Segmentation of street light networks

Mathijsen, B.W.J.

Award date:
2013

Link to publication
Segmentation of street light networks

B.W.J. Mathijsen

Supervised by

Eindhoven University of Technology
prof. dr. J.S.H. van Leeuwaarden
dr. N. Bansal

Philips Research
dr. T.J.J. Denteneer
M. Aoun

TU/e PHILIPS

Thesis submitted in partial fulfilment
of the requirements for the degree

Master of Science

at the Eindhoven University of Technology
Department of Mathematics and Computer Science

August, 2013
Abstract

This thesis studies the segmentation of deployments consisting of street lights equipped with wireless networking capabilities. This allows information on the performance of street lights to be communicated via wireless multi-hop networking to one central unit. The goal is to find a partition of street lights and positions of the so-called segment controllers, optimising network performance in terms of end-to-end delay, failure rate and duty cycle.

We first derived a mathematical model based on a connectivity graph from this model. Although there is a wide variety of graph partitioning methods available in literature, these were shown to be unfit for our problem, because the wireless performance measures do not trivially translate into a objective function based on graph structure. We verified this claim by investigating two conventional graph partitioning algorithms in the first part of this thesis, namely spectral clustering and $k$-medoids.

Instead, we resorted to a relatively new type of partitioning algorithms, namely the local algorithms. These algorithms optimise conductance of clusters, which is defined as the ratio between the number of external and internal edges. The local algorithms by Spielman-Teng and Andersen-Peres construct clusters step by step, only taking into account the local structure of the graph. This gave us the opportunity to influence the construction process in order to generate good quality clusters with respect to the network performance. Furthermore, we obtained insight into both local algorithms by applying the algorithms to stylised graphs and we stated some general observations on the algorithms.

In order to complete the partitioning algorithm based on these local methods, we developed a series of auxiliary routines. Finally, the compared the network performance of the old algorithms and the novel algorithms, by the means of a self-built simulation. Using the simulation, we were also able to provide a list of graph properties which serve as goals or objectives when partitioning a deployment of street lights. The general outcome of these comparison of algorithms was that $k$-medoids, under some assumptions, optimises for delay, whereas the local algorithms perform better on failure rate and duty cycle.
Acknowledgements

I would like to express my gratitude to the people without whom I would not have been able to write this thesis.

I owe my deepest gratitude to my first supervisor Johan van Leeuwaarden, whose support and enthusiasm I greatly appreciated. His endless encouragement through the highs and lows of this project motivated me to succeed in materialising this thesis. To Nikhil Bansal, my second supervisor, I like to give my thanks for the fruitful discussions and contributing his knowledge on the subject.

Also, I would like to thank the people at Philips for granting me the opportunity to experience working in a innovative company. Thank you Dee Denteneer for your supportive words and constructive criticism during our discussions. Marc Aoun, thank you for helping me on the practical part of the project.

I wish to thank my intern colleagues at Philips for the nice atmosphere and my ‘collegues’ at the university for the coffee breaks. In particular, I would like to express my appreciation to Jorn van der Pol for his insightful ideas and comments, which allowed me to complete some of the mathematical results in this thesis.

To my parents, Jan and Willemien, and my sister Kim I owe my gratitude for their unconditional support in everything but mathematics.

Last but not least, I like to thank my friends for making my time as a student, especially the last year, a very enjoyable phase of my life.

Britt Mathijsen

August 2013
### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.2</td>
<td>Local scaling</td>
<td>35</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Algorithm and implementation</td>
<td>35</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Example</td>
<td>36</td>
</tr>
<tr>
<td>3.3</td>
<td>k-Medoids</td>
<td>40</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Algorithm</td>
<td>40</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Initialisation</td>
<td>41</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Example</td>
<td>44</td>
</tr>
<tr>
<td>3.4</td>
<td>Remarks</td>
<td>44</td>
</tr>
<tr>
<td>4</td>
<td>Local partitioning methods</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>Local versus global</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Structure</td>
<td>48</td>
</tr>
<tr>
<td>4.3</td>
<td>Generating clusters</td>
<td>48</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Spielman-Teng</td>
<td>49</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Andersen-Peres</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>Stylised graphs</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Dumbbell graph</td>
<td>53</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Spielman-Teng</td>
<td>54</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Andersen-Peres</td>
<td>55</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Bounds on mixing time</td>
<td>60</td>
</tr>
<tr>
<td>5.2</td>
<td>Grid network</td>
<td>62</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Spielman-Teng</td>
<td>62</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Andersen-Peres</td>
<td>68</td>
</tr>
<tr>
<td>5.3</td>
<td>Conclusions</td>
<td>76</td>
</tr>
<tr>
<td>6</td>
<td>General observations on local algorithms</td>
<td>77</td>
</tr>
<tr>
<td>6.1</td>
<td>Properties of Spielman-Teng</td>
<td>77</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Simplified algorithm</td>
<td>77</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Monotonicity</td>
<td>78</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Connectivity</td>
<td>81</td>
</tr>
<tr>
<td>6.2</td>
<td>Properties of Andersen-Peres</td>
<td>83</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Randomness</td>
<td>83</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Inclusion of streets</td>
<td>83</td>
</tr>
<tr>
<td>6.3</td>
<td>Adding constraints</td>
<td>87</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Restricting construction</td>
<td>87</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Possible constraints</td>
<td>88</td>
</tr>
<tr>
<td>6.4</td>
<td>Stopping criteria</td>
<td>88</td>
</tr>
<tr>
<td>6.5</td>
<td>Including spatial information</td>
<td>89</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Introducing weights</td>
<td>89</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Weight function</td>
<td>91</td>
</tr>
<tr>
<td>7</td>
<td>Auxiliary routines</td>
<td>93</td>
</tr>
<tr>
<td>7.1</td>
<td>Framework</td>
<td>93</td>
</tr>
<tr>
<td>7.2</td>
<td>First approach</td>
<td>93</td>
</tr>
<tr>
<td>7.3</td>
<td>Seed selection</td>
<td>94</td>
</tr>
<tr>
<td>7.4</td>
<td>Choosing clusters for covering</td>
<td>95</td>
</tr>
<tr>
<td>7.5</td>
<td>Covering to partition</td>
<td>96</td>
</tr>
<tr>
<td>7.6</td>
<td>Example</td>
<td>99</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Problem setting

1.1.1 Street light networks

In daily life, street lighting provides us with comfort and safety. To ensure a reliable lighting network, the system needs to be monitored continuously, so that failures can be detected quickly. Furthermore, operators are able to control the lighting networks, by turning them on, off or dimming them, or distributing software updates. These issues illustrate the necessity of communication between individual street lights and a central operating unit. Until now, this has been done by wired communication between street lights and a central unit. In the Starsense Wireless project, Philips has been developing a system for replacing the wired communication by wireless communication via radio frequency. Wireless networking has many advantages over wired communication. As mentioned in [GSA+], the deployment costs of a wireless network are relatively low compared to the wired connections. Also, adding or removing street lights is much less of an issue.

The wireless network is set up in the following way. The two main devices considered in this thesis are outdoor lighting controllers (OLCs) and segment controllers (SCs). OLCs are relatively cheap devices that are equipped with wireless networking capabilities. One OLC is mounted on top of each street light, establishing the wireless connection between street lights. SCs are expensive devices, also equipped with wireless networking capabilities, coordinating the traffic of all OLCs assigned to it. All traffic passing through the network is either directed from the SC towards an OLC or vice versa. Typical messages being sent by the SC to its OLCs are software updates or lighting adjustment commands. Messages sent by an OLC towards the SC are usually log reports or acknowledgments. An SC is usually placed in a street cabinet, together with a modem which connects it to the internet, which in turn is connected to management software.

Network devices are able to communicate directly whenever they are within the reliable connectivity range and within line of sight. For this application, the connectivity range is fixed to 300 meters. Although an OLC may not be within direct reach of a segment controller, it is still possible for the OLC and SC to communicate by multi-hop communication. This means that a message generated at an OLC further than 300 meter away from its SC can be routed through other OLCs towards to SC (or vice versa). The SCs and OLCs together hence define a so-called mesh network.
CHAPTER 1. INTRODUCTION

1.1.2 Routing protocols

The process used to route messages from a segment controller to an OLC is called flooding. In this procedure, the segment controller broadcasts the message intended for an individual OLC or group of OLCs towards all OLCs within direct reach. OLCs that received the message correctly again forward the message to all downstream OLCs within their reach, and so on, until the message is correctly received by the destination OLC, and an acknowledgement is received by the segment controller. The message is therefore forwarded like a kind of wave through the network, which explains the name of the process. If no acknowledgement from the OLC the message was intended for is received within a certain time period, a retransmission is initiated at the SC and the procedure starts all over again.

Transmissions by an OLC to the SC are routed according to a procedure called SinkDV. Every OLC in the network keeps track of its distance to its assigned SC in terms of ‘hops’, which we will also refer to as hop count. The hop count of an OLC is equal to the minimum number of intermediary OLCs (including itself) a message generated at the OLC has to pass before reaching the segment controller. The OLC within direct reach of the segment controller hence have hop count 1. The OLCs reachable by the OLCs of hop count 1 have hop count equal to 2, etcetera. SinkDV is based on these parameters. At each OLC, one or more so-called uplinks (or parents) are stored, having hop count one less than the hop count of the OLC itself. Whenever the OLC has a queued message, it selects one of its uplinks to forward the message to. This way, the message is automatically routed towards the segment controller. An example network together with the hop count values and uplinks is depicted in Figure 1.1.

Because in practice, the amount of traffic towards the SC outweighs the amount generated by flooding, we will mainly focus on traffic routed towards the segment controller in this thesis.

1.1.3 Segmenting the network

Of course, using wireless networking instead of wired communication poses some restrictions on the network. Not all transmissions will be successful, because wireless sensor networks inherently suffer from interference induced by either the network itself or third parties. Also, traffic load may be adding up at OLCs close to the SC, since all traffic needs to pass through
these nodes, leading to limited capacity. Therefore, the number of OLCs a SC is capable to support is limited. Experience shows that the capacity per segment controller is about 2000 to 2500 OLCs, depending on the topology of the network. Since city deployments typically consist of tens of thousands of street lights, multiple SCs, although very costly, are needed to guide the communication between management software and the individual street lights. The fact that we employ more than one segment controller leads to the question of how to assign OLCs to SCs. In other words, how do we divide the full deployment into segments such that it satisfies the restrictions inherent to the wireless network and network performance is optimised? Also, where do we position the segment controller?

These are the main research questions we will try to answer in this thesis. Next to the restrictions associated with the wireless network, the segmentation has to meet following additional constraints:

- **Connectivity**
  In each segment, each OLC should be able to communicate with the segment controller.

- **Duty cycle requirement**
  European law prescribes that the percentage of time a device is transmitting should be below 1%. For deployments in China and the USA, this regulation does not apply.

- **Street connectivity**
  All OLCs in one street should be assigned to the same segment controller. This way, one segment controller is able to steer the controls of one street fully.

This last requirement is less strict, and we will not go further into this restriction for the fact that this is only a matter of slightly adjusting the partition afterwards.

### 1.2 Deployments

To get a feeling for the structure of the street light networks we are dealing with, we will take a look at three layouts that will be used most in the remainder of this thesis. The first and most important example will be referred to as deployment A. Philips has the ambition to install this network within the next five years and hence its analysis has priority over the analysis of other data sets. The street light network of deployment A is depicted in Figure 1.2, in which each dot represents a street light. The total number of lights in this deployment is 30547. The average number of neighbours, that is the number of lights within communication distance, is roughly 42. From this figure, we clearly see that the lights are mostly ordered along the streets. This characteristic is inherent to the city deployments we deal with. Furthermore, the density of lights differs enormously for different regions of the map. Figure 1.3 illustrates this nicely. The average number of neighbours in Figure 1.3a equals 22, roughly half of the overall average. Figure 1.3b depicts the city center, which has an average number of neighbours equal to 75. Despite the fact that dense networks typically allow many paths between individual lights and their segment controllers, and hence most probably spread traffic evenly, having a large number of neighbours may also cause increased interference, leading to higher loss (or failure) rates. Figure 1.3c shows an example of a long road contained in the map, of which the density is on average 27 neighbours per light. Intuition tells us that these structures in the deployment are severe bottlenecks, since load generated at the end of such a street needs to be routed through the whole street to reach the segment controller. Although the average number of neighbours is in this case lower than in Figure 1.3b, this still leads to an uneven
CHAPTER 1. INTRODUCTION

Figure 1.2: Street light deployment A. Distances are plotted in meters.

A second data set we will be working with is denoted by deployment B, depicted in Figure 1.4. This deployment contains 19849 lights. Note that the bounding box around all lights is much smaller than in the case of deployment A, resulting in a much denser network. This is also reflected in the average number of neighbours per light: 226. Any segmentation for this deployment will thus result in a small average number of hops between a light and its segment controller, while interference will play a big role in the performance of the network. Another observation we can make is that the lights are scattered quite evenly across the map in comparison with deployment A, yielding a fairly homogeneous network.

The third deployment we will discuss differs tremendously from the previous two. Deployment C consists of street lights along highways only and is depicted in Figure 1.5. Although this network is highly inhomogeneous, the lights along the streets are close to each other, giving an average number of neighbours per light of 58. Still, all traffic is routed along the same lines. Hence traffic load will increase along these streets in the direction of the segment controller, causing congestion and much interference. Networks like deployment C are hence likely to perform worse that deployments like A and B, in which it is possible to route the traffic more evenly. This feature of wireless multi-hop networks will become more clear in Chapter 9.
1.3. Mathematical modelling

To deal with the problem of designing a street light network segmentation as discussed in Subsection 1.1.3 we choose to formulate a mathematical model. This will enable us to tackle the question of how to segment the deployment, by applying existing and novel algorithms to this mathematical model. Next, we elaborate some important features of the model.

1.3.1 Connectivity graph

A first step in modelling the problem is to describe the street light network in terms of a graph. In this undirected graph, each OLC is represented by a vertex, and edges define which lights are able to communicate directly with each other. We will refer to this graph as the
CHAPTER 1. INTRODUCTION

Figure 1.4: Street light deployment B. Distances are plotted in meters.

