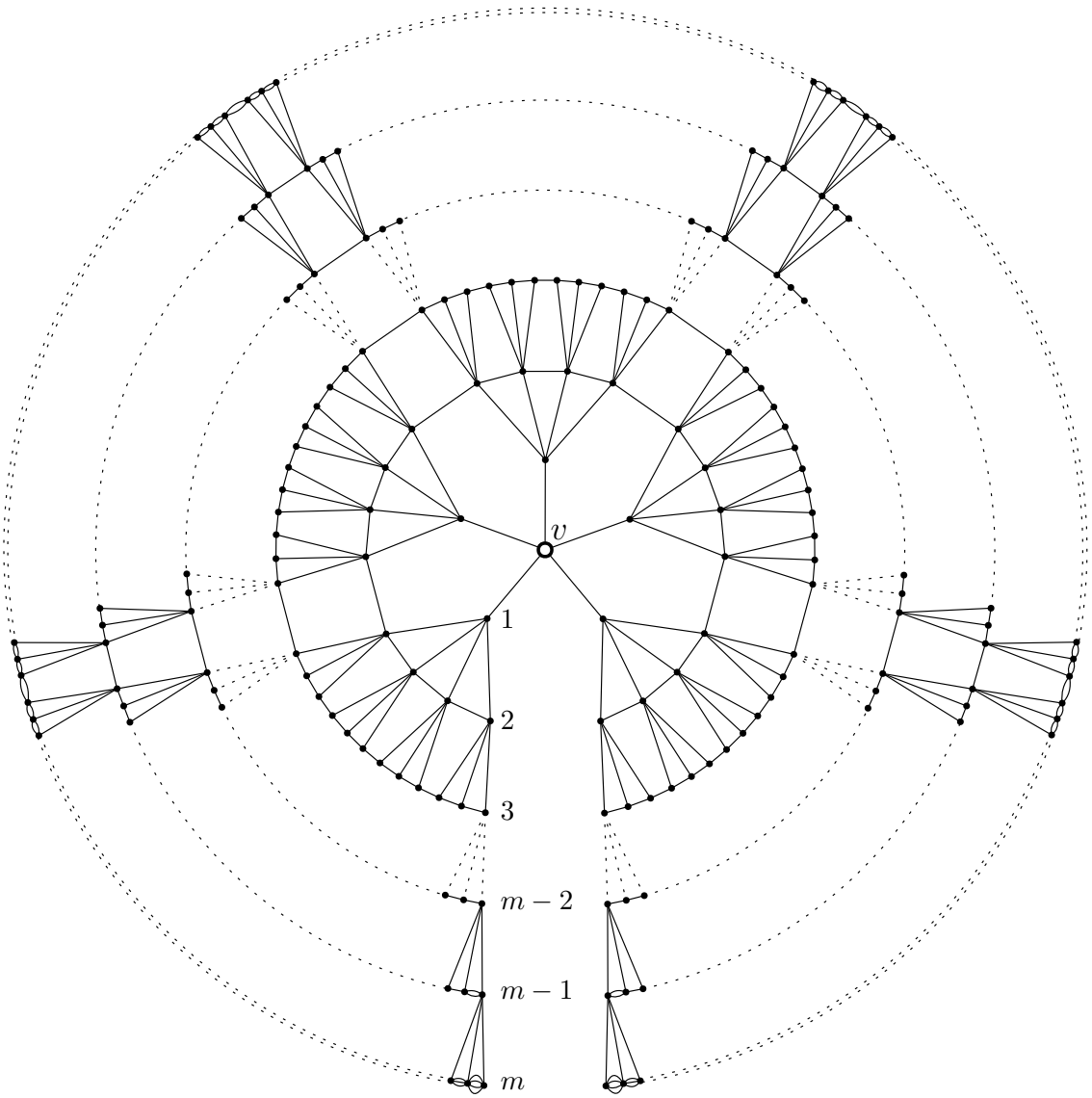


Figure 4: *Example graph for Lemma 4.32 in case $k = 5$. If vertex v is removed the graph will have $O(n)$ k-edge-connected components.*

□

4.7 Probabilistic and Limit Theorems

As we have geometric subadditivity, superadditivity, pointwise closeness, and smoothness for some functionals, several other theorems directly follow. All these properties are just tools to obtain useful results. These are mentioned in this section, along with a few other interesting theorems. We start with results similar to those in De Graaf and Manthey [12, Lemma 3.10–3.12]. These lemmas give a maximum length of an edge in an optimal k-edge-connected graph with high probability. Here $V = \{x_1, \ldots, x_n\} \subset [0, 1]^d$.

Lemma 4.33. *For every $\beta > 0$, there exists a $c_{ball} = c_{ball}(\beta, d)$ such that, with a probability of at least $1 - n^{-\beta}$, every hyperball of radius $r_{ball} = k \cdot c_{ball} \cdot (\log n/n)^{1/d}$ and with center in $[0,1]^d$ contains at least k points of V in its interior.*

Proof. We use Lemma 3.10 from [12] and see that at least k balls of radius $c_{ball} \cdot (\log n/n)^{1/d}$ fit in our hyperball. This means in the hyperball there will be at least k points of V in its interior. □

Lemma 4.34. *Assume that every hyperball of radius r_{ball} with center in $[0,1]^d$ contains at least k points of V. Then the following holds: For every choice of $u, v \in [0,1]^d$ with $|(u,v)| \geq 2r_{ball}$, there exists a point $w \in V$ with the properties*

- $|(u,w)| \leq 2r_{ball}$ *and*
- $|(v,w)| < |(u,v)|$.

Proof. The set of candidates for w contains a ball of radius r_{ball}, namely a ball of this radius whose center is at a distance of r_{ball} from u on the line between u and v. □

Lemma 4.35. *Fix $1 < p < d$. Consider graphs $G = (V, E)$ and $H = (V, F)$ achieving the optimal solution for $\mathrm{MkEE}^p(V, [0,1]^d)$ and $\mathrm{MkEE}^p_B(V, [0,1]^d)$ respectively. For every $\beta > 0$, there exists a constant $c_{edge} = c_{edge}(\beta)$ such that, with a probability of at least $1 - n^{-\beta}$, every edge of G and H is of length at most $r_{edge} = c_{edge} \cdot (\log n/n)^{1/d}$.*

Proof. We restrict this proof to graphs achieving the optimal solution for $\mathrm{MkEE}^p(V, [0,1]^d)$. The proof for MkEE^p_B is almost identical.

Let $G = (V, E)$ be a graph achieving the optimal solution for $\mathrm{MkEE}^p(V, [0,1]^d)$, and let (x, y) be a longest edge in G. Let $c_{edge} = 4kc_{ball}/\sqrt{4^{(p-1)/p} - 1}$. According to Lemma 4.33, with probability at least $1 - n^{-\beta}$, every ball B with radius $r_{ball} = \frac{1}{2} c_{edge} \cdot (\log n/n)^{1/d}$ contains at least k points of V. Now suppose $|(x,y)| > r_{edge}$. Let $B(r)$ be a hyperball of radius r_{ball} centered in the middle of (x, y). Then with probability at least $1 - n^{-\beta}$, $B(r)$ contains at least k points, $z_1, \ldots, z_\ell$ with $\ell \geq k$, using Lemma 4.33. Either all $z_1, \ldots z_\ell$ are connected to both x and y. This means we can remove xy and x and y would still have k edge-disjoint paths between them, so G would still be k-edge-connected. As G is optimal and $c_{xy} > 0$, this leads to a contradiction. Or at least one z_j is not connected to both x and y. We can replace (x, y) with (x, z_j) and (z_j, y). We only treat the case $|(x, z_j)| = |(z_j, y)|$ here, the other cases are similar as z_j is in a hyperball. When $|(x, z_j)| = |(z_j, y)|$, replacing the edges lowers the total cost by:

$$
\begin{aligned}
|(x,z_j)|^p + |(z_j,y)|^p &\leq \sqrt{(\tfrac{1}{2}|(x,y)|)^2 + r_{ball}^2}^{\,p} + \sqrt{(\tfrac{1}{2}|(x,y)|)^2 + r_{ball}^2}^{\,p} && (47)\\
&= 2\left(\tfrac{1}{4}|(x,y)|^2 + \left(\frac{k\sqrt{4^{(p-1)/p} - 1}}{4k} r_{edge}\right)^2\right)^{p/2} && (48)\\
&< 2\left(\frac{1 + \sqrt{4^{(p-1)/p} - 1}}{4}|(x,y)|\right)^p && (49)\\
&\leq 2\left(\frac{1}{2^{1/p}}|(x,y)|\right)^p && (50)\\
&= |(x,y)|^p && (51)
\end{aligned}
$$

Also in this case x and y would have k edge-disjoint paths between them, contradicting the optimality of G. So we can conclude that with a probability of at least $1 - n^{-\beta}$, the longest edge in an optimal MkEE^p graph is of length at most r_{edge}. □

For the following theorem we again use results from Yukich [37]. By using the results obtained in Section 4.2 – 4.6, we can get convergence in mean, and even complete convergence for our smooth functionals. This shows that our functionals do not grow very fast compared to the number of nodes in the set V. They even converge to a set constant.

Theorem 4.36. *Fix $1 \leq p < d$ and $k \in \mathbb{N}$. Let L^p be either* MkEE^p, MkEEm^p, *or* MkEP^p. *Then there exists a positive constant $\alpha = \alpha(L^p, d, k)$ such that*

$$\lim_{n\to\infty} L^p(V,R)/n^{(d-p)/p} = \alpha \quad c.c., \text{ and} \tag{52}$$

$$\lim_{n\to\infty} L^p_B(V,R)/n^{(d-p)/p} = \alpha \quad c.c., \tag{53}$$

where $n = |V|$. Here $\alpha(L^p, d, k)$ is equal to $\alpha(L^p_B, d, k)$, and c.c. denotes complete convergence.

Proof. As MkEE^p, MkEEm^p and MkEP^p are smooth subadditive Euclidean functionals which are pointwise close to their respective smooth superadditive boundary functionals, we can use the result in Yukich [37, Theorem 4.1] to directly get this theorem. □

The following corollary is a weaker version of Theorem 4.36, but is stated here for completeness.

Corollary 4.37. *Fix $1 \leq p < d$ and $k \in \mathbb{N}$. Let L^p be either* MkEE^p, MkEEm^p, *or* MkEP^p. *Then there exists a positive constant $\alpha = \alpha(L^p, d, k)$ such that*

$$\lim_{n\to\infty} \mathbb{E}[L^p(V,R)]/n^{(d-p)/p} = \alpha \quad \text{and} \tag{54}$$

$$\lim_{n\to\infty} \mathbb{E}[L^p_B(V,R)]/n^{(d-p)/p} = \alpha \tag{55}$$

where $n = |V|$. Here $\alpha(L^p, d, k)$ is equal to $\alpha(L^p_B, d, k)$, and both are equal to the constants in Theorem 4.36.

Proof. This result follows directly from Theorem 4.36, as Limit (54) is only a version in mean of Limit (52). □

The following famous limit theorem is due to Rhee [31]. This theorem states that the solution value will not be far from its expected value when we look at randomly places vertices.

Theorem 4.38. *Fix $d \geq 2$, $1 \leq p < d$ and $k \in \mathbb{N}$. Let L^p be either* MkEE^p, MkEEm^p, MkEP^p. *Let X_i, $i \geq 1$ be independent random variables with values in $[0,1]^d$. Then there exists constants $c_1 = c_1(L^p, d)$ and $c_2 = c_2(L^p, d)$ such that for all $t > 0$ we have*

$$\mathbb{P}[|L^p(\{X_1, \ldots, X_n\}, [0,1]^d) - \mathbb{E}[L^p(\{X_1, \ldots, X_n\}, [0,1]^d)]| > t] \leq c_1 \exp\left(\frac{-c_2 t^{2d/(d-p)}}{n}\right). \tag{56}$$

Proof. As MkEE^p, MkEEm^p and MkEP^p are smooth subadditive Euclidean functionals which are pointwise close to their respective superadditive boundary functionals, we can use the theorem in Rhee [31] to instantly get this result. □

The rates of convergence of means of functionals are useful for analysing certain heuristics. Theorem 4.39 gives us nice bound on these rates. The reason this theorem uses Poisson variables is that this creates a lot of indepence. The indepence can then be used to prove this theorem more easily.