Figure 1.5: Street light deployment C. Distances are plotted in meters.

connectivity graph of the network. A modelling assumption we make is that the connectivity distance is fixed to 300 meters. That is, OLCs are able to communicate when they are within
300 meters of each other. In reality, this bound is less sharp and OLCs might be able to communicate if interference is low even though the distance between them is more than 300 meters. Throughout this thesis, we will use the terms street light, OLC, vertex and node interchangeably. Here we should stress that the choice of \( d = 300 \) is without loss of generality, since our mathematical model and algorithms are equipped to deal with an arbitrary value \( d \). Furthermore, we assume in the remainder of this thesis that the connectivity graph consist of one component. If a deployment consists of multiple components, one can apply the algorithms we discuss to each of the components separately.

\[ \textbf{1.3. Graph partitioning} \]

Once the connectivity graph of the street light network is constructed, a natural mathematical approach is to apply to this graph one of the many graph partitioning algorithms. A graph partition is a division of the vertices of the graph into subsets. The partition must satisfy certain properties and optimise some objective function. Because of the many applications of graph partitioning, it is a well-studied topic in graph theory, see for instance [Sch07] and [KVV00]. Most algorithms studied in literature use an objective function related to the so-called cut of the partition. The cut of a partition is number of edges connecting the different subsets of the partition. In min-cut partitioning, this exact value is minimised, while in min-max cut partitioning the maximum number of edges leaving a subset is minimised. Of course, cutting a segment off where it has only a few to the outside of the set is also beneficial in our setting. By the reasoning in the previous section, we prefer long roads (weak connections) to be on the boundary of a segment rather than close to the segment controller, which is naturally placed at the center of the segment. However, there are also some major differences between usual graph partitioning and the problem setting discussed in this thesis, since we want to optimise performance measures of the network induced by the graph, which do not trivially follow from the structure of the graph. The performance measures we are interested in are discussed in the next subsection.

Our problem also differs from conventional graph partitioning regarding the constraints on the network posed in the previous section. The first constraint, requiring connectivity, is easily translated to the mathematical setting, by requiring that a solution to the partitioning problem is only feasible if the subgraph induced by each subset of the partition is connected. As we will see later, properties of the wireless network like loss rates and duty cycles are hard to compute exactly, due to the high levels of uncertainty introduced by interference and random features in the protocols used. Therefore, we will only be able to analyse network performance, including the duty cycle requirement, of the partitioning by simulation.

Next to the unconventional constraints posed on the partitioning of the connectivity graph, the position of the segment controller also plays a role in partitioning of the graph. Hence we also need to account for this in optimising the segmentation.

A possible segmentation of deployment A is presented in Figure 1.6.

\[ \textbf{1.3.3 Performance measures} \]

Before defining the performance measures of interest, we will very briefly elaborate the properties of the underlying wireless networking protocol, IEEE 802.15.4, see [IEE06]. Because wireless networks inherently suffer from packet losses due to interference, protocols like IEEE 802.15.4 were developed to guide the transmission in mesh networks. The IEEE 802.15.4 used
Figure 1.6: Segmentation of the deployment A. Different colours represent different segments.
1.4. RESEARCH QUESTIONS

by the application discussed in this thesis relies on the carrier sense multiple access principle, silencing nodes close to the transmitting node. The main structure of the a packet transmission following this protocol is as follows.

Suppose that at a certain point in time, a node either receives or generates a new packet. First, a back-off period of random length is initiated. When this back-off period expires, the node checks the channel for availability. If the node senses noise generated by neighbouring nodes above a certain threshold, the channel is considered busy, and a new (longer) back-off period is initiated. This process is repeated until the node eventually finds the channel idle, or the maximum number of back-off periods is reached, in which case the packet is dropped and considered lost. Assuming that the node finds an idle channel, the transmission is started. Still, the packet transmission may fail, because the noise level, caused by neighbouring transmissions, at the receiving node is too high. We call this type of packet loss a collision. After a successful transmission, the receiving node answers by sending an acknowledgement. If the transmitting node does not receive this acknowledgement withing a certain time, the node expects the packet to be lost and a retransmission is planned. The number of retrials is also limited. A more elaborate description of the IEEE 802.15.4 can be found in Subsection 8.2 and [IEE06].

We now discuss the main performance measures regarding the wireless network we want to optimise:

- **Delay**
  The delay of a packet is given by the time it takes a packet generated at an OLC to reach the segment controller correctly.

- **Failure rate**
  The failure rate is defined as the probability that a packet is unable to successfully reach the segment controller, either by a collision or medium access failure.

- **Maximum duty cycle**
  The duty cycle of a OLC is equal to the fraction of time the OLC is actively transmitting. The maximum duty cycle of a segment is determined by the maximum among the values of all OLCs.

Of course, we would like to minimise all three of these measures. Note that these measures apply to a single segment only. We choose to assess the quality of a segmentation on the worst case and average values among all segments.

1.4 Research questions

As we have seen, our problem statement differs in many ways from the widely studied classical graph partitioning problem. Therefore, a deeper study of the implications of these differences to finding the optimal graph segmentation is needed. To do so, our goal is to answer three central questions in this thesis.

**A) How do graph characteristics relate to the performance of the network?**

Intuition tells us that traffic routed through a long road without sideways will cause a lot of congestion compared to traffic routed through a nicely interlaced street network. On the other hand, areas densely populated by lights are more likely to suffer from message losses due to interference. Therefore, it is not immediately clear what graph characteristics guarantee
good network performance. Moreover, it is far from evident how to decide on a network level whether a segmentation is either good or bad with respect to throughput, delay and loss rates. Investigation of this topic is needed and we will try to determine one or more graph characteristics that are indicators for good or bad network performance.

B) Can we modify known partitioning algorithms such that they are suitable for the application of wireless networks?

There are many algorithms developed to partition a graph in order to optimise some objective functions. However, almost no research has been done on partitioning wireless networks. Because objective functions on performance measures for these networks are not as clear as in classical graph partitioning, we have to modify, or even develop our own algorithms. Since our problem setting also contains restrictions with regards to the size of the segments, delay and loss rate of messages, conventional algorithms will not be sufficient. Our goal is to construct an algorithm that produces a segmentation of the networks that optimises network performance, while satisfying these constraints. If we are able to design such a method, we will of course verify whether it indeed outperforms previously developed methods by means of simulation.

C) How does the fact that we have spatial information come into play in segmenting the network?

In classical graph partitioning problems, the adjacency matrix, describes the connections between vertices. Algorithms only use this information to construct a partition. However, we have the advantage of having more information about the vertices, namely the coordinates. From a segmentation point of view, using this information might be beneficial, because we prefer to group lights laying close together in same segment over two lights that are just within each others communication range. The question is how to include this spatial information in the construction of the segmentation, and whether this extra information indeed improves the quality of the resulting segmentation.

1.5 Outline and contributions

1.5.1 Structure of the thesis

Figure 1.7 gives an overview of the chapters of this thesis and how they are connected. In Chapter 2 we review some basic facts on graphs, random walks and spectral methods, which prove useful in later chapters. After this, there will be a clear division in the structure. Following the left column in Figure 1.7 the so-called global partitioning methods will be discussed, while the right column leads local partitioning algorithms. The difference between global and local algorithm will become clear in the thesis. First, two previously developed global algorithms will be discussed in Chapter 3, namely spectral clustering and the combination of spectral clustering and $k$-medoids by Marc Aoun in collaboration with Natasha Jovanovic and Richard Verhoeven. Next we discuss in four consecutive chapters local partitioning algorithms. In Chapter 4, a brief outline of the structure of local algorithms and two new clustering techniques are presented. Then, in Chapter 5, we apply the two local algorithms to some simple graphs, namely the two-dimensional grid and a graph with a clear bottleneck, to built some intuition on how both methods produce segments. Next, we try to clarify the complex theory behind the local partitioning algorithms, after which we show how to incorporate
additional constraints into the algorithms. We hope to be able to answer research questions B and C in this part. Here, we will also see how to include spatial information. Chapter 7 completes the local partitioning path by discussing some routines on seed vertex selection and transforming the clusters found by the algorithms into an actual partition. To compare the results of both the global and local algorithms, we wrote a simulation package. Chapter 8 describes this program and the abstracted model it implements. After Chapter that we have gathered enough to compare the results of all algorithms in terms of network performance using the simulation. This comparison is done in Chapter 9. Finally, we try to answer the research questions posed in Section 1.4 and draw our conclusions.

Readers that are more interested in the practical part of the project are recommended to follow the path of Chapters 2, 3, 4, 8 and 9. The reader with mathematical background might rather take the right-hand path in Figure 1.7 visiting Chapter 4, 5 and 6 more thoroughly.

1.5.2 Contributions

The contributions we made to solve the segmentation problem are the following:

- Because the two local clustering algorithms we discuss in this thesis have been developed quite recently, not much research has been done on their application to simple graph structures. We were able to perform an exact analysis of both algorithms applied to two stylised graphs. By this analysis, we derived bounds on the complexity of the algorithms for these types of graphs. These results can be found in Chapter 5.

- Chapter 6 is dedicated to making the theory behind the local clustering algorithm comprehensible. The two papers that present the local algorithms contain very deep mathematical notions and proofs. We attempt to clarify at least the reasoning behind these techniques and state a few general observations in this chapter.

- Since the two local algorithms only apply to unweighted graphs, we took the opportunity to extend them to weighted graphs, by which we could include spatial data of
the network in the algorithm. Also, we found a way to add constraints to the construction step of the algorithms. These modifications to the algorithms can also be found in Chapter 6.

- Although the papers on local clustering algorithms propose a method to partition a graph using the single cluster generating algorithms, these have been proven inefficient to our setting. Therefore, we came up with two methods of our own to construct a full segmentation, using the constructed clusters as building blocks. These routines are presented in Chapter 7. The framework which puts all the subroutines of the local clustering algorithms together is discussed in Chapter 4.

- In order to evaluate the quality of the segmentations constructed by the various algorithms, we built a simulation program in Java. The program both visualises the process, yielding structural insights, and derives the approximate values of the performance measures presented in the previous section. A detailed description of the program is given in Chapter 8.

- Chapter 9 presents the comparison we made between the old and newly developed algorithms in terms of network performance. Naturally, we hope to verify that the adjustments we made to the local algorithms indeed lead to better solutions to the segmentation problem.
Chapter 2

Preliminaries

2.1 Random walks on graphs

2.1.1 Basic notions

A graph is a structure consisting of objects, generally referred to as vertices and links between pairs of objects, called edges. The usual notation is $G = (V, E)$, $V$ representing the vertices, $E$ being the edges. The edges of $G$ are denoted by pairs $(u, v)$ for $u, v \in V$ and we write $u \sim v$ if and only if $(u, v) \in E$. If $u \sim v$ then $v$ is considered a neighbour of $u$, and vice versa and the degree of a vertex $v$ is equal to the number of neighbours of $v$, denoted by $d(v)$. Equivalently, a graph is defined by its adjacency matrix $A_G$, an $n \times n$-matrix with

$$A_G(i, j) = \begin{cases} 
1, & \text{if } (i, j) \in E, \\
0, & \text{else}.
\end{cases}$$

Clearly, $A_G$ is symmetric, and the sum of row $u$ is equal to $d(u)$. On top of this structure, the edges of the graph can be endowed with weights. In this case a weight function $w : E \to \mathbb{R}$ is defined on all edges, yielding a so-called weighted graph. Up until here, only undirected graphs are considered, that is, $u \sim v$ if and only if $j \sim i$. However, the concept can be extended to using one way connections. Then, the set of edges $E$ is replaced by a set of arcs $A$, consisting of pairs of vertices $\{u, v\}$, $u, v \in V$, not necessarily satisfying the condition $\{u, v\} \leftrightarrow \{v, u\}$. If this is the case, we call the graph $G = (V, A)$ directed. Naturally, the concept of the weight function can be extended to directed graphs, although we will not go deeper into that here, for we will mostly focus on undirected graphs in this thesis.

Let a graph $G = (V, E)$ be given, together with a starting vertex $v \in V$. Looking at neighbours of $v$, we select one of them at random and move there. In the next step, the a neighbour of this vertex is chosen at random and we move to this vertex. The sequence of vertices of the graph visited in this manner is called a random walk on the graph. There are several types of random walks that can be defined on a graph. A few variations will be discussed in the next paragraph. All of them work with the following general framework. Formalising the notion of a random walk on a graph, let $G = (V, E)$ be a graph, and denote the starting vertex by $v_0$. The vertex visited in the $t$th step is referred to as $v_t$ and the $k$-step sample path is represented by $\{v_0, v_1, ..., v_k\}$. The transition probability of moving from vertex $i$ to vertex $j$, is denoted by $p(i, j)$. Clearly, the $p(i, j)$ must satisfy

$$\sum_{j=1}^{n} p(i, j) = 1 \quad \forall j \in \{1, ..., n\},$$

where $n = |V|$, labeling the vertices by natural numbers. The $n \times n$ transition matrix $M$ contains the transition probabilities as entries corresponding to the position. Note that the
$p(i, j)$’s only depends on the current position $i$, and not on the past trajectory of the walk. This independence suggests that there is a relation between Markov chains and random walks on a graphs. This is why a random walk on an undirected graph is actually a finite Markov chain that is time-reversible and similarly, any time-reversible Markov chain can be viewed as a random walk on an undirected graph.

The starting vertex $v_0$ and transition matrix $M$ are the basic elements needed to describe the entire random walk process. Let the distribution of $v_t$, starting in $v_0$, for $t \geq 1$ by denoted by

$$P_t^{v_0}(i) = \mathbb{P}(v_t = i). \quad (2.3)$$

Let $e_i$ be the unit vector of length $n$ only having a 1 on the $i$th position. Then,

$$P_1^{v_0}(i) = p(v_0, i) = \left(M^T e_{v_0}\right)_i, \quad (2.4)$$

and upon iteration,

$$P_t^{v_0}(i) = \sum_{j \in V} P_{t-1}^{v_0}(j) p(j, i) = \sum_{j \in V} \left(M^T\right)^{t-1} p(j, i) = \left(M^T\right)^t e_{v_0}, \quad (2.5)$$

for any $t \in \mathbb{N}$. In vector notation, with $P_t^{v_0} = (P_t^{v_0}(1), \ldots, P_t^{v_0}(n))$,

$$P_t^{v_0} = M^T P_{t-1}^{v_0} = \left(M^T\right)^t e_{v_0}, \quad (2.6)$$

and more generally, if the starting vertex is selected according to a distribution $P_0$,

$$P_t = \left(M^T\right)^t P_0 \quad (2.7)$$

### 2.1.2 Variations

We now discuss several types of random walks on an undirected graph $G = (V, E)$, each leading to a different transition matrix. Given an undirected graph $G = (V, E)$ there are multiple ways to define a random walk on $G$. The walks differ through the choice of the transition matrix $M$. A few commonly used types of random walks will be discussed in the next few lines. First, some notation from graph theory in needed. Let $D$ denote the $n \times n$ diagonal matrix with $D_{ii} = d(i)$.

The most classical walk to be defined on $G$ is the simple random walk. The idea behind this walk is intuitive. Before each step, take a look at the neighbours and choose one of them uniformly at random and move to this vertex. Hence, the transition probability for this walk is defined as

$$p(i, j) = \begin{cases} 1/d(i), & \text{if } (i, j) \in E, \\ 0, & \text{otherwise} \end{cases} \quad (2.8)$$

Or equivalently,

$$M = D^{-1} A_G \quad (2.9)$$

A closely related random walk is the lazy random walk. This is more reluctant to move away from its current position and first flips a coin to decide whether to stay or leave the vertex. If
it decides to leave, then it moves to either one of its neighbours with equal probability, just as the simple random walk. This yields the transition probabilities.

\[
p(i, j) = \begin{cases} 
1/2, & \text{if } i = j, \\
1/(2d(i)), & \text{if } (i, j) \in E, \\
0, & \text{otherwise,}
\end{cases}
\]  

(2.10)

and hence,

\[
M = \frac{1}{2} \left( I + D^{-1}A_G \right).
\]  

(2.11)

For some special types of graphs, the transition matrices are easy to determine. For instance, if \( G \) is \( k \)-regular, i.e. every vertex has degree \( k \), one finds that \( D = kI \), where \( I \) is the \( n \times n \) identity matrix. Hence on \( k \)-regular graphs, the transition matrices of the simple and lazy walk simplify to \( \frac{1}{k}A_G \) and \( \frac{1}{2} \left( I + \frac{1}{k}A_G \right) \), respectively.

We next introduce a random walk that is applicable to weighted undirected graphs. Let \( w(i, j) \) denote the weight of edge \( (i, j) \in E \) with \( w \in \mathbb{R}_+ \) and \( W_G \) the \( n \times n \) matrix storing these values. Let \( w(i) \) be the total weight of a vertex \( i \) defined as

\[
w(i) = \sum_{j \in V: (i, j) \in E} w(i, j),
\]  

(2.12)

and let \( D_w \) be the \( n \times n \) diagonal matrix with \( (D_w)_{ii} = w(i) \). The transition probabilities of a \textit{biased simple random walk} are

\[
p(i, j) = \begin{cases} 
w(i, j)/w(i), & \text{if } (i, j) \in E, \\
0, & \text{otherwise,}
\end{cases}
\]  

(2.13)

and the corresponding transition matrix equals

\[
M = D_w^{-1}W_G.
\]  

(2.14)

Note that the random walk no longer moves uniformly to one of its neighbours. Instead, the probability of moving to one particular neighbour is proportional to the weight of the edge connecting the two vertices. This generalisation naturally extends to the \textit{biased lazy random walk} in which case the transition probabilities are given by

\[
p(i, j) = \begin{cases} 
1/2, & \text{if } i = j, \\
w(i, j)/(2w(i)), & \text{if } (i, j) \in E, \\
0, & \text{otherwise,}
\end{cases}
\]  

(2.15)

whereas the transition matrix is

\[
M = \frac{1}{2} \left( I + D_w^{-1}W_G \right).
\]  

(2.16)

By assigning equal weights to all edges, transition probabilities again reduce to the probabilities of the unbiased walks. The biased random walk can therefore be seen as the more general type. For this reason, statements in the remainder of this section will be proven for the biased version. We will omit the subscript in \( A_G \) and \( W_G \) if it is clear which graph \( G \) is meant.
2.1.3 Stationary distribution

Fix a transition matrix $M$ corresponding to a random walk on the graph $G = (V, E)$. The following concept is essential in the theory of Markov chains.

**Definition 2.1 (Stationary distribution).** Let $M$ be a transition matrix of a Markov chain. If $\pi$ is a distribution on $V$ satisfying

$$\pi = M^T \pi$$

or equivalently,

$$\pi(i) = \sum_{j \in V} \pi(j) p(j, i),$$

then $\pi$ is called a stationary distribution of the Markov chain.

The stationary distribution is sometimes also referred to as the steady-state distribution. As the name suggests, whenever the process is in stationary distribution, it stays there. It is easy to see that if the process is in steady-state after $k$ steps, hence $p^k = \pi$, then for all $t \geq k$

$$p^t = (M^T)^t p_0 = (M^T)^{t-k} (M^T)^k p_0 = (M^T)^{t-k} \pi = \pi$$

by (2.7) and (2.17). More specifically, if $P_0 = \pi$, then $p^t = p_0$ for all $t \geq 1$. This is called the stationary walk. The next proposition proves the existence of the stationary distribution for biased random walks on a graph, following the more general result given in Chapter 1 of [LPW06].

**Proposition 2.1.** Let $G = (V, E)$ be a connected weighted undirected graph with the biased simple random walk as in Equation (2.13) defined on it. Then the unique stationary distribution $\pi$ of the random walk exists and satisfies

$$\pi(i) = \frac{w(i)}{\sum_{j \in V} w(j)}.$$  \hspace{1cm} (2.19)

**Proof.** For $\pi$ to be a stationary distribution, it needs to satisfy,

$$\pi(i) = \sum_{j \in V} \pi(j) p(j, i) \quad \text{and} \quad \sum_{i \in V} \pi(i) = 1.$$  \hspace{1cm} (2.20)

Clearly, the second equality is met, since substituting (2.19) into (2.20) gives

$$\sum_{i \in V} \pi(i) = \sum_{i \in V} \frac{w(i)}{\sum_{j \in V} w(j)} = 1.$$  \hspace{1cm} (2.21)

Upon again using (2.19) we get

$$\sum_{j \in V} \pi(j) p(j, i) = \sum_{j \in V} \frac{w(j)}{\sum_{k \in V} w(k)} \cdot \frac{w(j, i)}{w(j)} = \frac{1}{\sum_{k \in V} w(k)} \sum_{j \in V} w(j, i) = \frac{w(i)}{\sum_{k \in V} w(k)} = \pi(i),$$

where the last equality follows from symmetry of the weight function. Thus $\pi$ is a stationary distribution of the random walk indeed. For proving uniqueness, suppose there are two stationary distributions, $\pi$ and $\pi^*$. By assumption, we have $\pi = M^T \pi$ and $\pi^* = M^T \pi^*$. Since $G$ is connected, for every $i \in V$, there
exists a $j$ such that $p(i, j) > 0$. Let $h$ be a vector in $\mathbb{R}^n$ such that $h = Mh$. We will prove that $h$ is necessarily constant. For $h = Mh$ to be true, one finds that

$$h(i) = \sum_{j \in V} p(i, j) h(j), \quad \forall i \in V. \quad (2.23)$$

Let $k \in V$ be such that $h(k) = H = \max_{j \in V} h(j)$. By connectivity of $G$, there exists a $j \in V$, such that $p(k, j) > 0$ and assume $h(j) < H$. Then we find

$$h(k) = \sum_{i \in V} p(k, i) h(i) = p(k, j) h(j) + \sum_{i \neq j} p(k, i) h(i) < H \quad (2.24)$$

which is a contradiction. Thus $h(j) = H$ for all $j \in V$ that share an edge with $k$. This argument can be repeated by applying the result to the neighbours of the neighbours of $k$, etcetera, and again by the connectivity of $G$, it has to hold that $h(i) = H$ for all $i \in V$, i.e. $h$ is constant.

The fact that $h = Mh$ if and only if $h$ is constant implies that $M$ has only one eigenvector with eigenvalue 1. Now use the fact that the spectrum of a matrix is equal to the spectrum of its transpose. So $M^T$ also has eigenvalue 1 with multiplicity one. Since $\pi$ and $\pi^*$ are both eigenvectors of $M^T$ with eigenvalue 1, it follows that $\pi$ and $\pi^*$ are equal up to a constant. However, because we are dealing with distributions, we have $\sum_{i \in V} \pi(i) = \sum_{i \in V} \pi^*(i) = 1$ and hence it needs to hold that $\pi = \pi^*$, proving uniqueness of $\pi$.

Proposition 2.1 only considers the simple biased random walk case. An easy trick, though, shows that the stationary distributions of the (biased) simple and lazy random walk coincide. To see this, note that the stationary distribution of the lazy random walk satisfies

$$\pi(i) = \sum_{j \in V} \pi(j) \frac{w(j, i)}{w(j)} = \frac{1}{2} \pi(i) + \frac{1}{2} \sum_{j \in V, j \neq i} \pi(j) p(j, i) \Rightarrow \pi(i) = \sum_{j \in V, j \neq i} \pi(j) \frac{w(j, i)}{w(j)} \quad (2.25)$$

Because $p(i,i) = 0$ and $p(i,j) = w(i,j)/w(i)$ for the simple random walk, the linear system of equations for finding the stationary distribution of the lazy random walk reduces to the one of the simple walk, resulting in the same solution $\pi$.

In the special case of $G$ being an unweighted $k$-regular graph, it follows that the stationary distribution for both the simple and lazy random walk is uniform on $V$.

### 2.1.4 Limiting probabilities and mixing times

Intuition tells us that the $t$-step distribution $p^t$ of the random walk tends to its stationary distribution, independent of the starting distribution. This idea, generalised for Markov chains, is formalised in Theorem 2.1 and is an elementary result given in many introductory books on Markov chains, for example [Ros06]. Before giving the statement, we need three more notions for Markov chains. Let $p^t_i(j)$ be the probability of a Markov chain being in state $j$ after $t$ steps, starting in state $i$.

**Definition 2.2** (Irreducibility of a Markov Chain). A Markov chain is said to be irreducible if for all pairs, $i, j \in V$, there exists a $t \in \mathbb{N}$ such that $p^t_i(j) > 0$.

Informally, irreducibility of a Markov chain implies that there is a path between all states of the chain. In case of a random walk defined on a graph, irreducibility of the corresponding Markov chain is equivalent to the graph being connected.

**Definition 2.3** (Periodicity of a Markov Chain). A state $i$ of a Markov chain is said to have period $d$, if $p^t_i(i) = 0$ for $t$ not divisible by $d$. If all states of the chain have period 1, the chain is said to be aperiodic and otherwise the Markov chain is periodic.
Working with a simple random walk on $G$, either biased or unbiased, the Markov chain corresponding to the walk is periodic if and only if $G$ is $k$-partite where $k$ is even. In case of the lazy walk, the chain is always aperiodic. Another notion often used in Markov theory is positive recurrence. In words, for finite-state Markov chains, positive recurrence means that each state is visited infinitely often as $t \to \infty$. Since the graphs considered in this thesis are finite, the underlying Markov chain is also. Positive recurrence of the random walks therefore follows directly from irreducibility. Hence the random walks we discuss are always positive recurrent. Markov chains that are both positive recurrent and aperiodic are called ergodic. We are now ready to state the following theorem as given in [Ros06].

**Theorem 2.1.** For an irreducible, ergodic Markov chain the stationary distribution $\pi$ exists and

$$\lim_{t \to \infty} p_t^i(j) = \pi_j,$$  \hspace{1cm} (2.26)

independent of the starting state $i$.

For the proof the reader is referred to paragraph 4.4 of [Ros06]. Applying the theorem to our case of random walks on graphs, the next result naturally follows.

**Corollary 2.1.** Let $G = (V, E)$ be a connected weighted undirected graph with weights $w(i, j)$. Then for both the biased simple and lazy random walk defined on $G$,

$$\lim_{t \to \infty} p_t^i(j) = \pi_j = \frac{w(j)}{\sum_{k \in V} w(k)},$$  \hspace{1cm} (2.27)

independent of the starting vertex $i$.

The corollary does not give information about how fast the $t$-step distribution converges to the stationary distribution. The next two concepts are related to this question.

**Definition 2.4 (Total variation distance).** Let $\mu$ and $\nu$ be two probability measures on $\Omega$. The total variation distance between $\mu$ and $\nu$ is defined as

$$\|\mu - \nu\|_{TV} = \max_{A \subseteq \Omega} |\mu(A) - \nu(A)|.$$  \hspace{1cm} (2.28)

If $\mu$ and $\nu$ are discrete measures, this definition is equivalent to

$$\|\mu - \nu\|_{TV} = \frac{1}{2} \sum_i |\mu(i) - \nu(i)|.$$  \hspace{1cm} (2.29)

**Definition 2.5 (Mixing time).** Let $p_t^i$ be the $t$-step distribution of a Markov chain starting in state $i$, $\pi$ the stationary distribution and $\epsilon > 0$. Then the $\epsilon$-mixing time of the Markov chain is given by

$$\tau(\epsilon) = \min \{ t : \max_i \|p_t^i - \pi\|_{TV} \leq \epsilon \}.$$  \hspace{1cm} (2.30)

The concept of mixing time of a Markov chain is closely related to properties of the underlying graph, such as conductance and escape probabilities, which will be introduced in the next section.
2.2 Graph partitioning

2.2.1 Building blocks and classical objectives

Graph partitioning is the task of grouping the vertices of a graph into subsets taking into account the edge structure in such a way that vertices are well-connected within the partition and contain relatively few edges between the partitions. Applications of the partitioning of graphs can be found in many unexpected areas. For instance, one can define a graph on the users of a social network, sharing edges when befriended, to detect social communities. Another example is image segmentation, where pixels can be represented by vertices and the edges, possibly with weights, connect pixels that are similar to some extend. Graph partitioning then translates to segmentation of structures in pictures. In this thesis, we use the connectivity graph of a street light deployment, discussed in Chapter 1, where the vertices represent the individual street lights and the (undirected) edges indicate which pairs of lights are able to communicate with each other, as a primary example. In this case, partitioning the connectivity graph is equivalent to finding a segmentation of the street light deployment. Because of the great amount of interest and relevance in real world problems, graph partitioning is among the most active research studies in graph theory. First, let us give the mathematical definition of a partition of a graph.