Theorem 4.39 (rates of convergence of means)**.** *Fix $d \geq 2$, $1 \leq p < d$ and $k \in \mathbb{N}$. Let L^p be either* MkEE^p, MkEEm^p, MkEP^p, MkVE^p, *or* MkVP^p. *If N is an independent Poisson random variable with parameter n, then we have*

$$|\mathbb{E}[L^p(\{U_1, \ldots, U_N\}, R)] - \alpha n^{(d-p)/d}| \leq C \max\{n^{(d-p-1)/d}, 1\} \tag{57}$$

where $\alpha = \alpha(L^p, d, k)$ is the same constant as in Theorem 4.36. Moreover, for L^p being either MkEE^p, MkEEm^p *of* MkEP^p *we have*

$$|\mathbb{E}[L^p(\{U_1, \ldots, U_N\}, R)] - \alpha n^{(d-p)/d}| \leq C \max\{n^{(d-p)/2d}, n^{(d-p-1)/d}\} \tag{58}$$

Proof. This result follows directly from Yukich [37, Theorem 5.1] as MkEE^p, MkEEm^p, MkEP^p, MkVE^p, and MkVP^p are subadditive Euclidean functionals which are close in mean to their respective superadditive boundary functionals, and as MkEE^p, MkEEm^p and MkEP^p are smooth as well. □

Yukich [37] presented an umbrella theorem for Euclidean functionals, both on compact sets and on $\mathbb{R}^d, d \geq 2$. These theorems are an extension of the limit law obtained in Theorem 4.36. Now the variables are not restricted to be drawn from a uniform distribution. We first show the result for our smooth functionals on compact sets.

Theorem 4.40 (umbrella theorem for Euclidean functionals on compact sets)**.** *Fix $d \geq 2$, $1 \leq p < d$ and $k \in \mathbb{N}$. Let L^p be either MkEE^p, MkEEm^p, or MkEP^p. Let $(X_i)_{i\geq 1}$ be i.i.d. random variables with values in $[0,1]^d$, then we have*

$$\lim_{n\to\infty} L^p(\{X_1,\ldots,X_n\},R)/n^{(d-p)/d} = \alpha \int_{[0,1]^d} f(x)^{(d-p)/d} dx \quad c.c., \tag{59}$$

where $\alpha = \alpha(L^p,d,k)$ is the same constant as in Theorem 4.36, and f is the density of the absolutely continuous part of the distribution of X_1.

Proof. As MkEE^p, MkEEm^p and MkEP^p are smooth subadditive Euclidean functionals which are pointwise close to their respective smooth superadditive boundary functionals, we can use the result in Yukich [37, Theorem 7.1]. □

To also obtain this result on $\mathbb{R}^d, d \geq 2$, we need some additional requirement. First of all, the functional has to be simple subadditive instead of just geometric subadditive. Luckily, this results follows immediately, since we proved geometric subadditivity without explicitly using that both R and its partitions were rectangles.

Theorem 4.41. *Fix $1 \leq p < d$ and $k \in \mathbb{N}$. Then MkEE^p, MkEEm^p, MkEP^p, MkVE^p, and MkVP^p are all simple subadditive.*

Proof. This results follows directly from the proof for geometric subadditivity, as the shape of the sets was not explicitly used. □

The following definition makes stating Theorem 4.43 a lot more compact.

Definition 4.42. *Let A_0 denote the ball in $\mathbb{R}^d$ centered at the origin and with radius 2. For all $k \geq 1$, let A_k denote the annular shell centered around A_0 with inner radius 2^k and outer radius 2^{k+1}. Given $f \in \mathrm{L}^1(\mathbb{R}^d)$, set*

$$a_k(f) := 2^{dkp/(d-p)} \int_{A_k} f(x)dx \tag{60}$$

The umbrella theorem on $\mathbb{R}^d, d \geq 2$ is as follows, as an extension of Theorem 4.40.

Theorem 4.43 (umbrella theorem for Euclidean functionals on $\mathbb{R}^d$, $d \geq 2$)**.** *Fix $d \geq 2$, $1 \leq p < d$ and $k \in \mathbb{N}$. Let $(X_i)_{i\geq 1}$ be i.i.d. random variables with an absolutely continuous distribution on $\mathbb{R}^d$ having a density $f(x)$. If we have*

$$\int_{\mathbb{R}^d} f(x)^{(d-p)/d} dx < \infty \quad \text{and} \tag{61}$$

$$\sum_{k=1}^{\infty} (a_k(f))^{(d-p)/d} < \infty, \tag{62}$$

then for L^p being either MkEE^p, MkEEm^p or MkEP^p we have

$$\lim_{n\to\infty} L^p(\{X_1,\ldots,X_n\},R)/n^{(d-p)/d} = \alpha \int_{\mathbb{R}^d} f(x)^{(d-p)/d} dx \quad a.s., \tag{63}$$

where $\alpha = \alpha(L^p,d,k)$ is the same constant as in Theorem 4.36.

Proof. As MkEE^p, MkEEm^p and MkEP^p are smooth simple subadditive Euclidean functionals which are pointwise close to their respective smooth superadditive boundary functionals, we can use the result in Yukich [37, Theorem 7.6]. □

Condition (62) is satisfied whenever f satisfies $\int_{\mathbb{R}^d} |x|^r f(x)dx < \infty$ for some $r > d/(d-p)$ (see Yukich [37]). Examples of functions f satisfying condition (62) are f having a Gaussian distribution, or f having compact support (it is zero outside of a compact set).

4.8 Partitioning Scheme

We apply the framework as given by Bläser et al. [6] on our functionals. This framework is used for the performance analysis of partitioning heuristics for Euclidean functionals. This performance analysis is also refered to as smoothed analysis. It is a hybrid of worst-case and average-case analysis. To be able to use it, we need a smooth subadditive Euclidean functional which is pointwise close to its superadditive boundary functional. We will use MkEE^p and MkEP^p for the scheme. As MkEEm^p would behave quite different due to the multigraph structure, we have decided not to include it in our simulations. This partitioning scheme is the basis of the partitioning algorithm used in Section 6. Based on the scheme, we can also give a smoothed approximation ratio of the algorithm.

Algorithm 4.44 (Partitioning Scheme)**.**
Input: *set $V \subseteq [0,1]^d$ of n points and number of points per cell s*

1. *Partition $[0,1]^d$ into $\ell = \sqrt[d]{n/s}$ stripes of dimension $d-1$ such that each stripe contains exactly $n/\ell = (n^{d-1}s)^{1/d}$ points.*

2. *Keep partitioning each $i+1$-dimensional stripe into ℓ stripes of dimension i such that each stripe contains exactly $n/\ell^i = (n^{d-i}s^i)^{1/d}$ points. Stop at $i = 1$ so that each 2-dimensional stripe is partitioned into ℓ cells with $n/\ell^d = s$ points. In this way we end up with $\ell^d = n/s$ cells. Here we assume $s > k$.*

3. *Compute a graph achieving the optimal solution of L^p for each cell.*

4. *Join the graphs to obtain a k-edge-connected graph on V Algorithm 4.47.*

Algorithm 4.45 (Joining Graphs)**.**
Input: *set $V \subseteq [0,1]^d$ of n points, divided into n/s cells with s points, and an optimal k-edge-connected graph on each cell.*

1. *Take a random point from each cell.*

2. *Compute a graph achieving the optimal solution of L^p on these points.*

The graph we get as an output from Algorithm 4.44 is k-edge-connected because of transitivity of k-edge-connectedness (Theorem 4.2). With these algorithms we can now give running time and approximation guarantees. Depending on the way we compute the optimal solution on each cell, we need to vary s to get a polynomial running-time.

Theorem 4.46. *If the algorithm for computing an optimal solution on each cell in Algorithm 4.44 and has a running time of $O(C^{n^2})$ for some constant C, the Partitioning Scheme has a polynomial running time if we choose $s = O(\sqrt{\log n})$. The approximation guarantee then becomes $\mathrm{MkEE}^p(V) + O((n/s)^{(d-p)/d})$ for k-edge-connected graphs and $\mathrm{MkEP}^p(V) + O((n/s)^{(d-p)/d})$ for k-edge-connected power assignment graphs.*

Proof. As we have n/s cells, each with s points, the algorithm would have a total running time of $n/s \cdot O(C^{s^2}) = n/O(\sqrt{\log n}) \cdot O(C^{\log n}) = O(n^2/\sqrt{\log n})$ which is polynomial. Now for the approximation guarantee. Let $\mathrm{PSE}^p(V)$ be the cost of the k-edge-connected graph on V computed

by Algorithm 4.44 when $L^p = \text{MkEE}^p$ and $\text{PSP}^p(V)$ when $L^p = \text{MkEP}^p$. By using subadditivity we can get an upper bound on $\text{PSE}^p(V)$ (Theorem 4.3): $\text{PSE}^p(V) \leq \text{MkEE}^p(V) + C_1 +$ "cost(Algorithm 4.47)", where $C_1 = C_1(d, p)$ is a finite constant as used in Definition 2.25.

As Algorithm 4.47 creates an k-edge-connected graph on all cells, treating each cell as one point, we can upper bound the cost by $O((n/s)^{(d-p)/d})$ (using Theorem 4.16). So we get $\text{PSE}^p(V) \leq \text{MkEE}^p(V)+O((n/s)^{(d-p)/d})$. For $\text{PSP}^p(V)$ we can get a similar expression: $\text{PSP}^p(V) \leq \text{MkEP}^p(V)+O((n/s)^{(d-p)/d})$. □

If we can find an even faster algorithm to compute optimal solutions on the cells, we can adjust s a little more.

Theorem 4.47. *If the algorithm for computing an optimal solution on each cell in Algorithm 4.44 and has a running time of $O(C^n)$ for some constant C, the Partitioning Scheme has a polynomial running time if we choose $s = O(\log n)$. The approximation guarantee then becomes $\text{MkEE}^p(V) + O((n/s)^{(d-p)/d})$ for k-edge-connected graphs and $\text{MkEP}^p(V) + O((n/s)^{(d-p)/d})$ for k-edge-connected power assignment graphs.*

Proof. This proof is very similar to the proof of Theorem 4.46. □

We do not use exact algorithms with a running time guarantee of $O(C^n)$ or even $O(C^{n^2})$ as we could not find any. We can however use a mixed integer linear program (MILP) to get optimal solutions on the cells. The reason why using the partition scheme in combination with MILPs is still useful, is because MILP solvers are quite powerful. We can however not give any running time guarantee. The approximation guarantee is still $\text{MkEE}^p(V) + O((n/s)^{(d-p)/d})$ for k-edge-connected graphs and $\text{MkEP}^p(V) + O((n/s)^{(d-p)/d})$ for k-edge-connected power assignment graphs, depending on which s we choose.

The mixed integer linear programs (MILP) used in the algorithms will be described in the next section.

5 Model Formulation

Before describing the MILPs used for solving MkEE^p, MkEEm^p, MkEP^p, MkVE^p, and MkVP^p to optimality, we will first explain the intuition behind them. When we look at MkEE^p, we want to find the cheapest k-edge-connected graph $G = (V, E)$. By Definition 2.2 we need to check if G has k edge-disjoint paths between every pair of vertices $u, v \in V$. This is equivalent to finding a flow of at least k from u to v with all edge capacities set to 1. As k-edge-connectivity is transitive, we only need to check this flow from one vertex to all other vertices.

For MkEEm^p edge capacities are set to the number of duplicates of that edge. For MkEP^p it works similar, though we need to only include edges when the power is large enough. As k-vertex-connectivity is not transitive, for MkVE^p we need to check the flow between all pairs of vertices. Instead of edge capacities, we set all vertex capacities to 1. For MkVP^p we also set vertex capacities to 1, and we include edges only if the power is sufficiently large.

We can now define all sets, parameters and variables used in the mixed integer linear programs. This will be followed by the actual MILPs.

Sets:

V	set of vertices

Parameters:

c_{ij}	power-weighted distance between vertex i and j $(= \|(i,j)\|^p)$

Variables:

X_{ij}	1 if we use edge (i,j), 0 otherwise. For a multigraph, this indicates how many edges are used between vertex i and j
A_i	power assigned to vertex i
F_{iuv}	1 if edge (u,v) is part of an edge-disjoint path from vertex 1 to i
G_{ijuv}	1 if edge(u,v) is part of a vertex-disjoint path from vertex i to j

Model for MkEE^p

$$\text{Minimise} \sum_{i \in V} \sum_{j \in V, j<i} c_{ij} X_{ij} \tag{64}$$

Subject to:

$$X_{ij} = X_{ji} \qquad \forall i,j \in V,\ i > j \tag{65}$$

$$F_{iuv} \leq X_{uv} \qquad \forall i,u,v \in V \tag{66}$$

$$\sum_{u \in V} F_{iuv} = \sum_{w \in V} F_{ivw} \qquad \forall i,v \in V,\ v \notin \{1,i\} \tag{67}$$

$$\sum_{v \in V} F_{i1v} \geq k \qquad \forall i \in V \tag{68}$$

$$X_{ij} \in \{0,1\} \qquad \forall i,j \in V \tag{69}$$

$$F_{iuv} \geq 0 \qquad \forall i,u,v \in V \tag{70}$$

Model for MkEEm^p

$$\text{Minimise} \sum_{i \in V} \sum_{j \in V, j<i} c_{ij} X_{ij} \tag{71}$$

Subject to:

$$X_{ij} = X_{ji} \qquad \forall i,j \in V,\ i > j \tag{72}$$

$$F_{iuv} \leq X_{uv} \qquad \forall i,u,v \in V \tag{73}$$

$$\sum_{u \in V} F_{iuv} = \sum_{w \in V} F_{ivw} \qquad \forall i,v \in V,\ v \notin \{1,i\} \tag{74}$$

$$\sum_{v \in V} F_{i1v} \geq k \qquad \forall i \in V \tag{75}$$

$$X_{ij} \in \mathbb{N} \qquad \forall i,j \in V \tag{76}$$

$$F_{iuv} \geq 0 \qquad \forall i,u,v \in V \tag{77}$$

Model for MkEP^p

$$\text{Minimise } \sum_{i \in V} A_i \tag{78}$$

Subject to:

$$A_i \geq c_{ij} X_{ij} \qquad \forall i, j \in V \tag{79}$$

$$X_{ij} = X_{ji} \qquad \forall i, j \in V,\ i > j \tag{80}$$

$$F_{iuv} \leq X_{uv} \qquad \forall i, u, v \in V \tag{81}$$

$$\sum_{u \in V} F_{iuv} = \sum_{w \in V} F_{ivw} \qquad \forall i, v \in V,\ v \notin \{1, i\} \tag{82}$$

$$\sum_{v \in V} F_{i1v} \geq k \qquad \forall i \in V \tag{83}$$

$$X_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{84}$$

$$A_i \geq 0 \qquad \forall i \in V \tag{85}$$

$$F_{iuv} \geq 0 \qquad \forall i, u, v \in V \tag{86}$$

Model for MkVE^p

$$\text{Minimise } \sum_{i \in V} \sum_{j \in V, j < i} c_{ij} X_{ij} \tag{87}$$

Subject to:

$$X_{ij} = X_{ji} \qquad \forall i, j \in V,\ i > j \tag{88}$$

$$G_{ijuv} \leq X_{uv} \qquad \forall i, j, u, v \in V \tag{89}$$

$$\sum_{u \in V} G_{ijuv} = \sum_{w \in V} G_{ijvw} \qquad \forall i, j, v \in V,\ v \notin \{i, j\} \tag{90}$$

$$\sum_{v \in V} G_{ijiv} \geq k \qquad \forall i, j \in V \tag{91}$$

$$\sum_{u \in V} G_{ijuv} \leq 1 \qquad \forall i, j, v \in V,\ v \notin \{i, j\} \tag{92}$$

$$X_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{93}$$

$$G_{ijuv} \geq 0 \qquad \forall i, j, u, v \in V \tag{94}$$

Model for MkVP^p

$$\text{Minimise} \sum_{i \in V} A_i \tag{95}$$

Subject to:

$$A_i \geq c_{ij} X_{ij} \qquad \forall i, j \in V \tag{96}$$

$$X_{ij} = X_{ji} \qquad \forall i, j \in V,\ i > j \tag{97}$$

$$G_{ijuv} \leq X_{uv} \qquad \forall i, j, u, v \in V \tag{98}$$

$$\sum_{u \in V} G_{ijuv} = \sum_{w \in V} G_{ijvw} \qquad \forall i, j, v \in V,\ v \notin \{i, j\} \tag{99}$$

$$\sum_{v \in V} G_{ijiv} \geq k \qquad \forall i, j \in V \tag{100}$$

$$\sum_{u \in V} G_{ijuv} \leq 1 \qquad \forall i, j, v \in V,\ v \notin \{i, j\} \tag{101}$$

$$X_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{102}$$

$$A_i \geq 0 \qquad \forall i \in V \tag{103}$$

$$G_{ijuv} \geq 0 \qquad \forall i, u, v \in V \tag{104}$$

The objective functions (64), (71) and (87) minimise the sum of the length of power-weighted edges used. In (78) and (95) this objective is changed to minimising the total power used. Here the power-weighted edge length c_{ij} makes sure A_i does not have to be raised to the power p again. Constraint (65), (72), (80), (88) and (97) ensure the edges in the graphs are all symmetric. In constraint (66), (73), (81), (89) and (98) we make sure we can only send flow over an edge if we use that edge, and at the same time make sure we can only send as much flow as we have edges.

Constraint (67), (74), (82), (90) and (99) ensure flow conservation. To make sure we start with a flow of k we have constraint (68), (75), (83), (91) and (100). To account for power assignment graphs, constraint (79) and (96) together with (80) and (97), respectively, only allow usage of an edge (i, j) if the power of both vertex i and j is more than or equal to the power-weighted edge length c_{ij}. Constraint (92) and (101) set the vertex capacities to 1. Constraint (69), (84), (93) and (102) make X_{ij} a binary variable, while in constraint (76) X_{ij} is allowed to be a natural number, as it is a multigraph.

As this model is a flow model F_{iuv} and G_{ijuv} only need to be positive and integrality will be ensured by the constraints itself. This is done in constraint (70), (77), (86), (94) and (104). Lastly, constraint (85) and (103) set A_i to positive values.

6 Computational Results and Discussion

In this section we present the results of the computational tests that we have carried out based on the partitioning scheme presented in Section 4.8. There are some slight differences between the code that we used to conduct these experiments, and the scheme. We will address these differences now.

First of all, Algorithm 4.44 assumes both ℓ and n/ℓ^i are integers for $1 \leq i \leq d$. This would substantially limit the number of vertices we could use for simulations. To solve this, we chose the following approach. We assume that ℓ is integer for now (we will explain in a bit why we can make this assumption). When we make a partition of n vertices in ℓ stripes, we calculate $t = \left\lfloor \frac{n}{\ell} \right\rfloor$. Each stripe will have t vertices, possibly excluding the last stripe. The last stripe contains either t vertices (if $t = n/\ell$), or $n - t \cdot (\ell - 1)$ vertices otherwise. In other words, we rather have too many vertices in a stripe, than too few.

The second difference is that we not neccesarilly create ℓ partitions in each dimension. As mentioned before, $\ell = \sqrt[d]{n/\log n}$ is not always integer. Also, part of the experiment is to see the

influence of the number of cells on the solution value. So instead of using ℓ, we give our own input on how many stripes are created in each dimension. In this way, we also ensure that number of stripes is integer at all times as used when calculating t.

An issue that could arise by choosing the number of partitions ourselves is that the number of vertices in each cell is smaller than $k+1$, or that the number of cells created is less than $k+1$. We recall that k is a constant, and n will be large compared to k, but for our computational experiments this might occur. The first of these options can be avoided by carefully choosing the number our partitions. So for all our simulations we make sure that the number of vertices per cell is at least $k+1$. The second option however cannot always be prevented. We solve this by adjusting Algorithm 4.47. Instead of creating a k-edge-connected (power assignment) graph on the cells, we lace cells together. We do this by creating a minimal length matching of k vertices from two cells. It is easy to check that this yields a k-edge-connected (power assignment) graph as well. In order for the distance between two laced cells not to be too large, we need a clever way of the cells following up one another. This is done by creating a snakelike succession. This is illustrated in Figure 5 in two dimensions. The difficulty with this structure is that the numbering of the cells is done stripe by stripe (as shown in Figure 5), contrary to what the snakelike succession will be. The exact formulation can be found in the code in Appendix A.3 and A.4. We did compare if it was cheaper to also use the snakelike succession when the number of cells is at least $k+1$, but it that was not the case.

The last difference is again with Algorithm 4.47. When $L^p = \text{MkEP}^p$, instead of taking a random vertex from each cell, we take the vertex with the largest power. The idea behind this is that this vertex might already reach some other cells, therefore it might lower the cost of joining graphs compared to taking a random vertex.

All simulations have been carried out on randomly generated vertex sets. For each problem of size $n = |V|$, we created 30 instances with n points chosen from a uniform distribution on $[0,1]^d$. We allow a 10 hour computation limit and a 6GB memory limit for each instance. The solver we use is ILOG CPLEX 12.6.1 with the Python 2.7 programming language using PuLP modeller 1.6.1 [27]. The computational experiments are carried out on an Intel Core i7-4790 machine with 8 GB of RAM.

All simulations have been done with $d = 2$, though the algorithms also work for other d. To get an idea of what the influence of p is on the cost, we did simulations with $p = 1$ and $p = 2$. We also tried two options for k, namely $k = 3$ and $k = 5$. For the number of cells by partitioning in two dimensions we chose how many partitions we wanted in each dimension. The maximum number of partition in one dimension is 4. This gives us the following options, where the numbers in brackets denote how many partitions we used in each dimension: 1 (optimal), 2 (2 and 1), 4 (2 and 2), 6 (3 and 2), 9 (3 and 3), 12 (4 and 3) and 16 (4 and 4). We have checked if noticeable differences occurred if we changed the order of partitioning (so instead of 3 and 2, we used 2 and 3), but the order hardly changed the solution value or time.

For MkEE^p we can solve cases of up to 40 vertices to optimality. This is why we decided to do our tests with $n = 30$ and $n = 40$. We also tried larger sets of vertices using only the partitioning heuristic (up to 100 vertices) for the case $p = 1$ and $k = 3$. As we take the average over 30 instances for each problem, this amounts to a total of 1410 simulations of MkEE^p. Here we take into account that we have a maximum number of cells to make sure the number of vertices in each cell is at least $k+1$. So for example, with $n = 30$ and $k = 3$ we can only use 1, 2, 4 and 6 cells. The results are displayed in Table 1

For MkEP^p the solution times drastically increased and we were only able to solve (some) cases of up to 20 vertices to optimality. Here we decided to do tests for $n = 20$, $n = 30$ and $n = 40$ if they were admissible by solution time and number of vertices per cell. We also tried $n = 60$ for the case $p = 1$ and $k = 3$. Again taking the average over 30 instances per problem, this amounts to 990 simulations of MkEP^p. The results are displayed in Table 2. Not all cases with 1 cell were solved to optimality due to memory or computation time limits. For these cases the number of instances not solved to optimality are shown behind the solution time in brackets. The complete code can be found in Appendix A.

In Figure 6 the solution times and values of MkEE^p are shown with their standard deviation

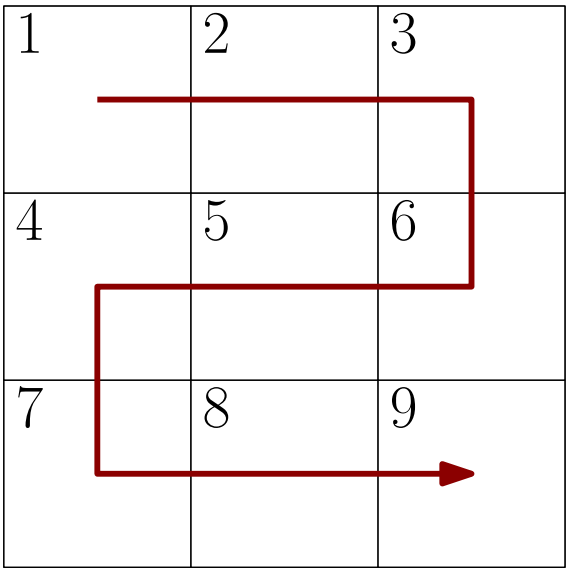


Figure 5: *Example of the snakelike succession used in our partitioning algorithm in two dimensions. The cells are numbered 1 to 9 as shown, but we want to make sure cells are close to each other when we lace them together. This is why we want successive cells to be neighbours. By creating a snakelike pattern over the cells, we make sure this happens. The order for lacing then becomes 1 2 3 6 5 4 7 8 9. This pattern is generalised for higher dimensions in the code in Appendix A.3 and A.4.*

for $n = 40$, $p = 1$ and $k = 3$. As expected the average solution value increases if the number of cells becomes larger. It is interesting to see that the biggest change in solution value happens between 2 and 4 cells. This could be explained as we have to create a complete graph on a random point in each of the 4 cells (as $k = 3$), instead of a minimal length matching of k vertices from two cells. Lacing has a much lower cost, as we can pick the k cheapest edges instead of having random vertices to create a k-edge-connected graph on. Another striking observation is that the average solution time grows enormously between using 2 cells, and solving the instances to optimality.

The same observation can be done in Figure 7, where the solution times and values of MkEP^p are shown with the standard deviation for $n = 20$, $p = 1$ and $k = 3$. This indicates that $n = 40$ for MkEE^p and $n = 20$ for MkEP^p really are a critical number of vertices. In Figure 7 we only have 1, 2 and 4 cells, as $n = 20$ and $k = 3$. Again we can see that the increase from 2 to 4 cells is quite steep.