**Definition 2.6** (Partition of a Graph). Let $G = (V, E)$ be graph. A collection of subsets of the vertices $V$ given by $\{V_1, \ldots, V_k\}$ is said to be a partition of $G$ if $\bigcup_{i=1}^k V_i = V$ and $V_i \cap V_j = \emptyset$ for all $i, j \in \{1, \ldots, k\}$, $i \neq j$.

In literature, partitioning is also referred to as clustering or segmentation of graphs. From the definition one can see that the number of possible partitions of a graph is enormous. However, it is not yet clear what a ‘good’ partition looks like. The definition of the quality of a partition differs through the choice of the objective function. The most commonly used objectives are based on the next two properties of the subsets of the vertices of a graph.

**Definition 2.7** (Volume and Cut). Let $G = (V, E)$ be an undirected graph and $S, T \subseteq V$. Let $d(u)$ denote the degree of vertex $u$. Then the volume of $S$ is defined as

$$\mu(S) = \sum_{u \in S} d(u).$$

The number of edges between $S$ and $T$ is given by

$$E(S, T) = |\{(u, v) \in E | u \in S, v \in T\}|.$$  

Furthermore, the cut of the set $S$ is defined as

$$\partial(S) = E(S, S^c),$$

where $S^c$ denotes the complement of $S$ in $V$ and hence $\partial(S) = \partial(S^c)$.

There are two main types of partitioning criteria. The first, called two-way partitioning, induces the second, multiway (or $k$-way) partitioning. As the name suggests, two-way partitioning considers dividing up the graph into two clusters, according to the choice of the objective function. The following three objective functions for two-way partitioning are the ones studied most in literature:

$$\text{MinCut}(S) = \partial(S),$$
Graph partitioning with the MinCut objective is the most famous version of the above. It is closely related to the flow going through the network by the max-flow min-cut theorem. The Ford-Fulkerson algorithm solves the maximum flow, and accordingly the two-way partitioning problem with minimum cut objective, in polynomial time, see [Vaz01], and therefore graph partitioning with MinCut objective is in P.

Using the MinCut objective, the optimal bipartition might consist of one subset with the great majority of vertices and the other with just a few vertices. If a two-way partition of somewhat similar size is preferred, the RatioCut criteria is a better option. Partitions found using this object will be more balanced, since the optimal value the term \( \frac{\partial(S)}{|S|} + \frac{\partial(S_c)}{|S_c|} \) is minimized for \( |S| = |V|/2 \). Therefore, by choosing this objective function a trade-off is done between the balancing of the sizes and the cut. As opposed to partitioning by the MinCut objective, solving the bipartitioning problem using the RatioCut objective for an arbitrary graph can be shown to be NP-hard.

The NormalisedCut objective is similar to the RatioCut for it tries to find a balanced partition. However, in this case the volumes of \( S \) and \( S^c \) are compared instead of the respective sizes. Partitions are hence more balanced in the sense of edges, and the solutions to this problem typically consist of two clusters of vertices which are well-connected within and have relatively few edges connecting the two clusters. This idea will come back later in this chapter, where we the concept of conductance is formally introduced.

So far, only two-way partitioning problems have been considered. Of course, the objectives discussed above have equivalent forms for the multiway partitioning of graphs. Let \( \{S_1, ..., S_k\} \) be a \( k \)-way partition of the graph.

\[
\text{MinCut}(S_1, ..., S_k) = \sum_{i=1}^{k} \partial(S_i)
\]

\[
\text{RatioCut}(S_1, ..., S_k) = \sum_{i=1}^{k} \frac{\partial(S_i)}{|S_i|}
\]

\[
\text{NormalisedCut}(S_1, ..., S_k) = \sum_{i=1}^{k} \frac{\partial(S_i)}{\mu(S_i)}
\]

Note that for all of these objectives, the number of sets in the partition \( k \) is predefined and not an optimisation variable. Even though bipartitioning with the MinCut objective is still possible to do in polynomial time, multiway partitioning of an arbitrary graph is NP-hard for all of the objectives above. Hence, no efficient methods are known, or more precisely, do not exist unless P=NP, that solve the problem optimally. Nevertheless, a lot of research has been done on finding approximation algorithms and heuristics to approximate the optimal
solution. One way of finding a \( k \)-way partition is to apply an algorithm for bipartitioning a graph iteratively on the solution of the previous partition until \( k \) subsets are found. For this method called recursive bipartitioning, one can for instance use the Ford-Fulkerson algorithm as the subroutine for constructing a reasonable \( k \)-way partition. Other examples of well-studied heuristics are \( k \)-means, \( k \)-medoids and agglomerative clustering. A more recent method called spectral clustering uses properties of the graph in terms of eigenvalues to determine good quality partitions of the vertices. An alternative clustering method is to first construct a collection of good quality clusters, together covering the entire graph, and apply a set covering algorithm afterwards, allowing overlaps. Both this technique and the subject of spectral clustering will be discussed thoroughly later on in this thesis.

2.2.2 Laplacian of a graph

In this section we consider an undirected graph \( G = (V, E) \) and assume for now that the edges all have weight one. Spectral graph theory aims study of the so-called Laplacian of a graph. The graph Laplacian is an \( n \times n \)-matrix which captures certain properties of the corresponding graph. In the literature, there is no unique definition of ‘the’ Laplacian of a graph although the matrix is always related to the adjacency matrix. We give a concise definition of the most general Laplacian below, as given in [vL07].

Definition 2.8 (Unnormalised graph Laplacian). Let \( G = (V, E) \) be an undirected graph, \( A \) its adjacency matrix and \( D \) the \( n \times n \) diagonal matrix with \( D_{ii} = d(i) \). The unnormalised Laplacian of a graph is defined as

\[
L = D - A
\]  

(2.40)

From this definition some properties of the matrix follow immediately.

Proposition 2.2 (See [vL07] p.4). Let \( G = (V, E) \) be an undirected graph with adjacency matrix \( A = (a_{ij}) \). The unnormalised graph Laplacian \( L \) has the following properties

1. For all \( x \in \mathbb{R} \),

\[
x^T L x = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} (x_i - x_j)^2. \tag{2.41}
\]

2. \( L \) is symmetric and positive semi-definite.

3. The smallest eigenvalue of \( L \) is 0, with corresponding eigenvector the constant one vector \( \mathbb{1} \).

Proof. (1) By definition of \( L \) we find

\[
x^T L x = x^T Dx - x^T Ax = (x_1, ..., x_n) \begin{pmatrix} d(1)x_1 \\ \vdots \\ d(n)x_n \end{pmatrix} - (x_1, ..., x_n) \begin{pmatrix} \sum_{j=1}^{n} a_{1j}x_j \\ \vdots \\ \sum_{j=1}^{n} a_{nj}x_j \end{pmatrix} = \sum_{i=1}^{n} d(i)x_i^2 - \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}x_ix_j = \frac{1}{2} \left( \sum_{i=1}^{n} d(i)x_i^2 - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}x_ix_j + \sum_{j=1}^{n} d(j)x_j^2 \right) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}(x_i - x_j)^2, \tag{2.42}
\]
where, in the last step, we use that
\[ d(i) = \sum_{j} a_{ij} = \sum_{j} a_{ji}. \]

(2) Symmetry of \( L \) follows by symmetry of both the adjacency matrix \( A \) and \( D \), being a diagonal matrix. The fact that \( L \) is positive semi-definite follows by the fact that \( x^T L x \) is a weighted sum of squares, see (2.41), and thereby non-negative, for all \( x \in \mathbb{R}^n \).

(3) Since \( L \) is positive semi-definite, all eigenvalues of \( L \) are non-negative, implying that the smallest possible eigenvalue is 0. Taking \( x = 1 \) implies that \( x_1 = \ldots = x_n = 1 \) and by (2.41) we again find
\[ 1^T L 1 = 0 \]
by which we can conclude that \( 1 \) is indeed an eigenvector of \( L \) with eigenvalue 0. \( \square \)

The eigenvectors and eigenvalues of \( L \) can be used to describe several properties of the graph \( G \). The key observation is that an eigenvector corresponding to the eigenvalue zero can be seen as an indicator function that detects components of the graph. The next proposition summarises this idea.

Proposition 2.3 (See [vL07] p.4). Let \( L \) be the graph Laplacian of an undirected graph \( G = (V, E) \). If the multiplicity of eigenvalue 0 of \( L \) is \( k \), then \( G \) has \( k \) connected components. Furthermore, if \( A_1, \ldots, A_k \subseteq V \) are the connected components of \( G \), then the eigenvectors with eigenvalue 0 are the indicator vectors \( 1_{A_1}, \ldots, 1_{A_k} \), where \( (1_A)_i = 1 \) if \( i \in A \) and 0 otherwise.

Proof. First write (2.41) as
\[ x^T L x = \frac{1}{2} \sum_{(i,j) \in E} (x_i - x_j)^2 \] (2.43)

We know that there is at least one eigenvector with eigenvalue zero. Now assume eigenvalue 0 has multiplicity \( k \), hence there are \( k \) linearly independent eigenvectors \( x^{(m)} \) for \( m = 1, \ldots, k \) such that \( x^{(m)^T L x^{(m)}} = 0 \), or equivalently
\[ \frac{1}{2} \sum_{(i,j) \in E} \left( x_i^{(m)} - x_j^{(m)} \right)^2 = 0. \] (2.44)

This in turn implies that \( x_i^{(m)} = x_j^{(m)} \) for all \( (i, j) \in E \). Thereby, we can deduce that the value \( x_i^{(m)} \) assigned to vertex \( i \) is equal to the value of all of its neighbours, and recursively the neighbours of its neighbours, etcetera. Consequently, \( x^{(m)} \) is constant on each of the connected components of \( G \) and thus is of the form
\[ x^{(m)} = \alpha_1 1_{A_1} + \ldots + \alpha_k 1_{A_k}, \quad \alpha_i \in \mathbb{R}. \] (2.45)

Without loss of generality we can assume that the \( A_i \)'s are disjoint, so that \( A_i \) contains, and hence equals, one connected component of \( G \). So \( G \) has \( k \) connected components. The reverse statement is now trivial. \( \square \)

In the literature, often normalised graph Laplacians are used. The most commonly used types are the symmetric and the random walk Laplacian.

Definition 2.9. Normalised Laplacian of a graph
Let \( G = (V, E) \) be an undirected graph with adjacency matrix \( A \) and diagonal degree matrix \( D \). The symmetric graph Laplacian is defined as
\[ L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}. \] (2.46)
2.2. GRAPH PARTITIONING

The random walk graph Laplacian is defined as

\[ L_{rw} = D^{-1}L = I - D^{-1}A. \]  

(2.47)

Note that there is some similarity between the random walk Laplacian and the transition matrix of a random walk on \( G \) in (2.11), although there is some difference in signs. It can be seen that the smallest eigenvalue of both the symmetric and random walk Laplacian is again zero, with corresponding eigenvectors \( D^{1/2}1 \) and \( 1 \), respectively.

Let us denote the eigenvalues of the graph Laplacian, by \( \lambda_1, \ldots, \lambda_n \) such that \( 0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \) and assume the graph \( G \) is connected, hence the multiplicity of eigenvalue 0 is 1, so that \( \lambda_2 > 0 \). Since the eigenvector corresponding to the eigenvalue 0 could be seen as an indicator vector for the components, the eigenvector associated with eigenvalue \( \lambda_2 \) seems a good approximation of the indicator function for a partition of the graph. This idea will be exploited in Chapter 3. The second eigenvector of the graph Laplacian is known as the Fiedler vector. Another notion that will be used a lot in the remainder of this thesis is the spectral gap of a graph, which is equal to the difference between the first and second eigenvalue of the graph Laplacian, i.e. \( \lambda_2 - \lambda_1 = \lambda_2 \), assuming that the graph is connected.

2.2.3 Conductance

A concept which brings together spectral graph theory and random walks on graphs is conductance.

**Definition 2.10 (Conductance).** Let \( G = (V, E) \) be an undirected graph and \( S \subseteq V \). The conductance of the set \( S \) is

\[ \varphi(S) = \frac{\partial(S)}{\min\{\mu(S), \mu(S^c)\}}. \]  

(2.48)

The conductance of the graph \( G \) is given by

\[ \varphi_G = \min_{S \subseteq V} \varphi(S). \]  

(2.49)

Note that \( \varphi(S) \in [0, 1/2] \) for all \( S \subseteq V \). Some slightly different definitions appear in literature, such as

\[ \varphi(S) = \frac{\partial(S)}{\mu(S)}, \]  

(2.50)

although these definitions will coincide for the settings we focus on in this thesis. This is because we will usually see \( S \) as one set of a multiway partition of \( G \), hence \( |S| \ll |V| \). Sometimes, conductance of a graph is also referred to as the bottleneck ratio or Cheeger’s constant. Notice the resemblance between the conductance of set \( S \) and the NormalisedCut objective for graph partitioning. Indeed solving the graph partitioning problem according to the NormalisedCut objective will yield the partition with smallest average conductance.

Because \( \varphi(S) \) resembles the ratio between the number of edges leaving \( S \) and the total number of connections within \( S \), subsets of low conductance will have few edges between \( S \) and its complement, compared to the number of edges connecting the vertices inside the set. Furthermore, the volume of \( S \) is of course related to the size of \( S \). Informally, we could say that a set of low conductance has the property of being of considerable size, well-connected inside and poorly connected to the outside vertices. A toy example of a graph with low conductance is depicted in Figure 2.1.
Figure 2.1: Toy example with $\phi(G) = \phi(\{1, 2, 3, 4\}) = 1/13$.