Figure 8 shows the solution times and values of MkEE^p with standard deviation for both $p = 1$ and $p = 2$. These two are shown for $n = 40$ and $k = 3$. To make the figure more clear, the solution time of the optimal solution is left out as well as the standard deviation of the solution times. The shape of the average solution value is similar for both cases, though we clearly see the solution value for $p = 2$ is lower. This makes sense as $|e| > |e|^2$ for edges e with Euclidean edge length $0 < |e| < 1$. Since $V \subset [0, 1]^2$, most edges satisfy $|e| < 1$. It is interesting that their solution times are almost identical. This could be explained by not having p appear anywhere in the MILP, but only affecting the cost of edges.

In Figure 9 the solution times and values of MkEE^p are shown with their standard deviation for both $k = 3$ and $k = 5$. These two are shown for $n = 40$ and $p = 1$. To make the figure more clear, the solution time of the optimal solution is left out as well as the standard deviation of the solution times. The number of cells only is 6 here at maximum, as for $k = 5$ and $n = 40$ we cannot use 9 cells. This would create cells with less than 6 vertices, so we would need a minimum of 54 vertices to use 9 cells for $k = 5$. We see that for $k = 5$ the average solution value grows even stronger between 4 and 6 cells compared to the growth between 2 and 4 cells. This might

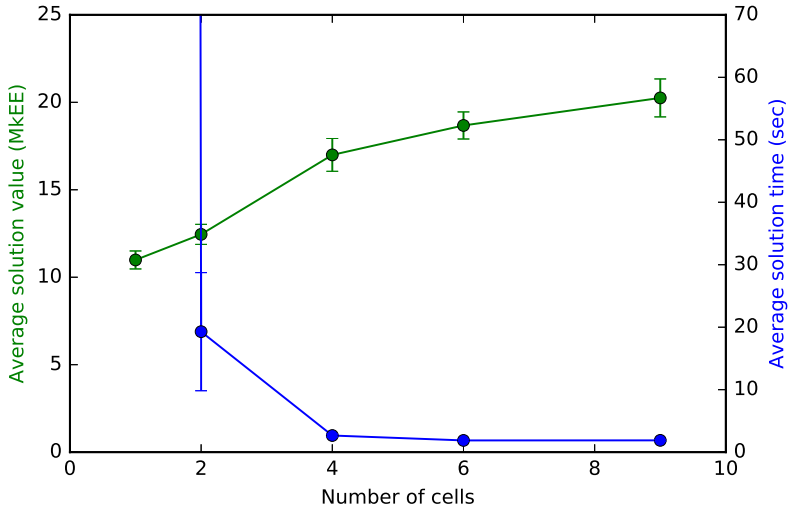


Figure 6: *The average solution values and times for* MkEE^p *with* $p = 1$, $d = 2$, $k = 3$ *and* $n = 40$ *for different partitions. The error bars on each point indicate the standard deviation.*

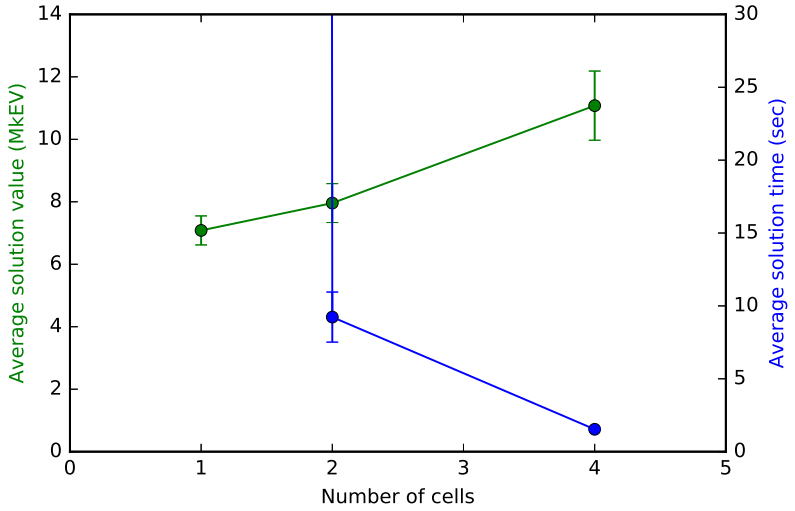


Figure 7: *The average solution values and times for* MkEP^p *with* $p = 1$, $d = 2$, $k = 3$ *and* $n = 20$ *for different partitions. The error bars on each point indicate the standard deviation.*

be explained by having to create a complete graph on a random point in each of the 6 cells (as $k = 5$), instead of 3 minimal length matching of k vertices from two cells by lacing them together. Furthermore, the solution time for 2 cells is a lot lower for $k = 5$ than for $k = 3$. This also holds for 1 cell, which can be seen in Table 1. This could be due to the graph needing more edges to be 5-edge-connected than 3-edge-connected. There are less choices between possible short edges used in the graph, as they are all needed in the graph, reducing the solution time.

In Figure 10 the solution times and values of MkEE^p are shown with their standard deviation for n ranging from 30 to 100. All cases have $p = 1$ and $k = 3$, and are partitioned in 4 cells. We see that the solution values grow linear, or even sublinear. This makes sense because as we add more vertices, all vertices are relatively closer together. In this way, the solution value would not

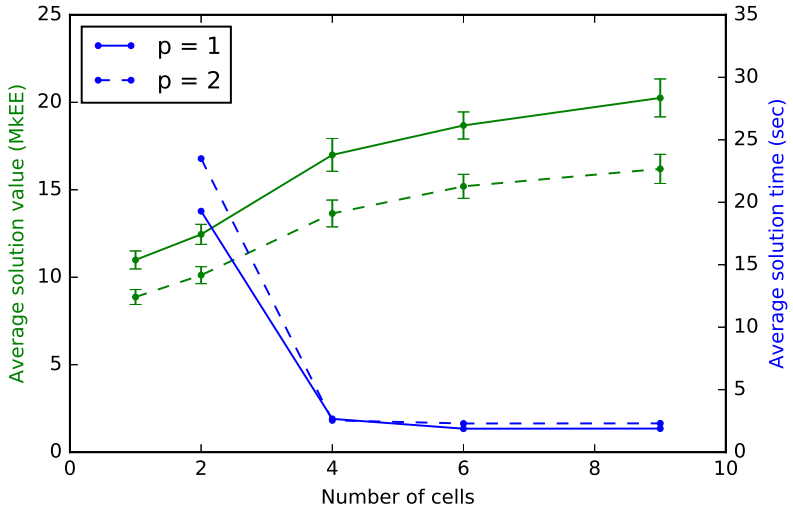


Figure 8: *Comparison of $p = 1$ and $p = 2$ for MkEE^p with $d = 2$, $k = 3$ and $n = 40$ for different partitions. The error bars on each point indicate the standard deviation. The case for $p = 1$ is shown in the uninterrupted line, and $p = 2$ in the interrupted line.*

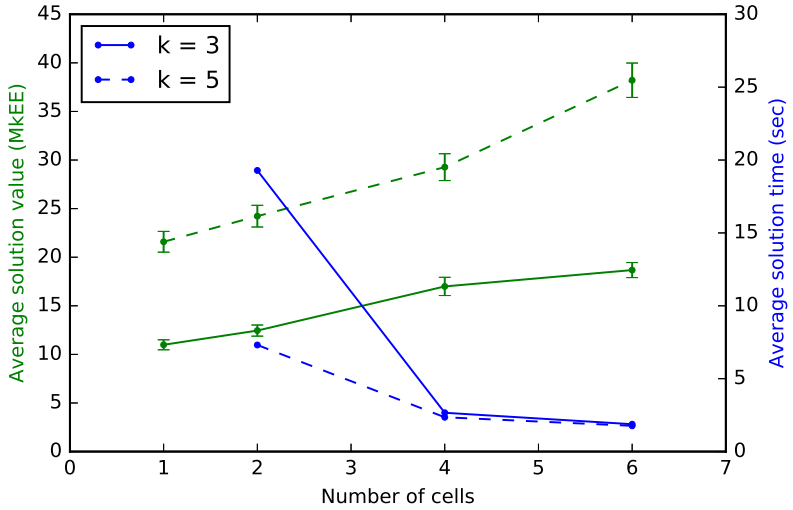


Figure 9: *Comparison of $k = 3$ and $k = 5$ for MkEE^p with $p = 1$, $d = 2$ and $n = 40$ for different partitions. The error bars on each point indicate the standard deviation. The case for $k = 3$ is shown in the uninterrupted line, and $p = 5$ in the interrupted line.*

increase too much eventually. In the solution times however, we see a steep increase as we add more vertices. This could well indicate a running time of exponential order.

As mentioned before Table 1 and 2 show the results of all simulations done. An interesting observation is that partitioning 60 vertices in 2 cells and solving those, is faster than twice the solution time of 30 vertices. This could mean that the structure of 60 vertices split in 2 is different as they are closer together in 1 dimension.

If we compare the solutions of the partition algorithm to the optimal solutions of MkEE^p, we get that for $n = 30$ partitioning in 2 cells is gives a solution value approximately 18% higher for

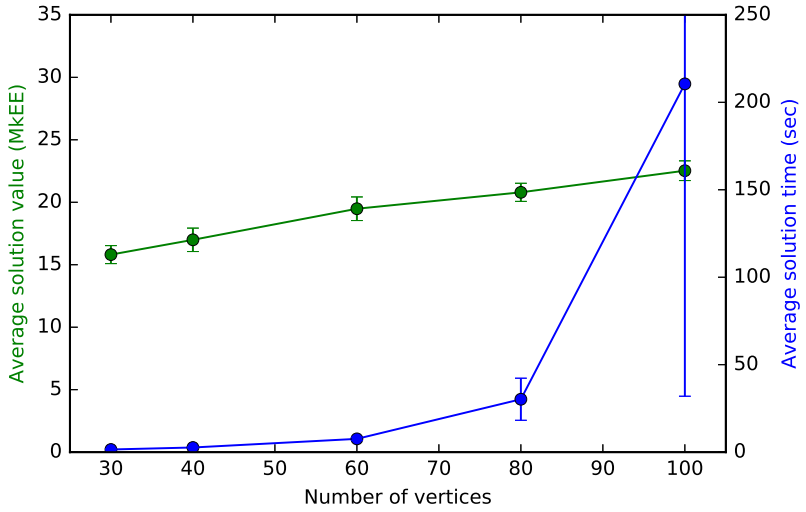


Figure 10: *The average solution values and times for* MkEE^p *with* $p = 1$, $d = 2$ *and* $k = 3$, *all for partitioning in 4 cells and n ranging from 30 to 100. The error bars on each point indicate the standard deviation.*

both $p = 1$ and $p = 2$, and both $k = 3$ and $k = 5$. Partitioning in 4 cells gives a solution value 63% higher when $k = 3$ and 42% higher for $k = 5$. For $n = 40$ this changes to 14 % for 2 cells in case $k = 3$ and 12% when $k = 5$. For 4 cells this becomes 54% when $k = 3$ and 35% when $k = 5$. Interestingly, p does not seem to have a substantial influence on these increases. For higher k, it seems like partitioning in more cells has less of an influence than it has for lower k. The increase is also bigger for smaller n. This makes sense as a partition of a smaller set has a relatively bigger influence.

7 Conclusions

In this thesis we have looked at fault tolerant networks in terms of connectivity. There has been a lot of research on this topic, since it has numerous applications. We studied both standard and wireless networks, looking at k-edge-connectedness and k-vertex-connectedness. The problem of finding a shortest k-edge-connected or k-vertex-connected graph on a given set of vertices in terms of summed power-weighted edge lengths can be seen as a functional. The same holds for finding a minimal power assignment resulting in a k-edge-connected or k-vertex-connected graph. We also looked at finding a shortest k-edge-connected multigraphs as a functional. We defined these functionals as MkEE^p, MkEEm^p, MkEP^p, MkVE^p, and MkVP^p. We wanted the functionals to have certain properties in order to do smoothed analysis on the performance of partitioning algorithms for the functionals.

First we proved subadditivity for MkEE^p, MkEEm^p, MkEP^p, MkVE^p, and MkVP^p. As these functionals were not superadditive, we used their boundary functionals and proved superadditivity for MkEE^p_B, MkEEm^p_B, MkEP^p_B, MkVE^p_B, and MkVP^p_B. To show that the original functionals had a form of near additivity, we proved pointwise closeness of MkEE^p, MkEEm^p, MkEP^p, MkVE^p, and MkVP^p to their respective boundary functional. For other results we also needed smoothness, and we were able to prove this for MkEE^p, MkEEm^p, and MkEP^p, and their boundary functionals.

With these properties we have shown a bound on the longest edge with high probability for MkEE^p. We also used the properties to show complete convergence and some limit theories for MkEE^p, MkEEm^p, and MkEP^p. Bounds on the rates of convergence of means of MkEE^p, MkEEm^p, MkEP^p, MkVE^p, and MkVP^p have been obtained. Finally we presented two umbrella

MkEE^p				Number of cells						
				1	2	4	6	9	12	16
$n = 30$	$p = 1$	$k = 3$	time (s)	655.95	4.23	1.52	1.79	-	-	-
			value	9.63	11.24	15.81	17.94	-	-	-
		$k = 5$	time (s)	26.49	3.37	1.85	-	-	-	-
			value	19.17	22.56	27.49	-	-	-	-
	$p = 2$	$k = 3$	time (s)	640.95	4.42	1.90	1.73	-	-	-
			value	7.77	9.16	12.58	14.47	-	-	-
		$k = 5$	time (s)	22.64	3.44	1.86	-	-	-	-
			value	15.40	18.17	21.76	-	-	-	-
$n = 40$	$p = 1$	$k = 3$	time (s)	5332.46	19.29	2.66	1.88	1.89	-	-
			value	10.99	12.45	16.99	18.68	20.25	-	-
		$k = 5$	time (s)	310.81	7.31	2.35	1.77	-	-	-
			value	21.59	24.22	29.27	38.21	-	-	-
	$p = 2$	$k = 3$	time (s)	6554.43	23.50	2.55	2.30	2.30	-	-
			value	8.87	10.11	13.65	15.20	16.19	-	-
		$k = 5$	time (s)	315.47	7.78	2.29	1.76	-	-	-
			value	17.37	19.51	23.38	30.45	-	-	-
$n = 60$	$p = 1$	$k = 3$	time (s)	-	776.15	7.59	3.67	2.92	3.45	-
			value	-	14.30	19.48	20.88	22.63	25.32	-
$n = 80$	$p = 1$	$k = 3$	time (s)	-	-	30.25	7.90	5.47	4.56	6.33
			value	-	-	20.79	22.59	24.60	26.15	28.80
$n = 100$	$p = 1$	$k = 3$	time (s)	-	-	210.53	18.94	8.00	6.86	6.33
			value	-	-	22.53	24.05	26.25	28.17	29.74

Table 1: *Solution values and times of* MkEE^p *averaged over 30 instances for different combinations of p, d, k, n and number of cells.*

theorems for MkEE^p, MkEEm^p, and MkEP^p as extensions on the limit theorem.

We used smoothed analysis to prove that the partitioning algorithms for MkEE^p obtains a solution with value MkEE$^p(V) + O((n/\log n)^{(d-p)/d})$, where $|V| = n$. For MkEP^p it becomes MkEP$^p(V) + O((n/\log n)^{(d-p)/d})$. For acquiring a solution on each cell we used the exact solution of mixed integer linear program (MILP). These MILPs are based on the flow from one vertex to another to show there are k edge-disjoint paths, or k vertex-disjoint paths. The cells are then joined by treating each cell as one point and making a k-edge-connected (power assignment) graph on it. Although smooth analysis is done only for MkEE^p, MkEEm^p, and MkEP^p, the MILPs are presented for all functionals.

Based on the partitioning algorithm we made a heuristic in which we could specify how many cells we get. This heuristic can be used for arbitrary p, d and k, though we have done simulations for $p = 1$ and 2, $d = 2$, $k = 3$ and 5 for both MkEE^p and MkEP^p. We varied the number of cells between 1 and 16, and the number of vertices between 20 and 100. Here we take into account that we have a maximum number of cells to make sure the number of vertices in each cell is at least $k + 1$. For each problem of size $n = |V|$, we created 30 instances with n points chosen from a uniform distribution on $[0, 1]^d$. In total we simulated and solved 2400 instances. For MkEE^p we could solve instances of up to 40 vertices to optimality and for MkEP^p only up to 20 vertices.

For $k = 3$ we saw that the solution value increased the most if we changed from partitioning into 2, to partitioning into 4 cells. For $k = 5$ this change was from 4 to 6 cells. Solution values of $p = 1$ were higher than of $p = 2$. Those of $k = 5$ were logically also higher than those of $k = 3$, though the solution time for $k = 5$ was noticeably lower. When comparing the partitioning in 4 cells for a different number of vertices, we could see the solution values were only growing (sub)linear. The running time, however, showed signs of being of exponential order, as expected. An interesting observation we did was that when we doubled the number of vertices and partitioned them in 2 cells, the solution time was less than doubled. This could mean that the structure of the vertices

MkEP^p				Number of cells					
				1	2	4	6	9	12
$n = 20$	$p = 1$	$k = 3$	time (s)	9091.83 (13)	9.23	1.54	-	-	-
			value	7.08	7.96	11.08	-	-	-
		$k = 5$	time (s)	15514.51 (16)	1.53	-	-	-	-
			value	9.62	11.29	-	-	-	-
	$p = 2$	$k = 3$	time (s)	11698.57	2.07	1.28	-	-	-
			value	5.68	6.43	8.66	-	-	-
		$k = 5$	time (s)	-	1.56	-	-	-	-
			value	-	9.01	-	-	-	-
$n = 30$	$p = 1$	$k = 3$	time (s)	-	129.62	1.92	2.31	-	-
			value	-	9.52	12.83	13.97	-	-
		$k = 5$	time (s)	-	146.08	2.56	-	-	-
			value	-	13.00	14.82	-	-	-
	$p = 2$	$k = 3$	time (s)	-	140.10	2.04	2.32	-	-
			value	-	7.63	9.88	11.21	-	-
		$k = 5$	time (s)	-	142.13	2.07	-	-	-
			value	-	10.29	11.53	-	-	-
$n = 40$	$p = 1$	$k = 3$	time (s)	-	-	3.93	2.87	2.64	-
			value	-	-	13.95	14.95	15.56	-
		$k = 5$	time (s)	-	-	3.51	2.28	-	-
			value	-	-	15.76	21.63	-	-
	$p = 2$	$k = 3$	time (s)	-	-	3.89	2.86	2.75	-
			value	-	-	10.96	12.08	12.39	-
		$k = 5$	time (s)	-	-	3.89	2.04	-	-
			value	-	-	12.48	16.91	-	-
$n = 60$	$p = 1$	$k = 3$	time (s)	-	-	294.78	5.96	4.30	5.49
			value	-	-	15.76	16.98	17.74	19.40

Table 2: *Solution values and times of* MkEP^p *averaged over 30 instances for different combinations of p, d, k, n and number of cells. The numbers in brackets indicate how many of the 30 instances could not be solved to optimality within the given computation time or memory limit.*

split in 2 is different as they are closer together in 1 dimension, speeding up computations.

When we checked how much worse our partitioning algorithm performs compared to the optimal solution, we see that it performs worse on instances with a lower number of vertices, and with lower k. p does not seem to have an influence on this. For example, we got that for $n = 40$ partitioning in 2 cells is gives a solution value approximately 14% higher with $k = 3$ for both $p = 1$ and $p = 2$, and 12% when $k = 5$. Partitioning in 4 cells gives a solution value 54% higher when $k = 3$ and 35% higher for $k = 5$.

8 Future Work

One prominent problem we encountered is that we could not show smoothness for MkVE^p or MkVP^p. The main issue with this is that k-vertex-connectivity is not transitive as k-edge-connectivity is. It would be worthwhile to find a way to work around this, as we already have subadditivity and pointwise closeness to its superadditive boundary functionals. This means we could extend all results obtained for MkEE^p, MkEEm^p, and MkEP^p directly to MkVE^p or MkVP^p. This would also give the option to use a partitioning algorithm on these functionals and get a performance approximation with smoothed analysis.

Another relevant extension would be to consider the case $p \geq d$. This would further generalise our results. We would probably need to depend on closeness and smoothness in mean as pointwise

closeness and smoothness are not guaranteed to hold for $p > d$. We could perhaps use the approach in De Graaf and Manthey [12] to obtain similar results.

Our smoothed analysis approach could be extended to be able to get stricter approximation ratios. As in De Graaf and Manthey [6] we could also analyse the running time of the partitioning algorithm. This would also help in finding an optimal trade-off between approximation factor and running time, which we could use in our partitioning algorithm.

We are aware that our analysis of our algorithm is a bit frugal as we did not compare it to other heuristics. This would be useful research topic. Another thing is that our algorithm could have further improvements, especially in joining all cells together. At this point we take a random point in each cell, but it might be smarter to base this choice on some other attributes the vertices have.

9 Acknowledgement

I would like to thank my supervisor Bodo Manthey for his useful feedback and fruitful meetings.

I also want to note that this thesis is thanks to:

- 221 cups of coffee
- 31 cups of chocolate milk
- 4 pencils
- 307 PhD Comics read
- 11 calls with ICT Service
- Not being able to spell functionals without fun

References

[1] Ernst Althaus, Gruia Calinescu, Ion I Mandoiu, Sushil Prasad, Nickolay Tchervenski, and Alexander Zelikovsky, *Power efficient range assignment for symmetric connectivity in static ad hoc wireless networks*, Wireless Networks **12** (2006), no. 3, 287–299.

[2] Mohsen Bahramgiri, Mohammadtaghi Hajiaghayi, and Vahab S Mirrokni, *Fault-tolerant and 3-dimensional distributed topology control algorithms in wireless multi-hop networks*, Wireless Networks **12** (2006), no. 2, 179–188.

[3] Fatiha Bendali, I Diarrassouba, Ali Ridha Mahjoub, M Didi Biha, and Jean Mailfert, *A branch-and-cut algorithm for the k-edge connected subgraph problem*, Networks **55** (2010), no. 1, 13–32.

[4] Christian Bettstetter, *On the minimum node degree and connectivity of a wireless multihop network*, Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing, ACM, 2002, pp. 80–91.

[5] M Didi Biha and Ali Rhida Mahjoub, *k-edge connected polyhedra on series-parallel graphs*, Operations research letters **19** (1996), no. 2, 71–78.

[6] Markus Bläser, Bodo Manthey, and BV Raghavendra Rao, *Smoothed analysis of partitioning algorithms for euclidean functionals*, Algorithmica **66** (2013), no. 2, 397–418.

[7] Gruia Calinescu and Peng-Jun Wan, *Range assignment for high connectivity in wireless ad hoc networks*, International Conference on Ad-Hoc Networks and Wireless, Springer, 2003, pp. 235–246.

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

[9] Sunil Chopra, *The k-edge-connected spanning subgraph polyhedron*, SIAM Journal on Discrete Mathematics **7** (1994), no. 2, 245–259.

[10] Andrea EF Clementi, Paolo Penna, and Riccardo Silvestri, *On the power assignment problem in radio networks*, Mobile Networks and Applications **9** (2004), no. 2, 125–140.

[11] Kamiel Cornelissen, Ruben Hoeksma, Bodo Manthey, NS Narayanaswamy, and CS Rahul, *Approximability of connected factors*, International Workshop on Approximation and Online Algorithms, Springer, 2013, pp. 120–131.

[12] Maurits de Graaf and Bodo Manthey, *Probabilistic analysis of power assignments*, International Symposium on Mathematical Foundations of Computer Science, Springer, 2014, pp. 201–212.

[13] Ibrahima Diarrassouba, Hakan Kutucu, and A Ridha Mahjoub, *Two node-disjoint hop-constrained survivable network design and polyhedra*, Networks **67** (2016), no. 4, 316–337.

[14] Ibrahima Diarrassouba, Meriem Mahjoub, and A Ridha Mahjoub, *The k-node-connected subgraph problem: Facets and branch-and-cut*, 2016 12th International Conference on the Design of Reliable Communication Networks (DRCN), IEEE, 2016, pp. 1–8.

[15] Michael R Gary and David S Johnson, *Computers and intractability: A guide to the theory of np-completeness*, 1979.

[16] Martin Grötschel, Clyde Monma, and Mechthild Stoer, *Polyhedral approaches to network survivability*, Inst. für Mathematik, 1990.

[17] Martin Grötschel and Clyde L Monma, *Integer polyhedra arising from certain network design problems with connectivity constraints*, SIAM Journal on Discrete Mathematics **3** (1990), no. 4, 502–523.

[18] Martin Grötschel, Clyde L Monma, and Mechthild Stoer, *Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints*, Operations Research **40** (1992), no. 2, 309–330.

[19] ———, *Polyhedral and computational investigations for designing communication networks with high survivability requirements*, Operations Research **43** (1995), no. 6, 1012–1024.

[20] Ning Li and Jennifer C Hou, *Flss: a fault-tolerant topology control algorithm for wireless networks*, Proceedings of the 10th annual international conference on Mobile computing and networking, ACM, 2004, pp. 275–286.