From a random walk point of view, starting in a vertex that is contained in a set $S$ of low conductance intuitively implies that the probability of escaping $S$ is relatively low, whereas the particle stays in $S$ with high probability, due to the fact that $S$ is well-connected within. The graph in Figure 2.1 demonstrates this nicely. A simple calculation shows that the $S = \{1, 2, 3, 4\}$ is a set of minimum conductance (hence also $\{5, 6, 7, 8, 9\}$ is). Choosing either one of the vertices in $S$ as the starting point, the random walk can only leave $S$ by visiting vertex 3 and jumping to 5 immediately in the next step. However, being in 3, the probability of leaving $S$ is 1/4 for the simple random walk, and even 1/8 for the lazy random walk, while the probability of staying in $S$ is 3/4 or 7/8, respectively. The probability of those two events happening consecutively is hence very low. This is caused by the presence of a so-called bottleneck. The easiest interpretation of a bottleneck of a graph is the following. Considering the shortest paths between all pairs of vertices, then a bottleneck of the graph is an edge, or set of edges, contained in an increased portion of those path. Indeed, the edge between vertex 3 and vertex 5 is a clear bottleneck. This description justifies the synonym of the conductance measure.

A consequence of the fact that the probability to escape a low conductance set after a few steps is small, is that $P^t_{\pi}(j)$ for $i \in S, j \in S^c$. I.e. the probability of being in $j$ after $t$ steps starting from $i$, is very small for $t$ small enough. Recall that $\pi(j) = d(j) / \sum_i d(i)$ for the unbiased random walk, which implies that $\|P^t - \pi\|_{TV}$ is large. It follows that the mixing time of the underlying Markov chain, $\tau(\varepsilon)$, is relatively large. Another way to say this is that a Markov chain does not admit rapid mixing if the graph contains clear bottlenecks. Hence, it is obvious that there is a relation between the conductance of a graph and the mixing time of the Markov chain associated with the random walk on the graph.

The conductance of a graph $G$ can be seen as a measure of whether a graph contains a bottleneck or not. Since the optimisation problem of finding a set of vertices in a graph is NP-hard (it is a more general version of the NormalisedCut partitioning problem), the value $\phi_G$ is not known beforehand. Nevertheless, a discretisation of Cheeger’s inequalities, which was originally applied to manifolds in differential geometry, leads to a relation between $\phi_G$ and the second smallest eigenvalue of the Laplacian in the following central theorem.

**Theorem 2.2. Cheeger Inequalities**

Let $G = (V, E)$ be an undirected, unweighted graph with conductance $\phi$, $L_{sym}$ its normalised symmetric Laplacian and $\lambda_2$ the second smallest eigenvalue of $L_{sym}$. Then the following inequalities hold:

$$\frac{\phi_G^2}{2} \leq \lambda_2 \leq 2\phi_G.$$  (2.51)
2.3. EVOLVING SETS

Of course, the Cheeger inequalities can be rewritten in such a way that we derive an upper and lower bound on the conductance of $G$.

$$\frac{\lambda_2}{2} \leq \varphi_G \leq \sqrt{2\lambda_2}. \quad (2.52)$$

To check that the bounds indeed make sense, consider the case in which $G$ consists of two connected components. By Proposition 2.3, $\lambda_2$ must be equal to zero. Theorem 2.2 states that $\varphi_G = 0$, which is indeed true, since choosing either one of the two components as the required subset $S$ of $V$ gives a cut of value 0, hence a subset of conductance zero. As discussed in the previous section, the spectral gap $\lambda_2$ is a measure of separation of the vertices. The smaller the value $\lambda_2$ the better the graph can be divided in clusters. From the upper bound in (2.52) it can be seen that also the conductance of $G$ decreases as $\lambda_2$ approaches zero, confirming the intimate relation between the spectral gap and the conductance of a graph. Conductance will be the central objective function for the partitioning graphs in this thesis.

2.3 Evolving sets

2.3.1 Evolving set process on graphs

We return to the theory of random walks on graphs. Suppose we have defined an unbiased lazy random walk on an unweighted graph $G = (V, E)$, with transition probabilities as given in (2.10). Then for $S \subseteq V$,

$$p(y, S) = \sum_{x \in S} p(y, x)$$

(2.53)

denotes the probability of entering $S$ in the next step, while currently staying in vertex $y$. $\pi(S) = \sum_{x \in S} \pi(x)$ represents the stationary probability of being in $S$, in other words, asymptotically $\pi(S)$ is the fraction of time resided in $S$. Since we are dealing with the unbiased lazy random walk, a more explicit form of $p(y, S)$ can be derived.

$$p(y, S) = \sum_{x \in S} p(y, x) = p(y, y) \mathbb{1}_{y \in S} + \sum_{\substack{x \in S \atop x \sim y}} p(y, x)$$

(2.54)

$$= \frac{1}{2} \mathbb{1}_{y \in S} + \sum_{\substack{x \in S \atop x \sim y}} \frac{1}{2d(y)} = \frac{1}{2} \left( \mathbb{1}_{y \in S} + \frac{E(S, \{y\})}{d(j)} \right). \quad (2.55)$$

The evolving set process (ESP) is a Markov chain on the subsets of $V$ first described by Morris and Peres in [MP03] for the state space graph of a Markov chain. The ESP adapted to graphs is studied in [AP08]. Suppose the current state is $S$ and let $U$ be a uniform random variable on $[0, 1]$. The next state of the chain is

$$S' = \{y \in V | p(y, S) \geq U\}. \quad (2.56)$$

Hence, the next set of the chain contains only vertex which have probability greater than $U$ of entering (or staying) in $S$. A few properties of the ESP can be derived immediately from this definition.

- If $U > 1/2$, one finds that

$$y \in S' \Rightarrow p(y, S) > 1/2. \quad (2.57)$$

Since $p(y, S) \leq 1/2$ if $y \in S^c$, this implies $S' \subseteq S$. 


• If \( U \leq 1/2 \), clearly \( p(y, S) \geq 1/2 \) for all \( y \in S \), since \( p(y, y) = 1/2 \). Hence \( S \subseteq S' \).

• The transition probabilities of for the Markov chain of the ESP are given by

\[
K(S, S') = \mathbb{P} \left( p(y, S) \geq U \ \forall y \in S', \ p(y, S) < U \ \forall y \in S^c \right)
\]

\[
= \mathbb{P} \left( \max_{y \in S'} p(y, S) < U \leq \min_{y \in S'} p(y, S) \right) = \max \left\{ 0, \min_{y \in S'} p(y, S) - \max_{y \in S'} p(y, S) \right\}
\]

(2.59)

since \( U \) is uniform on \([0,1]\).

### 2.3.2 Connection to conductance

Let us we take a deeper look into the choice of the next set in the evolving set process. A vertex \( y \), not in \( S \), has probability \( p(y, S) \) of being included in the next set. From Equation (2.55), we observe that this probability increases as the number of edges linking \( y \) with vertices in \( S \) increases. Alternatively, if only a small portion of the edges incident to \( y \) connect the vertex to \( S \) and more connections to the complement of \( S \), it is likely to be left out. Similarly, if \( y \) is already in \( S \), and it has weak connections to the rest of \( S \), the probability of \( y \) being contained in the next set is low, whereas vertices in \( S \) that are well-connected to the other vertices will most probably be kept, possibly even with probability 1 (if all neighbours of \( y \) are also in \( S \)). From these observations we can deduce that the process naturally searches for a subset of the volume in the ESP. The proof given here is similar to the proof given in [LPW06], but adapted to graphs. Let \( \mathbb{P}_S(\cdot) \) denote the probability measure for the ESP Markov chain with current state \( S \).

**Proposition 2.4** ([AP08] p.4). Let \( U \) be a uniform random variable on \([0,1]\) and \( S_t \) the set selected in the \( t^{th} \) step of the evolving set process. Then

\[
\mathbb{E}_{S_t} \left[ \mu(S_{t+1}) \mid U \geq 1/2 \right] = \mu(S_t) + \partial(S_t) = \mu(S_t) \left( 1 + \varphi(S_t) \right)
\]

(2.60)

\[
\mathbb{E}_{S_t} \left[ \mu(S_{t+1}) \mid U < 1/2 \right] = \mu(S_t) - \partial(S_t) = \mu(S_t) \left( 1 - \varphi(S_t) \right)
\]

(2.61)

Furthermore, \( \{\mu(S_t)\}_{t \in \mathbb{N}} \) is a martingale.

**Proof.**

\[
\mathbb{P}_{S_t}(y \in S_{t+1} \mid U \leq 1/2) = \mathbb{P}_{S_t}(p(y, S_t) \geq U \mid U \leq 1/2)
\]

(2.62)

\[
= \frac{\mathbb{P}(U \leq p(y, S_t), U \leq 1/2)}{\mathbb{P}(U \leq 1/2)}
\]

(2.63)

\[
= \frac{\mathbb{P}(U \leq \min\{1/2, p(y, S_t)\})}{1/2}
\]

(2.64)

\[
= \begin{cases}
2 \cdot \frac{1}{2} & \text{if } \frac{1}{2} \leq p(y, S_t) \\
2 \cdot p(y, S_t) & \text{if } \frac{1}{2} > p(y, S_t)
\end{cases}
\]

(2.65)
Since $p(y, S_t) < \frac{1}{2}$ if and only if $y \in S'_t$, this yields
\[
P_{S_t}(y \in S_{t+1} \mid U \leq 1/2) = \begin{cases} 
1 & \text{if } y \in S_t \\
2p(y, S_t) & \text{else}
\end{cases}
\] (2.66)

Hence,
\[
E_{S_t}[\mu(S_{t+1}) \mid U \geq 1/2] = E_{S_t} \left[ \sum_{x \in V} d(x)I_{x \in S_{t+1}} \mid U \geq 1/2 \right]
\] (2.67)
\[
= \sum_{x \in V} d(x)P_{S_t}(x \in S_{t+1} \mid U \leq 1/2)
\] (2.68)
\[
= \sum_{x \in S} d(x)P_{S_t}(x \in S_{t+1} \mid U \leq 1/2)
\] (2.69)
\[
+ \sum_{x \in S'} d(x)P_{S_t}(x \in S_{t+1} \mid U \leq 1/2)
\] (2.70)
\[
= \sum_{x \in S} d(x) + \sum_{x \in S'} 2d(x)p(x, S_t)
\] (2.71)
\[
= \mu(S_t) + \sum_{x \in S'} 2d(x) \cdot \frac{E(\{x\}, S_t)}{2d(x)}
\] (2.72)
\[
= \mu(S_t) + \sum_{x \in S'} E(\{x\}, S_t)
\] (2.73)
\[
= \mu(S_t) + E(S_t, S'_t) = \mu(S_t) + \partial(S_t)
\] (2.74)

Something similar can be argued for $U > 1/2$.
\[
P_{S_t}(y \in S_{t+1} \mid U > 1/2) = P(p(y, S_t) > U \mid U > 1/2)
\] (2.75)
\[
= \frac{P(1/2 < U \leq p(y, S_t))}{P(U > 1/2)}
\] (2.76)
\[
= \begin{cases} 
2 \cdot 0 & \text{if } p(y, S_t) \leq 1/2 \\
2 \cdot (p(y, S_t) - \frac{1}{2}) & \text{else}
\end{cases}
\] (2.77)

Again by the observation that $p(y, S_t) > \frac{1}{2}$ if and only if $y \in S$,
\[
P_{S_t}(y \in S_{t+1} \mid U \leq 1/2) = \begin{cases} 
2p(y, S_t) - 1 & \text{if } y \in S_t \\
0 & \text{else}
\end{cases}
\] (2.78)
\[
E_{S_t}[\mu(S_{t+1}) \mid U > 1/2] = \sum_{x \in V} d(x)P_{S_t}(x \in S_{t+1} \mid U > 1/2)
\] (2.79)
\[
= \sum_{x \in S_t} d(x)(2p(x, S_t) - 1)
\] (2.80)
\[
= \sum_{x \in S_t} d(x) \left( 1 + \frac{E(\{x\}, S_t)}{d(x)} - 1 \right) = \sum_{x \in S_t} E(\{x\}, S)
\] (2.81)
\[
= \sum_{x \in S_t} \left( d(x) - E(\{x\}, S'_t) \right)
\] (2.82)
\[
= \sum_{x \in S_t} d(x) - \sum_{x \in S_t} E(\{x\}, S_t)
\] (2.83)
\[
= \mu(S_t) - E(S_t, S'_t) = \mu(S_t) - \partial(S_t)
\] (2.84)

proving the first two equations. Clearly, 
\[
\mu(S_t) + \partial(S_t) = \mu(S_t) \left( 1 + \frac{\partial(S_t)}{\mu(S_t)} \right) = \mu(S_t)(1 + \varphi(S_t))
\] (2.85)
The claim that \( \mu(S_t) \) is a martingale follows easily now, since

\[
\mathbb{E}_{S_t}[\mu(S_{t+1})] = \frac{1}{2} \cdot \mathbb{E}_{S_t}[\mu(S_{t+1}) \mid U \leq 1/2] + \frac{1}{2} \cdot \mathbb{E}_{S_t}[\mu(S_{t+1}) \mid U > 1/2]
\]

\[
= \frac{1}{2} (\mu(S_t) + \partial(S_t) + \mu(S_t) - \partial(S_t)) = \mu(S_t)
\]  

\[ (2.87) \]

2.3.3 Volume-biased evolving set process

Usually, the evolving set process evolves from a set consisting of a single vertex, i.e. \( S_0 = \{ v \} \) for \( v \in V \). One might notice that the Markov chain of the ESP has two absorbing states, namely \( S = \emptyset \) and \( S = V \). For \( S = \emptyset \) this is evident. For \( S = V \) this follows from the fact that

\[
p(y, V) = \frac{1}{2} \left( \mathbb{1}_{y \in V} + \frac{E\{y\}, V}{d(y)} \right) = \frac{1}{2} \left( 1 + \frac{d(y)}{d(y)} \right) = 1 \quad \forall y \in V
\]

So that

\[
S' = \{ y \in V \mid p(y, V) \geq U \} = V
\]  

\[ (2.90) \]