[21] Errol L Lloyd, Rui Liu, Madhav V Marathe, Ram Ramanathan, and SS Ravi, *Algorithmic aspects of topology control problems for ad hoc networks*, Mobile Networks and applications **10** (2005), no. 1-2, 19–34.

[22] Ali Ridha Mahjoub, *Two-edge connected spanning subgraphs and polyhedra*, Mathematical Programming **64** (1994), no. 1-3, 199–208.

[23] Ali Ridha Mahjoub and Charles Nocq, *On the linear relaxation of the 2-node connected subgraph polytope*, Discrete applied mathematics **95** (1999), no. 1, 389–416.

[24] Bodo Manthey and Heiko Röglin, *Smoothed analysis: Analysis of algorithms beyond worst case*, it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik **53** (2011), no. 6, 280–286.

[25] Bodo Manthey and Marten Waanders, *Approximation algorithms for k-connected graph factors*, International Workshop on Approximation and Online Algorithms, Springer, 2015, pp. 1–12.

[26] Karl Menger, *Zur allgemeinen kurventheorie*, Fundamenta Mathematicae **10** (1927), no. 1, 96–115.

[27] Stuart Mitchell, Michael OSullivan, and Iain Dunning, *Pulp: A linear programming toolkit for python*, 2011.

[28] Roberto Montemanni and Luca Maria Gambardella, *Exact algorithms for the minimum power symmetric connectivity problem in wireless networks*, Computers & Operations Research **32** (2005), no. 11, 2891–2904.

[29] Ram Ramanathan and Regina Rosales-Hain, *Topology control of multihop wireless networks using transmit power adjustment*, INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol. 2, IEEE, 2000, pp. 404–413.

[30] Charles Redmond, *Boundary rooted graphs and euclidean matching algorithms*, Ph.D. thesis, Bethlehem, PA, USA, 1993.

[31] Wansoo T Rhee, *A matching problem and subadditive euclidean functionals*, The Annals of Applied Probability **3** (1993), no. 3, 794–801.

[32] Paolo Santi, Douglas M Blough, and Feodor Vainstein, *A probabilistic analysis for the range assignment problem in ad hoc networks*, Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing, ACM, 2001, pp. 212–220.

[33] Daniel A Spielman and Shang-Hua Teng, *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time*, Journal of the ACM (JACM) **51** (2004), no. 3, 385–463.

[34] ———, *Smoothed analysis: an attempt to explain the behavior of algorithms in practice*, Communications of the ACM **52** (2009), no. 10, 76–84.

[35] J Michael Steele, *Subadditive euclidean functionals and nonlinear growth in geometric probability*, The Annals of Probability **9** (1981), no. 3, 365–376.

[36] Hassler Whitney, *Congruent graphs and the connectivity of graphs*, American Journal of Mathematics **54** (1932), no. 1, 150–168.

[37] Joseph E Yukich, *Probability theory of classical euclidean optimization problems*, Lecture Notes in Mathematics, vol. 1675, Springer-Verlag, Berlin, 1998.

A Python Code

A.1 Code for solving MkEE^p to optimality

```
"""
Gives the optimal solutions of specified number of instances of MkEE for
    specified parameters
"""

import pulp
import cplex
import numpy as np
```

```
import time as tm

#Parameters
k=3
n=40
d=2
p=1
num_seeds=30 #number of instances
sol_times=[]
sol_values=[]
sol_status=[]

for sd in range(0,num_seeds):
    np.random.seed(sd) #To get the same solutions if we rerun it
    vertices=range(1,n+1)
    locations=dict() #Contains coordinates of all vertices
    for l in range(d):
        locations[l]=dict()
        for i in vertices:
            locations[l][i]=np.random.random()
    distance=dict() #Power-weighted distance between vertices
    for i in vertices:
        distance[i]=dict()
        for j in vertices:
            if i==j:
                distance[i][j]=0
            else:
                distance_temp_temp=0
                for l in range(d):
                    distance_temp_temp+=(abs(locations[l][i]-locations[l][j]))**p
                distance[i][j]=distance_temp_temp**(1./p)
    n=len(vertices)

    start=tm.clock()

    MkEE=pulp.LpProblem('Basic Fragmentation Model',pulp.LpMinimize)

    #Variables
    X={}
    for i in vertices:
        for j in vertices:
            X[i,j]=pulp.LpVariable("X_%s_%s"
                %(i,j),lowBound=0,upBound=1,cat=pulp.LpInteger)

    F={}
    for i in vertices:
        for u in vertices:
            for v in vertices:
                F[i,u,v]=pulp.LpVariable('F_%s_%s_%s' %(i,u,v),lowBound=0)

    #Objective function
    MkEE += sum( [sum([distance[i][j]*X[i,j] for j in vertices if j>i]) for i
        in vertices])

    #Constraints
    for i in vertices:
        for j in vertices:
            if j>i:
```

```
64                  MkEE += X[i,j]==X[j,i]
65
66      for i in vertices:
67          for u in vertices:
68              for v in vertices:
69                  MkEE += F[i,u,v]<=X[u,v]
70
71      for i in vertices: #extra constraint to make sure it does not send flow
            to itself, as the distance[i][i]=0
72          MkEE += X[i,i]==0
73          for u in vertices:
74              MkEE += F[i,u,u]==0
75
76      for i in vertices:
77          for v in vertices:
78              if (v != i and v != vertices[0] and i!=vertices[0]):
79                  MkEE += sum( F[i,u,v] for u in vertices) == sum( F[i,v,w] for
                        w in vertices)
80
81      for i in vertices:
82          MkEE += sum( F[i,vertices[0],v] for v in vertices) - sum(
                F[i,u,vertices[0]] for u in vertices)>= k
83
84      MkEE.writeLP("MkEE_OPT_seed_%s_n_%s_k_%s_d_%s_p_%s_lp.lp" %(sd,n,k,d,p))
            #Write the LP file to solve
85      problem = cplex.Cplex("MkEE_OPT_seed_%s_n_%s_k_%s_d_%s_p_%s_lp.lp"
            %(sd,n,k,d,p)) #Loads the LP
86      problem.parameters.timelimit.set(36000) #Sets computation time limit in
            seconds
87      problem.parameters.mip.limits.treememory.set(6000) #Sets the memory limit
            in MB
88
89      try:
90          problem.solve()
91      except CplexSolverError as exc: #Catch the error if CPLEX cannot solve
            it, ignore code analysis
92          print 'Cplex encountered an error with seed %s, continue to next.'
                %(sd)
93          continue
94
95      #Even is the LP is not solved to optimality (but f.e. with gap) we want
            to get the answer
96      if problem.solution.get_status() in
            [1,6,10,11,12,13,14,15,18,19,21,22,23,24,25,101,102,104,105,107,111,120,
            121,122,124,125,129,130,131]:
97          sol_times.append(tm.clock()-start)
98          sol_values.append(problem.solution.get_objective_value())
99          sol_status.append(problem.solution.get_status())
100     else:
101         sol_times.append('NaN')
102         sol_values.append('NaN')
103         sol_status.append(problem.solution.get_status())
104     print 'Total Cost = %s' %(problem.solution.get_objective_value())
105     print 'Total solution time = %s' %(tm.clock()-start)
```

A.2 Code for solving MkEP^p to optimality

```
1   """
```

```
2  Gives the optimal solutions of specified number of instances of MkEV for
       specified parameters
3  """
4
5  import pulp
6  import cplex
7  import numpy as np
8  import time as tm
9
10 #Parameters
11 k=3
12 n=40
13 d=2
14 p=1
15 num_seeds=30 #number of instances
16 sol_times=[]
17 sol_values=[]
18 sol_status=[]
19
20 for sd in range(0,num_seeds):
21     np.random.seed(sd) #To get the same solutions if we rerun it
22     vertices=range(1,n+1)
23     locations=dict() #Contains coordinates of all vertices
24     for l in range(d):
25         locations[l]=dict()
26         for i in vertices:
27             locations[l][i]=np.random.random()
28     distance=dict() #Power-weighted distance between vertices
29     for i in vertices:
30         distance[i]=dict()
31         for j in vertices:
32             if i==j:
33                 distance[i][j]=0
34             else:
35                 distance_temp_temp=0
36                 for l in range(d):
37                     distance_temp_temp+=(abs(locations[l][i]-locations[l][j]))**p
38                 distance[i][j]=distance_temp_temp**(1./p)
39     n=len(vertices)
40
41     start=tm.clock()
42
43     MkEV=pulp.LpProblem('Basic Fragmentation Model',pulp.LpMinimize)
44
45     #Variables
46     X={}
47     for i in vertices:
48         for j in vertices:
49             X[i,j]=pulp.LpVariable("X_%s_%s"
                   %(i,j),lowBound=0,upBound=1,cat=pulp.LpInteger)
50
51     F={}
52     for i in vertices:
53         for u in vertices:
54             for v in vertices:
55                 F[i,u,v]=pulp.LpVariable('F_%s_%s_%s' %(i,u,v),lowBound=0)
56
57     A={}
```

```
58      for i in vertices:
59          A[i]=pulp.LpVariable('A_%s' %(i),lowBound=0)
60
61      #Objective function
62      MkEV += sum( [A[i] for i in vertices])
63
64      #Constraints
65      for i in vertices:
66          for j in vertices:
67              MkEV += A[i] >= distance[i][j]*X[i,j]
68
69      for i in vertices:
70          for j in vertices:
71              if j>i:
72                  MkEV += X[i,j]==X[j,i]
73
74      for i in vertices:
75          for u in vertices:
76              for v in vertices:
77                  MkEV += F[i,u,v]<=X[u,v]
78
79      for i in vertices: #extra constraint to make sure it does not send flow
            to itself, as the distance[i][i]=0
80          MkEV += X[i,i]==0
81          for u in vertices:
82              MkEV += F[i,u,u]==0
83
84      for i in vertices:
85          for v in vertices:
86              if (v != i and v != vertices[0] and i!=vertices[0]):
87                  MkEV += sum( F[i,u,v] for u in vertices) == sum( F[i,v,w] for
                        w in vertices)
88
89      for i in vertices:
90          MkEV += sum( F[i,vertices[0],v] for v in vertices) - sum(
                F[i,u,vertices[0]] for u in vertices)>= k
91
92      MkEV.writeLP("MkEV_OPT_seed_%s_n_%s_k_%s_d_%s_p_%s_lp.lp" %(sd,n,k,d,p))
            #Write the LP file to solve
93      problem = cplex.Cplex("MkEV_OPT_seed_%s_n_%s_k_%s_d_%s_p_%s_lp.lp"
            %(sd,n,k,d,p)) #Loads the LP
94      problem.parameters.timelimit.set(36000) #Sets computation time limit in
            seconds
95      problem.parameters.mip.limits.treememory.set(6000) #Sets the memory limit
            in MB
96
97      try:
98          problem.solve()
99      except CplexSolverError as exc: #Catch the error if CPLEX cannot solve
            it, ignore code analysis
100         print 'Cplex encountered an error with seed %s, continue to next.'
                %(sd)
101         continue
102
103     #Even is the LP is not solved to optimality (but f.e. with gap) we want
            to get the answer
104     if problem.solution.get_status() in
            [1,6,10,11,12,13,14,15,18,19,21,22,23,24,25,101,102,104,105,107,111,120,
```

```
        121,122,124,125,129,130,131]:
105         sol_times.append(tm.clock()-start)
106         sol_values.append(problem.solution.get_objective_value())
107         sol_status.append(problem.solution.get_status())
108     else:
109         sol_times.append('NaN')
110         sol_values.append('NaN')
111         sol_status.append(problem.solution.get_status())
112     print 'Total Cost = %s' %(problem.solution.get_objective_value())
113     print 'Total solution time = %s' %(tm.clock()-start)
```

A.3 Code for solving MkEE^p with partitioning algorithm

```
1  """
2  Uses the partitioning algorithm with specified number of cells to get
       solutions of specified number of instances of MkEE for specified
       parameters
3  """
4
5  import pulp
6  import cplex
7  import numpy as np
8  import time as tm
9
10 #Parameters
11 k=3
12 n=40
13 d=2
14 p=1
15 num_seeds=30 #number of instances
16 sol_times=dict()
17 sol_values=dict()
18
19 for sd in range(num_seeds):
20     sol_values[sd]=0 #Starts with 0 and adds the solution on each cell
21     np.random.seed(sd)
22     vertices=range(1,n+1)
23     locations=dict() #Contains coordinates of all vertices
24     for l in range(d):
25         locations[l]=dict()
26         for i in vertices:
27             locations[l][i]=np.random.random()
28     distance=dict() #Power-weighted distance between vertices
29     for i in vertices:
30         distance[i]=dict()
31         for j in vertices:
32             if i==j:
33                 distance[i][j]=0
34             else:
35                 distance_temp_temp=0
36                 for l in range(d):
37                     distance_temp_temp+=(abs(locations[l][i]-locations[l][j]))**p
38                 distance[i][j]=distance_temp_temp**(1./p)
39     n=len(vertices)
40
41     location_sort=dict() #location_sort[l]: all vertices sorted on their
           (l+1)-th coordinate
42     for l in range(d):
43         location_sort[l]=sorted(locations[l], key=locations[l].__getitem__)
```

```
44
45      num_subsets=dict() #num_subsets[l][i]: Number of [subsets after
            partitioning over l+1 dimensions] we want out of [subset i from after
            partitioning over l dimensions],
46      vertices_per_subset=dict() #vertices_per_subset[l]: minimum number of
            vertices per subset after partioning over l+1 dimensions
47      tot_subsets=dict() #tot_subsets[l]: total number of subsets after
            partitioning over l dimensions
48      num_subsets[0]=dict()
49      num_subsets[0][0]=2  #Number of subsets we want after the first
            partitioning
50      vertices_per_subset[0]=dict()
51      vertices_per_subset[0]=int(n/num_subsets[0][0])
52      tot_subsets[0]=1
53
54      subsets=dict() # subsets[l][i]: vertices in subset i after partitioning
            over l dimensions
55      subsets_sort=dict() # subsets_sort[l][i]: vertices in subset i after
            partitioning over l dimensions sorted on their (l+1)-th coordinate
56      subsets[0]=dict()
57      subsets[0][0]=vertices
58      subsets_sort[0]=dict()
59      subsets_sort[0][0]=location_sort[0]
60
61      for l in range(1,d+1):
62          subsets[l]=dict()
63          num_subsets[l]=dict()
64          subsets_sort[l]=dict()
65          vertices_per_subset[l]=25  #Can be adjusted to get the required
                minimal number of vertices in each subset
66          for i in range(tot_subsets[l-1]):
67              num_subsets_done=0 #Number of subsets we have created this
                    iteration
68              for j in range(int(num_subsets[l-1][i])):
69                  num_subsets_done=sum(int(num_subsets[l-1][r]) for r in
                        range(i))
70                  if (j+1)==int(num_subsets[l-1][i]):
71                      subsets[l][j+num_subsets_done]=
                            subsets_sort[l-1][i][j*vertices_per_subset[l-1]:]
72                  else:
73                      subsets[l][j+num_subsets_done]=
                            subsets_sort[l-1][i][j*vertices_per_subset[l-1]:
                            (j+1)*vertices_per_subset[l-1]]
74                  if l<d:
75                      subsets_sort[l][j+num_subsets_done]=
                            sorted(set(location_sort[l])&set(subsets[l][j+num_subsets_done]),
                            key=location_sort[l].index)
76                      num_subsets[l][j+num_subsets_done]=2  #Can be adjusted to
                            get the required number of subsets from one subset in
                            the previous iteration
77          tot_subsets[l]=int(sum(num_subsets[l-1].itervalues()))
78
79      #Solve the MILP for each cell to get a k-edge-connected graph
80      start=tm.clock()
81      for ss in subsets[d]: #now vertices=subsets[d][ss]
82
83          MkEE=pulp.LpProblem('Basic Fragmentation Model',pulp.LpMinimize)
84
```

```
85          #Variables
86          X={}
87          for i in subsets[d][ss]:
88              for j in subsets[d][ss]:
89                  X[i,j]=pulp.LpVariable("X_%s_%s"
                        %(i,j),lowBound=0,upBound=1,cat=pulp.LpInteger)
90
91          F={}
92          for i in subsets[d][ss]:
93              for u in subsets[d][ss]:
94                  for v in subsets[d][ss]:
95                      F[i,u,v]=pulp.LpVariable('F_%s_%s_%s' %(i,u,v),lowBound=0)
96
97          #Objective function
98          MkEE += sum( [sum([distance[i][j]*X[i,j] for j in subsets[d][ss] if
                j>i]) for i in subsets[d][ss]])
99
100         #Constraints
101         for i in subsets[d][ss]:
102             for j in subsets[d][ss]:
103                 if j>i:
104                     MkEE += X[i,j]==X[j,i]
105
106         for i in subsets[d][ss]:
107             for u in subsets[d][ss]:
108                 for v in subsets[d][ss]:
109                     MkEE += F[i,u,v]<=X[u,v]
110
111         for i in subsets[d][ss]: #extra constraint to make sure it does not
                send flow to itself, as the distance[i][i]=0
112             MkEE += X[i,i]==0
113             for u in subsets[d][ss]:
114                 MkEE += F[i,u,u]==0
115
116         for i in subsets[d][ss]:
117             for v in subsets[d][ss]:
118                 if (v != i and v != subsets[d][ss][0] and
                        i!=subsets[d][ss][0]):
119                     MkEE += sum( F[i,u,v] for u in subsets[d][ss]) == sum(
                            F[i,v,w] for w in subsets[d][ss])
120
121         for i in subsets[d][ss]:
122             MkEE += sum( F[i,subsets[d][ss][0],v] for v in subsets[d][ss]) -
                    sum( F[i,u,subsets[d][ss][0]] for u in subsets[d][ss])>= k
123
124         MkEE.solve(pulp.CPLEX(msg=1))
125         sol_values[sd]+=pulp.value(MkEE.objective)
126
127     #Merging solutions
128     merge_set=[]
129     if len(subsets[d])<(k+1):#Choose if we use snakelike structure or MILP
130         #We add string comparable to a one dimensional snippet of the
                hyperdimensional rectangle, as this snippet has succesive numbers
131         # F.e. in [1 2 3] we can add a snippet like [4 5 6] at once, though we
132         #          [4 5 6] need to take into account if we should at it as
133         #          [7 8 9] reverse ([6 5 4]) or as [4 5 6]
134         correct_order_set=[] #Outputs the snakelike order to lace them
                together
```

```
135         start_of_string =0 #Makes sure we are at the right number
136         reverse_string=0 #Do we need to add a reverse string or a normal one
137         order_string=dict()
138         for l in range(d):
139             order_string[l]=1
140         for l in range(d-1,0,-1): #Start with lowest dimension and work
                upwards
141             while len(correct_order_set)<np.prod([int(num_subsets[t][0]) for
                    t in range(d-1,l-2,-1)]): #while this dimension is not
                    finished yet
142                 if reverse_string==0:
143                     temp_string=range(start_of_string,(start_of_string+
                            int(num_subsets[d-1][0]))) #normal string
144                     start_of_string+=(int(num_subsets[d-1][0])*(1+
                            np.prod([order_string[z] for z in range(d)]))-1)
145                     reverse_string=1
146                 else:
147                     temp_string=range(start_of_string,(start_of_string-
                            int(num_subsets[d-1][0])),-1) #reverse string
148                     start_of_string-=(int(num_subsets[d-1][0])*(1-
                            np.prod([order_string[z] for z in range(d)]))-1)
149                     reverse_string=0
150                 correct_order_set+=temp_string #Add string of length l_(d-1)
                        (as we work backwards here)
151
152                 for t in range(l-1,d-1): #Reverse for-loop, as we want to
                        find the largest dimension that fits, to compensate for
                        switching from one x dimensional slab to the next
153                     if len(correct_order_set) %
                            np.prod([int(num_subsets[z][0]) for z in
                            range(t,d)])==0: #Check if len(correct_order_set) %
                            l_(d-1)*...*l_x ==0
154                         start_of_string-=int(num_subsets[d-1][0])*
                                np.prod([order_string[z] for z in range(d)])
155                         start_of_string+=np.prod([int(num_subsets[z][0]) for
                                z in range(t,d)])*np.prod([order_string[z] for z
                                in range(t)])#Add proper amount to start of
                                string, as for next one we need to add
                                l_(d-1)*...*l_x instead of l_(d-1):
156                         if order_string[t]==1:
157                             order_string[t]=-1
158                         else:
159                             order_string[t]=1
160                         break
161
162         set_belong=dict() #Assign each vertex to its cell
163         for ss in subsets[d]:
164             set_belong[ss]=dict()
165             for i in subsets[d][ss]:
166                 set_belong[ss][i]=1
167
168         for ss in range(len(subsets[d])-1):
169             merge_set=[]
170             merge_set+=subsets[d][correct_order_set[ss]][:]
171             merge_set+=subsets[d][correct_order_set[ss+1]][:]
172             # Laces two cells together
173             Match=pulp.LpProblem('Basic Fragmentation Model',pulp.LpMinimize)
174
```

```
175             #Variables
176             X={}
177             for i in merge_set:
178                 for j in merge_set:
179                     X[i,j]=pulp.LpVariable("X_%s_%s"
                            %(i,j),lowBound=0,upBound=1,cat=pulp.LpInteger)
180
181             #Objective function
182             Match += sum( [sum([distance[i][j]*X[i,j] for j in merge_set if
                    j>i]) for i in merge_set])
183
184             #Constraints
185             for i in merge_set: #Match needs to be symmetric
186                 for j in merge_set:
187                     if j>i:
188                         Match += X[i,j]==X[j,i]
189
190             for st in [correct_order_set[ss],correct_order_set[ss+1]]: #There
                    need to be k matches from one cell to the other
191                 Match += sum(sum(set_belong[st].get(i,0)*X[i,j] for j in
                        merge_set if set_belong[st].get(j,0)<0.1) for i in
                        merge_set) >= k
192
193             Match.solve(pulp.CPLEX(msg=1))
194             sol_values[sd]+=pulp.value(Match.objective)
195         sol_times[sd]=tm.clock()-start
196
197     else:
198         for ss in subsets[d]:
199             merge_set.append(subsets[d][ss][0]) #Use 0th vertex from each
                    subset to create k-edge-connected graph on
200
201         MkEE=pulp.LpProblem('Basic Fragmentation Model',pulp.LpMinimize)
202
203         #Variables
204         X={}
205         for i in merge_set:
206             for j in merge_set:
207                 X[i,j]=pulp.LpVariable("X_%s_%s"
                        %(i,j),lowBound=0,upBound=1,cat=pulp.LpInteger)
208
209         F={}
210         for i in merge_set:
211             for u in merge_set:
212                 for v in merge_set:
213                     F[i,u,v]=pulp.LpVariable('F_%s_%s_%s' %(i,u,v),lowBound=0)
214
215         #Objective function
216         MkEE += sum( [sum([distance[i][j]*X[i,j] for j in merge_set if j>i])
                for i in merge_set])
217
218         #Constraints
219         for i in merge_set:
220             for j in merge_set:
221                 if j>i:
222                     MkEE += X[i,j]==X[j,i]
223
224         for i in merge_set:
```

```
225                 for u in merge_set:
226                     for v in merge_set:
227                         MkEE += F[i,u,v]<=X[u,v]
228 
229             for i in merge_set: #extra constraint to make sure it does not send
                    flow to itself, as the distance[i][i]=0
230                 MkEE += X[i,i]==0
231                 for u in merge_set:
232                     MkEE += F[i,u,u]==0
233 
234             for i in merge_set:
235                 for v in merge_set:
236                     if (v != i and v != merge_set[0] and i!=merge_set[0]):
237                         MkEE += sum( F[i,u,v] for u in merge_set) == sum(
                                F[i,v,w] for w in merge_set)
238 
239             for i in merge_set:
240                 MkEE += sum( F[i,merge_set[0],v] for v in merge_set) - sum(
                        F[i,u,merge_set[0]] for u in merge_set)>= k
241 
242             MkEE.solve(pulp.CPLEX(msg=1))
243             sol_values[sd]+=pulp.value(MkEE.objective)
244             sol_times[sd]=tm.clock()-start
245         print 'Total Cost = %s' %(sol_values[sd])
246         print 'Total solution time = %s' %(sol_times[sd])
```