Another observation shows that, starting from a single vertex, the process enters state \( \emptyset \) after just one step with probability \( \frac{1}{2} \), because

\[
p(v, \{ v \}) = \frac{1}{2} \left( \mathbb{1}_{y \in \{ v \}} + \frac{E\{v\}}{d(y)} \right) = \frac{1}{2} (1 + 0) = \frac{1}{2}
\]

\[ (2.91) \]

In this thesis the ESP will be used to find low-conductance clusters in the graph, for which it is not preferable to end up in state \( \emptyset \)\footnote{From this definition, one can easily see that the probability of transitioning to the empty set is equal to zero, whereas the probability of choosing a lower volume set. Still, the set does not grow if \( U \geq 1/2 \) and does not shrink if \( U < 1/2 \). We will later see that using the volume-biased version of the ESP also improves the running time of algorithms to find low-conductance sets based on evolving sets.}

\[ \tag{2.92} \]

2.4 Algorithms

In this section, the three central partitioning techniques to be investigated in this thesis will be introduced. Here, we will only give brief descriptions, as we will do an extensive analysis on each of the techniques separately in the next three chapters.
2.4. Algorithm

2.4.1 Spectral clustering

Partitioning methods based on the analysis of the spectrum of the graph Laplacian are grouped under the name of spectral clustering. Following [vL07], we will present a general framework for all spectral clustering methods, using either the normalised or unnormalised graph Laplacian. All spectral methods make use of the $k$-means algorithm as a subroutine. For completeness of this text, the pseudo code of the $k$-means algorithm is presented in Algorithm 1. Any norm can be chosen in the $k$-means algorithm, although the usual choice is the Euclidean norm. The partition resulting from the $k$-means algorithm can be interpreted as the Voronoi diagram generated by the centers. The general scheme for spectral clustering is given in Algorithm 2.

\begin{algorithm}
Input: Set of points $x_i \in \mathbb{R}^d$, $i = 1, \ldots, m$, number of clusters $k$, initial centers $m_1, \ldots, m_k \in \mathbb{R}^d$
Output: Partition $\{S_1, \ldots, S_k\}$ of the points
repeat
  for $i = 1, \ldots, k$ do
    Set $S_i = \{x_j | \|x_j - m_i\| = \min_l \|x_j - m_l\|\}$;
  end
  for $i = 1, \ldots, k$ do
    Set $m_i = \left( \sum_{x_j \in S_i} x_j \right) / |S_i|$
  end
until convergence is reached;
\end{algorithm}

Algorithm 1: $k$-Means

2.4.2 Spielman-Teng

The algorithm presented in [ST08] is based on the $t$-step distribution of a random walk on the graph. In the subroutine Nibble, a single cluster of conductance less than a certain threshold $\psi_{\text{thres}}$ is constructed using this distribution, together with a truncation rule. Let $e_v$ again denote the all zero vector with one on the $v$th entry. Furthermore, the truncation operation $[\cdot]_\epsilon$ does the following for a vector $p$.

\begin{equation}
[p]_\epsilon(v) = \begin{cases} 
p(v) & \text{if } p(v) \geq d(v)\epsilon \\
0 & \text{else} \end{cases}
\end{equation}

Here, $M = (I + AD^{-1})/2$ the transposed transition matrix of the lazy random walk. A stripped version of Nibble is given in Algorithm 3.

In the paper [ST08] itself the algorithm is presented with numerous non-trivial constants, which are omitted in Algorithm 3. For a complete description of Nibble the reader is referred to Chapter 4. On top of the subroutine, Spielman and Teng introduce the algorithm Partition. A simplified version is described in Algorithm 4. The basic idea behind Partition is to select a starting vector of high degree with high probability, grow a good quality cluster (if possible) from this vertex, remove the clustered vertices and then apply the procedure to remaining vertices until no vertices are left. In Chapter 4, 5 and 6 we will do an extensive analysis on this technique and present some toy examples on the functioning of the algorithm.
CHAPTER 2. PRELIMINARIES

Input: Graph $G = (V, E)$ with $|V| = n$, adjacency matrix $A$, Laplacian $L$, $k$ the required number of clusters

Output: Partition $\{V_1, ..., V_k\}$ of the vertices of $V$

- Compute the eigenvalues of $L$ and place them in non-decreasing order.
- Compute the $k$ eigenvectors $u_1, ..., u_k$ associated with the $k$ smallest eigenvalues.
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing $u_1, ..., u_k$ as columns.
- Let $y_i \in \mathbb{R}^k$ represent the $i$th row of $U$.
- Group the points $y_i$, for $i = 1, ..., n$, using the $k$-means algorithm into clusters $C_1, ..., C_k$
- Set $V_i = \{ j \mid y_j \in C_i \}$ for $i = 1, ..., k$.

Algorithm 2: General framework Spectral Clustering

Input: Graph $G = (V, E)$, adjacency matrix $A$ of $G$, starting vertex $v \in V$, conductance threshold $\varphi_{\text{thres}}$, number of iterations $T$, truncation threshold $\varepsilon$, maximum number of iterations $t_{\text{last}}$.

Output: Cluster $C \subset V$ of conductance less than $\varphi_{\text{thres}}$ or $C = \emptyset$.

Set $M$ as in (2.11); Set $q_0 = e_v$ and $r_0 = e_v$;

for $t = 1, ..., t_{\text{last}}$ do

| Set $q_t = M^T r_{t-1}$; |
| Set $r_t = [q_t]_{\varepsilon}$; |
| Let $S$ be all vertices $u \in V$ such that $r_t > 0$. Order $S$ by the permutation $\Pi$, such that $p(\Pi(i)) \geq p(\Pi(i+1)) \geq d(\Pi(i))$; |
| and let $S_j = \{ \Pi(1), ..., \Pi(j) \}$ for $j = 1, ..., |S|$; |
| for $j = 1, ..., |S|$ do |
| if $\varphi(S_j) \leq \varphi_{\text{thres}}$ then |
| | Set $C = S_j$; |
| | Terminate procedure; |
| end |
| end |

end

return $C = \emptyset$;

Algorithm 3: Spielman and Teng’s Nibble
2.4. ALGORITHMS

Input: Graph $G = (V, E)$, conductance threshold $\phi_{thres}$, truncation threshold $\epsilon$.

Output: Partition $C = \{C_1, ..., C_k\}$ of $V$.

Set $W_0 = V$;
Set $t = 0$;

while $W_t \neq \emptyset$ do
  Choose $u$ as starting vertex $v$ with probability $d(u) / \sum_{x \in W_t} d(x)$;
  Set $t = t + 1$;
  Let $G[W_{t-1}]$ be the graph induced by the vertices $W_{t-1}$;
  Set $C_t = \text{Nibble}(G[W_t], v, \phi_{thres}, \epsilon)$;
  Set $W_t = W_{t-1} \setminus C_t$.
end
return $C = \{C_1, ..., C_t\}$;

Algorithm 4: Spielman and Teng’s Partition

2.4.3 Andersen-Peres

The third and most state of the art procedure to be studied in this thesis is due by Andersen and Peres as proposed in [AP08]. The algorithm GenerateSample presented in their paper uses the theory on evolving sets as introduced in the previous section. Moreover, GenerateSample in fact simulates the volume-biased ESP until a cluster of low conductance is found. Formally, the procedure follows the path given in Algorithm 5. Just as Nibble, Generate-Sample only constructs a single cluster of low conductance, whereas a partition is needed. Therefore Andersen and Peres cite Spielman and Teng in their paper on using their method Partition to indeed generating a partition of the vertices. More detailed analysis on the procedure of GenerateSample can be found in Chapter 6.

Input: Graph $G = (V, E)$, starting vertex $v \in V$, conductance threshold $\phi_{thres}$, maximum number of iterations $T$.

Output: Cluster $S \subseteq V$ of conductance less than $\phi_{thres}$ or $S = \emptyset$.
Set $S_0 = \{v\}$;

for $t = 1, ..., T$ do
  Choose $S_t = S'$ with probability $\hat{K}(S_{t-1}, S')$ as in Equation (2.92);
  if $\phi(S_t) \leq \phi_{thres}$ then
    Set $S = S_t$;
    Terminate procedure;
  end
end
return $S = \emptyset$;

Algorithm 5: Andersen and Peres’ GenerateSample
Chapter 3

Global partitioning algorithms

3.1 Introduction

This chapter presents two global partitioning algorithms that partition a graph \( G = (V, E) \), where \(|V| = n\). Because these algorithms are believed to be suboptimal for our purposes, we will use them as benchmark for drawing a comparison with the new algorithms developed in Chapters 4 to 7.

The first algorithm described in this chapter is the self-tuning spectral clustering algorithm, which relies on the framework given in Subsection 2.5.1. The second algorithm is the \( k \)-medoids algorithm, initialised by the spectral clustering routine or a technique developed by Natasa Jovanovic and Richard Verhoeven. After this, we will apply both algorithms to some stylised graphs in order to develop some intuition for how they work in network settings.

3.2 Self-tuning spectral clustering

3.2.1 Affinity matrix and new Laplacian

As we have seen in Subsection 2.4.1, the original spectral clustering algorithm uses the eigenvalues and eigenvectors of the graph Laplacian \( L \) to derive a partition of the graph. The possible forms of this Laplacian as given in this section all somehow relate to the adjacency matrix. However, by only using this adjacency matrix, we neglect the valuable information on the coordinates of the vertices and thereby the distances between them. As we prefer nodes close to each other to be assigned to the same segment, rather than nodes that lay far apart but just within connectivity distance, it seems advantageous to include this information in a partitioning algorithm. This could for instance be done by assigning a value to each pair of vertices \((i, j)\), indicating their similarity \( w_{ij} \). These values can be seen as weights on the edges of the graph. In this case, close vertices would get a large similarity value compared to distant vertices, whereas vertices not able to communicate directly get a similarity value equal to zero. Storing these similarity values in an \( n \times n \)-matrix \( W \), we construct a so-called affinity matrix. Ng et al. were the first to define such a matrix in terms of distances in [NJW01].

**Definition 3.1** (NJW affinity matrix). Let \( \sigma > 0 \) be the scaling parameter. The NJW affinity matrix

\[ W_{ij} = \exp\left(-\frac{(d(i,j))^2}{\sigma^2}\right) \]

is the affinity matrix, where \( d(i,j) \) is the Euclidean distance between vertices \( i \) and \( j \).

---

1Natasa Jovanovic and Richard Verhoeven are affiliates of the Security and Embedded Networked Systems research group of the department of Mathematics and Computer Science of Eindhoven University of Technology.
CHAPTER 3.GLOBAL PARTITIONING ALGORITHMS

$W_{NJW} \in \mathbb{R}^{n \times n}$ is given by $(w_{ij})_{i,j=1}^{n}$ and

$$w_{ij} = \begin{cases} \exp\left(\frac{-\|v_i - v_j\|^2}{\sigma^2}\right), & \text{if } i \neq j, \\ 0, & \text{else,} \end{cases}$$

where $\| \cdot \|$ is a distance function defined on the vertices.

The distance function in the definition can be chosen arbitrarily as long as it is symmetric. We will use the Euclidean distance measure in the remainder of this chapter. The scaling parameter $\sigma$ plays a big role in the definition of the affinity matrix. It can be seen as a measure of to what extent two vertices are considered similar, or in our case, belong to the same well-connected segment. The higher the value of $\sigma$, the more surrounding vertices will be considered similar. From a starting point, it is certainly not clear how to choose the value of $\sigma$. The scaling is of course highly dependent on the topology of the graph. If the average distance to the closest neighbouring node is, say 50 meters, then you might not want to set $\sigma$ to 1, or something of that order. We will come back to this in the next subsection.

Now that we have been able to include spatial information into a matrix, our hopes are that the spectral clustering routine stated in Subsection 2.4.1 allows modification to handle also the real-valued matrices, instead of just the binary adjacency matrices. Let us define the diagonal matrix $D_{W} \in \mathbb{R}^{n \times n}$ with

$$(D_{W})_{ii} = \sum_{j=1}^{n} w_{ij},$$

with $w_{ij}$ as defined in (3.1). Then the new unnormalised Laplacian is given by

$$L_{W} = D_{W} - W,$$

and the normalised version by

$$L_{W}^{nor} = D_{W}^{-1/2}L_{W}D_{W}^{-1/2} = D_{W}^{-1/2}(D_{W} - W)D_{W}^{1/2} = I - D_{W}^{-1/2}WD_{W}^{1/2}$$

[NJW01] uses $L_{W}^{*} = D_{W}^{-1/2}WD_{W}^{1/2}$ instead of $L_{W}^{nor}$ as given above. Note however that the eigenvectors of both matrices are the same since $L_{W}^{*} = I - L_{W}^{nor}$, while the eigenvalues of the second Laplacian are equal to one minus the eigenvalues of the first matrix. For consistency with the theory developed in Chapter 2, we will work with the normalised Laplacian in (3.4).

In Proposition 2.2, we derived some properties of the unnormalised Laplacian based on the adjacency matrix of a graph, which induced similar properties for the normalised symmetric Laplacian. By looking at the proof of the proposition closely, we see that it does not depend on the exact value of $a_{ij} \in \{0, 1\}$. Hence, we are free to substitute $a_{ij}$ by $w_{ij} \in \mathbb{R}_{+}$ without changing the proof. We thus find that the unnormalised Laplacian based on an affinity matrix is also positive semi-definite and has smallest eigenvalue equal to zero, with corresponding eigenvector $1$. The same result applies to the normalised Laplacian based on the affinity matrix, except that the eigenvector with eigenvalue zero is equal to $D_{W}^{1/2}1$. Although technically the graph is considered complete if we use weights as in (3.1), very distant pairs of nodes are given similarity very close to zero. This way, the reasoning in Subsection 2.2.2 of why the eigenvectors of the Laplacian provide insight in the clustering structure of the graph still applies. Hence, the full method of spectral clustering using the affinity matrix of a graph is promising to give a good partition of the graph that takes into account the available spatial information.