A.4 Code for solving MkEP^p with partitioning algorithm

```
1  """
2  Uses the partitioning algorithm with specified number of cells to get
       solutions of specified number of instances of MkEE for specified
       parameters
3  """
4 
5  import pulp
6  import cplex
7  import numpy as np
8  import time as tm
9  import operator
10 
11 #Parameters
12 k=3
13 n=40
14 d=2
15 p=1
16 num_seeds=30 #number of instances
17 sol_times=dict()
18 sol_values=dict()
19 
20 for sd in range(num_seeds):
21     sol_values[sd]=0 #Starts with 0 and adds the solution on each cell
22     np.random.seed(sd)
23     vertices=range(1,n+1)
24     locations=dict() #Contains coordinates of all vertices
25     for l in range(d):
26         locations[l]=dict()
27         for i in vertices:
28             locations[l][i]=np.random.random()
29     distance=dict() #Power-weighted distance between vertices
```

```
30      for i in vertices:
31          distance[i]=dict()
32          for j in vertices:
33              if i==j:
34                  distance[i][j]=0
35              else:
36                  distance_temp_temp=0
37                  for l in range(d):
38                      distance_temp_temp+=(abs(locations[l][i]-locations[l][j]))**p
39                  distance[i][j]=distance_temp_temp**(1./p)
40      n=len(vertices)
41
42      location_sort=dict() #location_sort[l]: all vertices sorted on their
            (l+1)-th coordinate
43      for l in range(d):
44          location_sort[l]=sorted(locations[l], key=locations[l].__getitem__)
45
46      num_subsets=dict() #num_subsets[l][i]: Number of [subsets after
            partitioning over l+1 dimensions] we want out of [subset i from after
            partitioning over l dimensions],
47      vertices_per_subset=dict() #vertices_per_subset[l]: minimum number of
            vertices per subset after partioning over l+1 dimensions
48      tot_subsets=dict() #tot_subsets[l]: total number of subsets after
            partitioning over l dimensions
49      num_subsets[0]=dict()
50      num_subsets[0][0]=2   #Number of subsets we want after the first
            partitioning
51      vertices_per_subset[0]=dict()
52      vertices_per_subset[0]=int(n/num_subsets[0][0])
53      tot_subsets[0]=1
54
55      subsets=dict() # subsets[l][i]: vertices in subset i after partitioning
            over l dimensions
56      subsets_sort=dict() # subsets_sort[l][i]: vertices in subset i after
            partitioning over l dimensions sorted on their (l+1)-th coordinate
57      subsets[0]=dict()
58      subsets[0][0]=vertices
59      subsets_sort[0]=dict()
60      subsets_sort[0][0]=location_sort[0]
61
62      for l in range(1,d+1):
63          subsets[l]=dict()
64          num_subsets[l]=dict()
65          subsets_sort[l]=dict()
66          vertices_per_subset[l]=25   #Can be adjusted to get the required
                minimal number of vertices in each subset
67          for i in range(tot_subsets[l-1]):
68              num_subsets_done=0 #Number of subsets we have created this
                    iteration
69              for j in range(int(num_subsets[l-1][i])):
70                  num_subsets_done=sum(int(num_subsets[l-1][r]) for r in
                        range(i))
71                  if (j+1)==int(num_subsets[l-1][i]):
72                      subsets[l][j+num_subsets_done]=
                            subsets_sort[l-1][i][j*vertices_per_subset[l-1]:]
73                  else:
74                      subsets[l][j+num_subsets_done]=
                            subsets_sort[l-1][i][j*vertices_per_subset[l-1]:(j+1)*vertices_per_sub
```

```
75                  if l<d:
76                      subsets_sort[l][j+num_subsets_done]=sorted(set(location_sort[l])&set(subse
                            key=location_sort[l].index)
77                      num_subsets[l][j+num_subsets_done]=2  #Can be adjusted to
                            get the required number of subsets from one subset in
                            the previous iteration
78          tot_subsets[l]=int(sum(num_subsets[l-1].itervalues()))
79
80      #Solve the MILP for each cell to get a k-edge-connected power assignment
            graph
81      start=tm.clock()
82      pa=dict() #Remember how much power has been assigned to each vertex
83      for ss in subsets[d]:
84
85          MkEV=pulp.LpProblem('Basic Fragmentation Model',pulp.LpMinimize)
86
87          #Variables
88          X={}
89          for i in subsets[d][ss]:
90              for j in subsets[d][ss]:
91                  X[i,j]=pulp.LpVariable("X_%s_%s"
                        %(i,j),lowBound=0,upBound=1,cat=pulp.LpInteger)
92
93          F={}
94          for i in subsets[d][ss]:
95              for u in subsets[d][ss]:
96                  for v in subsets[d][ss]:
97                      F[i,u,v]=pulp.LpVariable('F_%s_%s_%s' %(i,u,v),lowBound=0)
98
99          A={}
100         for i in subsets[d][ss]:
101             A[i]=pulp.LpVariable('A_%s' %(i),lowBound=0)
102
103         #Objective function
104         MkEV += sum( [A[i] for i in subsets[d][ss]])
105
106         #Constraints
107         for i in subsets[d][ss]:
108             for j in subsets[d][ss]:
109                 MkEV += A[i] >= distance[i][j]*X[i,j]
110
111         for i in subsets[d][ss]:
112             for j in subsets[d][ss]:
113                 if j>i:
114                     MkEV += X[i,j]==X[j,i]
115
116         for i in subsets[d][ss]:
117             for u in subsets[d][ss]:
118                 for v in subsets[d][ss]:
119                     MkEV += F[i,u,v]<=X[u,v]
120
121         for i in subsets[d][ss]: #extra constraint to make sure it does not
            send flow to itself, as the distance[i][i]=0
122             MkEV += X[i,i]==0
123             for u in subsets[d][ss]:
124                 MkEV += F[i,u,u]==0
125
126         for i in subsets[d][ss]:
```

```
127             for v in subsets[d][ss]:
128                 if (v != i and v != subsets[d][ss][0] and
                        i!=subsets[d][ss][0]):
129                     MkEV += sum( F[i,u,v] for u in subsets[d][ss]) == sum(
                            F[i,v,w] for w in subsets[d][ss])
130
131         for i in subsets[d][ss]:
132             MkEV += sum( F[i,subsets[d][ss][0],v] for v in subsets[d][ss]) -
                    sum( F[i,u,subsets[d][ss][0]] for u in subsets[d][ss])>= k
133
134         MkEV.solve(pulp.CPLEX(msg=1))
135         for i in subsets[d][ss]:
136             pa[i]=pulp.value(A[i])
137         sol_values[sd]+=pulp.value(MkEV.objective)
138
139     #Merging solutions
140     merge_set=[]
141     if len(subsets[d])<(k+1):#Choose if we use snakelike structure or MILP
142         #We add string comparable to a one dimensional snippet of the
                hyperdimensional rectangle, as this snippet has succesive numbers
143         # F.e. in [1 2 3] we can add a snippet like [4 5 6] at once, though we
144         #          [4 5 6] need to take into account if we should at it as
145         #          [7 8 9] reverse ([6 5 4]) or as [4 5 6]
146         correct_order_set=[] #Outputs the snakelike order to lace them
                together
147         start_of_string =0 #Makes sure we are at the right number
148         reverse_string=0 #Do we need to add a reverse string or a normal one
149         order_string=dict()
150         for l in range(d):
151             order_string[l]=1
152         for l in range(d-1,0,-1): #Start with lowest dimension and work
                upwards
153             while len(correct_order_set)<np.prod([int(num_subsets[t][0]) for
                    t in range(d-1,l-2,-1)]): #while this dimension is not
                    finished yet
154                 if reverse_string==0:
155                     temp_string=range(start_of_string,(start_of_string+
                            int(num_subsets[d-1][0]))) #normal string
156                     start_of_string+=(int(num_subsets[d-1][0])*(1+
                            np.prod([order_string[z] for z in range(d)]))-1)
157                     reverse_string=1
158                 else:
159                     temp_string=range(start_of_string,(start_of_string-
                            int(num_subsets[d-1][0])),-1) #reverse string
160                     start_of_string-=(int(num_subsets[d-1][0])*(1-
                            np.prod([order_string[z] for z in range(d)]))-1)
161                     reverse_string=0
162                 correct_order_set+=temp_string #Add string of length l_(d-1)
                        (as we work backwards here)
163
164                 for t in range(l-1,d-1): #Reverse for-loop, as we want to
                        find the largest dimension that fits, to compensate for
                        switching from one x dimensional slab to the next
165                     if len(correct_order_set) %
                            np.prod([int(num_subsets[z][0]) for z in
                            range(t,d)])==0: #Check if len(correct_order_set) %
                            l_(d-1)*...*l_x ==0
```

```
166                     start_of_string-=int(num_subsets[d-1][0])*
                            np.prod([order_string[z] for z in range(d)])
167                     start_of_string+=np.prod([int(num_subsets[z][0]) for
                            z in range(t,d)])*np.prod([order_string[z] for z
                            in range(t)])#Add proper amount to start of
                            string, as for next one we need to add
                            l_(d-1)*...*l_x instead of l_(d-1):
168                     if order_string[t]==1:
169                         order_string[t]=-1
170                     else:
171                         order_string[t]=1
172                     break
173
174         set_belong=dict() #Assign each vertex to its cell
175         for ss in subsets[d]:
176             set_belong[ss]=dict()
177             for i in subsets[d][ss]:
178                 set_belong[ss][i]=1
179
180         for ss in range(len(subsets[d])-1):
181             merge_set=[]
182             merge_set+=subsets[d][correct_order_set[ss]][:]
183             merge_set+=subsets[d][correct_order_set[ss+1]][:]
184             # Laces two cells together
185             Match=pulp.LpProblem('Basic Fragmentation Model',pulp.LpMinimize)
186
187             #Variables
188             X={}
189             for i in merge_set:
190                 for j in merge_set:
191                     X[i,j]=pulp.LpVariable("X_%s_%s"
                                %(i,j),lowBound=0,upBound=1,cat=pulp.LpInteger)
192
193             A={} #Takes into account the A[i] already present before matching
194             for i in merge_set:
195                 A[i]=pulp.LpVariable('A_%s' %(i),lowBound=pa[i])
196
197             #Objective function
198             Match += sum( [(A[i]-pa[i]) for i in merge_set])
199
200             #Constraints
201             for i in merge_set: #We can only use an edge if the power is
                    large enough
202                 for j in merge_set:
203                     MkEV += A[i] >= distance[i][j]*X[i,j]
204
205             for i in merge_set: #Match needs to be symmetric
206                 for j in merge_set:
207                     if j>i:
208                         Match += X[i,j]==X[j,i]
209
210             for st in [correct_order_set[ss],correct_order_set[ss+1]]: #There
                    need to be k matches from one cell to the other
211                 Match += sum(sum(set_belong[st].get(i,0)*X[i,j] for j in
                        merge_set if set_belong[st].get(j,0)<0.1) for i in
                        merge_set) >= k
212
213             Match.solve(pulp.CPLEX(msg=1))
```

```
214             sol_values[sd]+=pulp.value(Match.objective)
215         sol_times[sd]=tm.clock()-start
216 
217     else:
218         for ss in subsets[d]:
219             sub_pa=dict() #connect each vertex with the correct power
                    assignment
220             for i in subsets[d][ss]:
221                 sub_pa[i]=pa[i]
222             merge_set.append(max(sub_pa.iteritems(),
                    key=operator.itemgetter(1))[0]) # Take vertex from each cell
                    with highest A[i]
223 
224         MkEV=pulp.LpProblem('Basic Fragmentation Model',pulp.LpMinimize)
225 
226         #Variables
227         X={}
228         for i in merge_set:
229             for j in merge_set:
230                 X[i,j]=pulp.LpVariable("X_%s_%s"
                        %(i,j),lowBound=0,upBound=1,cat=pulp.LpInteger)
231 
232         F={}
233         for i in merge_set:
234             for u in merge_set:
235                 for v in merge_set:
236                     F[i,u,v]=pulp.LpVariable('F_%s_%s_%s' %(i,u,v),lowBound=0)
237 
238         A={} #Takes into account the A[i] already present before merging
239         for i in merge_set:
240             A[i]=pulp.LpVariable('A_%s' %(i),lowBound=pa[i])
241 
242         #Objective function
243         MkEV += sum( [(A[i]-pa[i]) for i in merge_set])
244 
245         #Constraints
246         for i in merge_set:
247             for j in merge_set:
248                 MkEV += A[i] >= distance[i][j]*X[i,j]
249 
250         for i in merge_set:
251             for j in merge_set:
252                 if j>i:
253                     MkEV += X[i,j]==X[j,i]
254 
255         for i in merge_set:
256             for u in merge_set:
257                 for v in merge_set:
258                     MkEV += F[i,u,v]<=X[u,v]
259 
260         for i in merge_set: #extra constraint to make sure it does not send
                flow to itself, as the distance[i][i]=0
261             MkEV += X[i,i]==0
262             for u in merge_set:
263                 MkEV += F[i,u,u]==0
264 
265         for i in merge_set:
266             for v in merge_set:
```

```
267                 if (v != i and v != merge_set[0] and i!=merge_set[0]):
268                     MkEV += sum( F[i,u,v] for u in merge_set) == sum(
                            F[i,v,w] for w in merge_set)
269
270         for i in merge_set:
271             MkEV += sum( F[i,merge_set[0],v] for v in merge_set) - sum(
                        F[i,u,merge_set[0]] for u in merge_set)>= k
272
273         MkEV.solve(pulp.CPLEX(msg=1))
274         sol_values[sd]+=pulp.value(MkEV.objective)
275         sol_times[sd]=tm.clock()-start
276     print 'Total Cost = %s' %(sol_values[sd])
277     print 'Total solution time = %s' %(sol_times[sd])
```