**Remark 3.1.** The properties of the weighted Laplacian are not restricted to the choice of similarity function Ng et al. made in (3.1). In fact, one can take any function in terms of distance can be chosen, as long as it is decreasing.
3.2.2 Local scaling

In this subsection, we dive deeper into the choice of the scaling parameter $\sigma$, as described in Definition 3.1. Naturally, the value of $\sigma$ should depend on the structure of the graph, focusing on the distances between neighbouring vertices. A first guess would relate the value of $\sigma$ to the average distance to the vertices within communication distance. However, if the density of vertices differs throughout the graph, which is the case in our practical setting, where we are dealing with inhomogeneous networks, the weights on the edges will not resemble the similarities correctly. For this reason, fixing a global scaling parameter might fail to provide a good basis for the clustering procedure for these kinds of networks. It might be a better idea to set the scaling locally. Zelnik-Manor and Perona explored this idea in [ZmP04]. Instead of fixing one $\sigma$ for the network, they set a scaling parameter $\sigma_i$ for every $v_i \in V$. The value of $\sigma_i$ depends on the local statistics of the neighbourhood of $v_i$. The locally scaled affinity matrix defined in this paper is given in the next definition.

**Definition 3.2 (Locally scaled affinity matrix).** Let

$$\sigma_i = \|v_i - v_i(K)\|, \quad \text{(3.5)}$$

where $v_i(K)$ is the $K$th closest neighbour of $v_i$. The locally scaled affinity matrix is defined by $W_{loc}^{ij} = (w_{ij}^{loc})_{ij=1}^n$, where

$$w_{ij}^{loc} = \begin{cases} 
\exp\left(-\frac{\|v_i - v_j\|^2}{\sigma_i \sigma_j}\right), & \text{if } i \neq j, \\
0, & \text{else.} \end{cases} \quad \text{(3.6)}$$

[ZmP04] suggests to use $K = 7$, and we will stick to this value in our implementation. A subject for further research could concern the value of $K$. There are a few remarks to be made. First we note that the similarity of a pair of vertices depends on both local scaling parameters, yielding a symmetric measure. Furthermore, we clearly see that if we are dealing with a network in which nodes are uniformly spread, we get $\sigma_i \approx \sigma_j$, approximately yielding the original NJW affinity matrix with one fixed $\sigma$. In less populated regions of the graphs the distances to neighbours will be large and so will $\sigma_i$. This implies that vertices situated relatively far apart are still considered ‘good’ neighbours, whereas this would not be the case if both vertices would be situated in a dense area. This is beneficial for our problem, because it reflects the property of wireless mesh networks that in less populated regions, links over large distances may exist, while in denser regions these long-distance links are less likely, because communication distance is limited by the large amount of noise causing interference. The locally scaled affinity matrix seems to be a better choice in stored spatial information on the network than the affinity matrix of Definition 3.1. Consequently, this matrix will be used for the spectral clustering algorithm in the next subsection.

3.2.3 Algorithm and implementation

Because in Subsection 3.2.1 we found that the Laplacian based on the affinity matrix inherits the properties of the Laplacian solely based on the adjacency matrix, the locally scaled affinity matrix is suitable for the spectral clustering routine. However, taking a closer look at the structure of the matrix in Definition 3.2 we see that $w_{ij} > 0$ for all $i \neq j$. Storing all of these values for a network of say 30,000 nodes, is infeasible, let alone computing its eigenvectors, of which the complexity is $O(n^3)$. Hence, for our implementation a sparse affinity matrix $\hat{W}$ will
CHAPTER 3. GLOBAL PARTITIONING ALGORITHMS

Input: Graph $G = (V, E)$ with $|V| = n$, $k$ the required number of clusters

Output: Partition $\{V_1, ..., V_k\}$ of the vertices of $V$

- For all $v_i \in V$, compute the local scaling parameter $\sigma_i$ as in (3.5).
- Construct the sparse locally scaled affinity matrix $\hat{W}$ as in (3.7).
- Compute the diagonal matrix $\hat{D}$ with $(\hat{D})_{ii} = \sum_{j=1}^{n} \hat{w}_{ij}$.
- Set $\hat{L} = I - \hat{D}^{-1/2}\hat{W}\hat{D}^{-1/2}$.
- Compute the eigenvalues of $\hat{L}$ and place them in non-decreasing order.
- Compute the $k$ eigenvectors $u_1, ..., u_k$ associated with the $k$ smallest eigenvalues.
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing $u_1, ..., u_k$ as columns.
- Let $y_i \in \mathbb{R}^k$ represent the $i$th row of $U$.
- Group the points $y_i$, for $i = 1, .., n$, using the $k$-means algorithm into clusters $C_1, ..., C_k$.
- Set $V_i = \{ v_j \mid y_j \in C_i \}$ for $i = 1, ..., k$.

Algorithm 6: Self-tuning spectral clustering

be used, only storing the similarity of a pair of vertices if the two are within communication distance $R_c$. The entries of $\hat{W}$ are given by

$$\hat{w}_{ij} = \begin{cases} \exp\left(\frac{-||v_i - v_j||^2}{\sigma_i \sigma_j}\right), & \text{if } ||v_i - v_j|| \leq R_c \text{ and } i \neq j, \\ 0, & \text{else.} \end{cases} \quad (3.7)$$

Replacing $A$ with $\hat{W}$ in the general spectral clustering framework of Algorithm 2, we come to the following partitioning algorithm, also given in [ZmP04]. Except for the so-called self-tuning feature just described, [ZmP04] includes method to detect the right number of clusters in the algorithm presented. Since we know the number of required clusters beforehand, we left it out in Algorithm 6.

3.2.4 Example

To get some more insight into the spectral clustering algorithm and its self-tuning variant, we apply the algorithms to a stylised example. Consider the layout of nodes depicted in Figure 3.1 and fix the connectivity distance to 3. The edges indicate which pairs of vertices are within the connectivity range. We see that there are a lot of connections between the nodes, and this is typical for our setting of street lights, which are usually about a few tens of meters apart, with a connectivity range of 300 meters. By the whole picture, we see two clusters: one dense group of vertices on the left (nodes 1 to 16) and a less dense group surrounding it on the right (nodes 17 to 33). We analyse how both the spectral clustering algorithm with the NJW affinity matrix and the self-tuning variant partition this graph into two segments and whether either one of them indeed finds the desired division.

First, we run the NJW spectral clustering routine. For this, we need the NJW affinity matrix of the graph depicted in Figure 3.1 as in (3.1). For illustrating the differences between
3.2. SELF-TUNING SPECTRAL CLUSTERING

![Example graph with edges indicating pairs of nodes able to communicate.](image)

Figure 3.1: Example graph with edges indicating pairs of nodes able to communicate.

![Plot of the coordinates given by the first two eigenvectors of the NJW Laplacian.](image)

Figure 3.2: Plot of the coordinates given by the first two eigenvectors of the NJW Laplacian.

We can see the columns as coordinates $\mathbb{R}^2$ and plot them on the 2D plane, yielding the plot in Figure 3.2. Afterwards the $k$-means algorithm clusters these points according to the geographical position, giving the partition of these points in $\mathbb{R}^2$ as depicted by different colours in the figure. Because every single point in this plot corresponds to one vertex in the graph, this partition of points can be translated back to a partition of the vertices of the graph. Doing this yields the partition shown in Figure 3.3.

This method did not produce the desired clustering. Hence we resort to the self-tuning variant of the spectral clustering method. As described before, this method uses a local scaling parameter $\sigma_i$ for every vertex, indicating the density of vertices in the area surrounding it. For
CHAPTER 3. GLOBAL PARTITIONING ALGORITHMS

Figure 3.3: Partition of example graph resulting from NJW spectral clustering.

Table 3.1: Local scaling values.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\sigma_i$</th>
<th>$i$</th>
<th>$\sigma_i$</th>
<th>$i$</th>
<th>$\sigma_i$</th>
<th>$i$</th>
<th>$\sigma_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6973</td>
<td>10</td>
<td>0.7616</td>
<td>18</td>
<td>1.1916</td>
<td>26</td>
<td>1.1481</td>
</tr>
<tr>
<td>2</td>
<td>0.5117</td>
<td>11</td>
<td>0.6879</td>
<td>19</td>
<td>1.1916</td>
<td>27</td>
<td>1.1978</td>
</tr>
<tr>
<td>3</td>
<td>0.5372</td>
<td>12</td>
<td>0.7209</td>
<td>20</td>
<td>1.2289</td>
<td>28</td>
<td>1.3401</td>
</tr>
<tr>
<td>4</td>
<td>0.6739</td>
<td>13</td>
<td>0.5620</td>
<td>21</td>
<td>1.1978</td>
<td>29</td>
<td>1.7459</td>
</tr>
<tr>
<td>5</td>
<td>0.5620</td>
<td>14</td>
<td>0.7634</td>
<td>22</td>
<td>1.2904</td>
<td>30</td>
<td>1.2444</td>
</tr>
<tr>
<td>6</td>
<td>0.6543</td>
<td>15</td>
<td>0.7083</td>
<td>23</td>
<td>1.1236</td>
<td>31</td>
<td>1.2916</td>
</tr>
<tr>
<td>7</td>
<td>0.5265</td>
<td>16</td>
<td>0.6205</td>
<td>24</td>
<td>1.4127</td>
<td>32</td>
<td>1.3576</td>
</tr>
<tr>
<td>8</td>
<td>0.5427</td>
<td>17</td>
<td>1.7411</td>
<td>25</td>
<td>1.2764</td>
<td>33</td>
<td>1.4127</td>
</tr>
<tr>
<td>9</td>
<td>0.7616</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.2. SELF-TUNING SPECTRAL CLUSTERING

Figure 3.4: Example graph, where the marked edges show pairs of vertices with similarity bigger than 0.3.

Figure 3.5: Plot of the coordinates given by the first two eigenvectors of the locally scaled Laplacian.

The graph used in our example, these values of $\sigma_i$ are given in Table 3.1. From this table, we already see a clear division between vertices 1 to 16, for which $\sigma_i < 0.77$ and vertices 17 to 33, for which $\sigma_i > 1.12$. Furthermore, if we construct the locally scaled affinity matrix $\hat{W}$ as in (3.7) and search for edges indicating high similarity, for instance $\hat{w}_{ij} \geq 0.3$, we find the marked edges in Figure 3.4. This figure clearly shows that similarity now becomes dependent on the density of the local area of the vertices. For instance, although $||v_2 - v_5|| < ||v_{24} - v_{33}||$, vertices 24 and 33 are considered similar, while vertices 2 and 5 are not. The self-tuned affinity matrix hence highlights the edges that are really relevant for the clustering. If we compute the corresponding Laplacian together with the first two eigenvectors and plot the corresponding 2D coordinates on the plane, this yields Figure 3.5. Compared to the scatter plot in Figure 3.2, Figure 3.5 already shows a much better separation of the points, implying a more evident partition as depicted by the different colours. Translating the clustered points back to the original graph gives the clustering as in Figure 3.6, which is indeed the desired partition of the vertices. As the self-tuning spectral method takes much more account to inhomogeneously populated graphs compared to the NJW method, we will use this method for the comparison of partitioning algorithms in Chapter 9.
3.3 \( k \)-Medoids

3.3.1 Algorithm

The \( k \)-medoids algorithm is probably the most intuitive method for finding a partitioning of an unweighted graph \( G = (V, E) \) with known coordinates. It is related to the \( k \)-means method described in Algorithm 1. Whereas the \( k \)-means method minimises the average Euclidean distance of each vertex to the closest mean, the \( k \)-medoid minimises the average distance in hops to the closest so-called medoid. Before we go deeper into the algorithm, we give a concise description of the medoid of a graph.

Definition 3.3 (Medoid). Let \( G = (V, E) \) be an unweighted graph with \( V = \{v_1, ..., v_n\} \). The medoid of \( G \) is the vertex \( v_{\text{med}} \in V \) that minimises the average number of hops to every other vertex in the graph. That is,

\[
v_{\text{med}} = \arg \max_{v_i \in V} \frac{1}{n} \sum_{j=1}^{n} d(v_i, v_j),
\]

where \( d(v_i, v_j) \) denotes length of the shortest path (in terms of hops) between \( v_i \) and \( v_j \).

The medoid of a graph is typically situated at the center of the graph. Therefore, when considering the induced subgraph on the vertices of one cluster, the medoid would be a good first choice for the segment controller position, although we are not able to tell whether is it optimal, because we do not yet have a clear view on the relation between the hop count and the network performance measures. We will stick to this idea in the next chapters, and we will verify in Chapter 9 that this is indeed a good position, given the partitioning of the graph as input. The remainder of the algorithm is similar to the \( k \)-means algorithm. The full description is given in Algorithm 7.

By convergence we mean that the set of medoids \( \{m_1, ..., m_k\} \) does not change any more. Note that in each iteration step the full distance matrix of each subset of the partition needs to be computed, in order to find the medoids. There are multiple ways to achieve this. We
3.3. K-MEDOIDS

**Input**: Graph $G = (V, E)$ with $|V| = n$, $k$ the required number of clusters, initial medoids $M = \{m_1, ..., m_k\} \subset V$.

**Output**: Partition $\{V_1, ..., V_k\}$ of the vertices of $V$.

repeat
  1. For all $i \in \{1, ..., k\}$ set $V_i = \{v_j \mid m_i = \arg \min_{m \in M} d(m, v_j)\}$.
  
  2. For all $i \in \{1, ..., k\}$ update $m_i$ by computing the medoid of the graph induced by $V_i$.

until convergence.

**Algorithm 7**: k-medoids algorithm

could for instance apply Dijkstra’s algorithm to all vertices in each subset of the partition, see . However, the complexity of Dijkstra’s algorithm applied to a vertex in $V_i$ is $O(|V_i|^2)$, and since the procedure has to be done for all $|V_i|$ vertices of each subset, the total complexity is $O(|V_i|^3)$. Because the $k$-medoids algorithm, depending on the starting configuration, typically needs a few dozens of iterations to terminate, depending on the initialisation, this method will be very time consuming. It is possible to reduce the running time of the full algorithm by observing that the new medoids found in the update step will most probably be close to the old positions. Therefore, checking only a portion of the vertices which are close to the old medoid might be sufficient for finding the next medoid of each cluster.

### 3.3.2 Initialisation

The $k$-medoids algorithm attempts to minimise the error function

$$E = \sum_{i=1}^{k} \sum_{v \in V_i} d(v, m_i)$$

by iterative improvement. In doing so, however, the algorithm might get stuck in a local minimum. The attentive reader will have noticed that the $k$-medoids algorithm asks for an initial set of medoids as input. A good choice of input vertices is essential for indeed finding the global minimum with corresponding medoids. Figure 3.7 illustrates this issue. Suppose $k = 2$ and we choose the two red vertices as the two initial vertices in Figure 3.7a. The partitioning resulting from the first step is indicated with the dashed lines. Figure 3.7b shows the updated positions of the medoids. In this case $E = 16$. From this point onwards, the medoid position and corresponding partition will not change and the procedure terminates. However, the medoids minimising $E$ is given in Figure 3.7c, yielding $E = 12$. A reasonable guess of the medoid positions is thus necessary for not ending up in a local optimum. Secondly, the number of iterations required for convergence is also reduced tremendously. However, if we knew the optimal positions of the medoids beforehand, there would be no need for the algorithm. Evidently, choosing the initial medoids is by no means trivial.

A first requirement of the initial positions of the initial medoids is that they are evenly spread out throughout the graph. Secondly, vertices with high degree are more likely to be ‘good’ medoids, for they have many vertices at distance 1. Besides that, these high degree vertices are likely to have short connections to all other nodes. In choosing an initial configuration of the medoids, we have a preference towards vertices that fit these requirements.
CHAPTER 3. GLOBAL PARTITIONING ALGORITHMS

Figure 3.7: Example graph in which $k$-medoids fails to obtain the global minimum.

The spectral clustering methods discussed in the previous section already yielded a clustering of the vertices using a similarity measure. Because the computation time of that algorithm is relatively short, and it produces a well spread out partition of the graph, we could use this clustering as a starting point for the $k$-medoids algorithm. We then choose the initial medoids for Algorithm 7 equal to the medoids of the subsets of the partition by the spectral clustering routine in Algorithm 6. This method of choosing the initial subset of medoids satisfies the first requirement we set. However, we cannot know whether it indeed has a high degree. The second initialisation algorithm, proposed by Natasa Jovanovic and Richard Verhoeven, does take this into account.

We illustrate the method for initialisation of the medoids by an example. Suppose we want to construct an initial medoids set of size $k = 14$ for deployment B depicted in Figure 1.4. The first step is to do draw the bounding box around all vertices. In case of deployment B, we have $x_{\text{min}} = -3894$, $x_{\text{max}} = 2770$, $y_{\text{min}} = -2198$, $y_{\text{max}} = 3117$. Therefore, the bounding box is $[-3894, 2770] \times [-2198, 3117]$. In the next step, we divide the bounding box into $l \cdot m > k$ sub-rectangles, as is done in Figure 3.8 for $l = 6$, $m = 5$. After that, we search for the vertex with highest degree in each rectangle, say $t_1, \ldots, t_{lm}$, which are depicted in Figure 3.8 as the red dots. In the final step we choose the $k$ vertices of $t_1, \ldots, t_{lm}$ of highest degree and set these as the initial vertices for the $k$-medoids algorithm. In the figure, these selected vertices are depicted with a blue border.

The fact that the graph is first divided into smaller areas in which only one candidate medoid may be selected accounts for the requirement that the initial medoids should be evenly dispersed. However, some fine tuning of the size of the rectangles is in order. Choosing the rectangles too small results in an uneven spread of the medoids, while choosing them too large will cause dense areas to contain only one medoid, where multiple are favored. Indeed, We see in Figure 3.8 that in the upper left area no medoid is selected and thus we might want to consider trying different values of $l$ and $m$. Also, this procedure has a preference for vertices with high degree, satisfying the second requirement.
Figure 3.8: Deployment B with bounding box, divided into 30 sub-rectangles.
3.3.3 Example

In this subsection, we analyse the performance of the $k$-medoids algorithm for the graph in Figure 3.3. Let us set vertex 1 and 33 as the initial medoids. Given these medoids, we need to assign each vertex to the closest medoid. Because we might find vertices with equal hop distance to both medoids and we want the method to be deterministic, we need to set a preference. For this example we say that if the distance of a vertex to both medoids is equal, the vertex in assigned to the second set. For the case with medoids 1 and 33, this yields the partition of the vertices displayed in Figure 3.9. The average number of hop to the medoid is now 12/11 and 13/10 for set one and two, respectively. Next, for each of the sets, the vertex with minimum average distance to all other vertices in the set is computed. Some quick computation shows that this is vertex 14 for the first set and vertex 22 for the second. Again, we assign each of the remaining vertices to the medoid at minimum distances, with preference for the second set. This yields the partition of Figure 3.10. Here, the average hop distance is 19/17 and 9/8 respectively, giving an overall average hop distance of 37/33. Performing one more iteration of this process outputs the same medoids and corresponding partition. Hence we have reached convergence and the algorithm is terminated.

3.4 Remarks

In this chapter we discussed two widely studied partitioning methods. From Figures 3.3, 3.6 and 3.10 we see that the NJW spectral clustering, self-tuning spectral clustering and $k$-medoids algorithm produces different partitions of a relatively simple graph. The self-tuning spectral clustering algorithm constructed the most intuitive clustering, grouping the vertices according to density, while the other two algorithms clustered the vertices mainly by geographical information. Whichever one is desired has to be verified by simulation and comparison, as is done in Chapter 8 and 9.
We have seen that all algorithms investigated in this chapter use properties of the entire graph in the process of constructing a partition. In case of spectral clustering, this is done in computing the (locally scaled) affinity matrix and the corresponding Laplacian. For the \( k \)-medoids algorithm, this is done by assigning all vertices in each iteration step and thereby calculating the distance matrix of the whole graph. In the literature, such algorithms are called \textit{global}. As we have mentioned before, these global algorithms tend to be very slow. For spectral clustering, the most time consuming part is to compute the first \( k \) eigenvalues of the Laplacian, which typically has size in the order of \( 10^4 \) for our application, not to speak about computing the hop distance matrix for such a graph.

An important fact is that the global algorithms are geared solely at a function of distance. These algorithms do not allow for incorporating more conditions or performance criteria, while this might be necessary for the partitioning of actual light networks. Here are several of the conditions we would like to include in the construction of a partition:

- Maximum hop count,
- Size,
- Load distribution.

Because of the lack of control, we will resort to a newer kind of algorithms that do allow the introduction of constraints and modification. These will be discussed in the next four chapters. We will use the spectral clustering and \( k \)-medoids algorithms as a benchmark.
Chapter 4

Local partitioning methods

4.1 Local versus global

We return to our original setting of partitioning a graph $G = (V, E)$ with $|V| = n$ into $k$ segments that contain many internal edges compared to the number of external edges. In Chapter 3, we discussed two global algorithms, that use information on the entire graph to determine a partition. In this chapter, we introduce local algorithms. An algorithm is said to be local if, given a particular vertex as input, in each step only vertices connected to those seen before, are examined. The concept of searching a graph locally is a recent addition to the field of graph algorithms. Spielman and Teng were the first in their paper [ST08] to introduce and exploit this idea. Their main focus was to develop a method to discover clusters in very large graphs. In local clustering techniques, each cluster is generated independently from any other cluster. As the name suggests, this kind of methods neglects the rest of the information contained in the entire graph. This of course has its advantages and disadvantages.

Since in each step only a relatively small number of vertices are explored, local algorithms are much faster than global algorithms, especially on massive graphs. Because we are free to choose an initial vertex from which to search the graph, we are able to fix the region in which a cluster should be found. Another advantage is the possibility to incorporate additional information. That is, because clusters are grown step by step, it is possible to steer the algorithm by introducing constraints. For instance, we can impose that it is not allowed to let the cluster grow beyond a certain maximal size. This is a major advantage compared to the spectral algorithm, which is like a black box that cannot be controlled.

Although local clustering methods might seem like a great improvement compared with spectral clustering, there are some issues we need to address. As mentioned before, local algorithms only consider the local properties of the graph, whereas information on all other regions of the graph is neglected. This naturally leads to the question on to what extend we should keep on growing a cluster. We illustrate this by an example, depicted in Figure 4.1. Assume that at a certain stage, the algorithm produced the cluster in the shaded area, and we want to partition the graph into two segments. It is clear from the overall picture of the graph that this cluster is contained in the larger (and better) cluster. However, from a local point of view, it seems like we found a good, low-conductance cluster. This issue is inherent for all local algorithms. Nevertheless, we will try to address this by introducing an alternative stopping criterion later in this thesis.
CHAPTER 4. LOCAL PARTITIONING METHODS

Figure 4.1: Example graph with typical scenario encountered in local cluster searching.

4.2 Structure

The algorithms discussed in this thesis generate clusters one by one, and it is not evident how this leads to a partitioning of the entire network. The complete procedure for local clustering algorithms can be divided into four steps, all consisting of one or two subroutines. We give a brief description of each of the steps.

1. Selecting seed nodes
   Select the vertices that will be used as initial vertices for the local clustering algorithms. Determine the number of initial vertices $N$.

2. Generating clusters
   Construct a cluster from each of the seed nodes using a local clustering algorithm.

3. Selecting clusters for cover
   Select $k$ out of the $N$ constructed clusters that leads to the best quality partition.

4. Transforming cover into partition
   The union of the selected $k$ clusters selected will with high probability exhibit overlap and some vertices might not be covered at all. How do we account for this and how do we derive a partitioning from this?

Our main focus is on step 2. The clustering algorithms by Spielman-Teng and Andersen-Peres as introduced in Section 2.4 will be the subject of the next three chapters. First, we derive some exact analytic results on the performance of both methods on stylised graphs. After that, we dive deeper into underlying theory and make some modifications to the algorithms, in order to make them suited for our application. Step 1,3 and 4 will be discussed in Chapter 7.

4.3 Generating clusters

In Chapter 2, we briefly discussed the cluster generating algorithms proposed by both Spielman-Teng in [ST08] and Andersen-Peres in [AP08]. Here, we will state the original algorithms, so
that we can attain a deeper understanding of them. The two algorithms are both designed to search for sets of low conductance. The criterion for termination of the algorithms is therefore based on a conductance threshold $\phi^*$.

### 4.3.1 Spielman-Teng

In this section, we cite [ST08]. On input of conductance threshold $\phi^*$, define:

\[
\begin{align*}
    l &= \lceil \log_2(\mu(V)/2) \rceil, \\
    t_1 &= \left\lceil \frac{2}{\phi^*} \ln \left( c_1(l + 2)\sqrt{\mu(V)/2} \right) \right\rceil, \\
    t_h &= ht_{t_1}, \quad 0 \leq h \leq l + 1, \\
    t_{\text{last}} &= (l + 1)t_1, \\
    f_1(\phi) &= \frac{1}{c_2(l + 2)}t_{\text{last}}.
\end{align*}
\]

Spielman and Teng pose some constraints on the constants $c_1, c_2$ and $c_3$, in order to consolidate the proof on the efficiency of the algorithm. The values they choose are $c_1 = 200$, $c_2 = 280$ and $c_3 = 1800$. However, they give no clear reason for these particular values.

Introduce the function

\[
I(p, x) = \max_{w \in [0,1]^n, \sum w(u)d(u) = x} \sum_{u \in V} w(u)p(u).
\]

Note that for $x = \lambda_j(p)$, we find $I(p, x) = p(S_j(p))$, the total probability accumulated in the support set $S_j$.

**Remark 4.1.** The specific choices for constants (4.1)-(4.5) are made in [ST08] in order to complete a technical proof of convergence and complexity. We will not make explicit use of these constants.

Using these ingredients, the complete description of the method is given in Algorithm ??

### 4.3.2 Andersen-Peres

The Andersen-Peres algorithm simulates the volume-biased evolving set process. By Diaconis-Fill, there is a coupling between the growth of the set and a lazy random walk on the same graph. We will discuss this subject in more detail in Chapter 6. Using this coupling, the implementation of the algorithm, that is the simulation of the ESP, becomes more efficient. Furthermore, instead of using a conductance threshold, an upper bound $B \in \mathbb{N}_+$ on the amount of work the algorithm has done, is introduced. This reduces the computation time, by terminating the procedure whenever the so-called ‘cost’ of a sample path exceeds $B$. This cost function is defined as follows.
CHAPTER 4. LOCAL PARTITIONING METHODS

\begin{algorithm}
\textbf{Input:} Graph $G = (V,E)$, $v \in V$, $\varphi^*$, $b \in \mathbb{N}_+$.
\textbf{Output:} $C \subseteq V$, such that either $\varphi(C) \leq \varphi$ or $C = \emptyset$.

Set
\begin{equation}
\epsilon = \frac{1}{c_3(l + 2)t_{last}2^b},\tag{4.9}
\end{equation}

Set $q_0 = \chi_v$, $r_0 = [q_0]_\epsilon$.

\textbf{for} $t = 1, \ldots, t_{last}$ \textbf{do}

1. Set $q_t = Mr_{t-1}$;

2. Set $r_t = [q_t]_\epsilon$;

3. If there exists a $j$ such that
   \begin{enumerate}
   \item $\varphi(S_j(q_t)) \leq \varphi^*$,
   \item $\lambda_j(q_t) \leq (5/6)\mu(V)$,
   \item $2^b \leq \lambda_j(q_t)$,
   \item $I_x(q_t, 2^b) \geq 1/c_4(l + 2)2^b$,
   \end{enumerate}
   then terminate and return $C = S_j(q_t)$.

4. Return $C = \emptyset$.

\textbf{end}

\textbf{Algorithm 8:} Spielman and Teng’s algorithm called Nibble
4.3. Generating Clusters

Definition 4.1. The cost of a sample path \((S_0, ..., S_t)\) of the (volume-biased) ESP is equal to

\[
\text{cost}(S_0, ..., S_t) := \mu(S_0) + \sum_{j=1}^{t} (\mu(S_j \Delta S_{j-1}) + \partial(S_{j-1})),
\]

(4.10)

where \(\Delta\) denotes the symmetric difference.

Also a maximum number of iteration steps \(T \in \mathbb{N}_+\) is set.

Definition 4.2. Let \(T, B \in \mathbb{N}_+\) be given, let

\[
\theta_T = \sqrt{4T^{-1} \log \mu(V)}.
\]

(4.11)

Then define \(\tau(T, B)\) to be the first time \(t\) such that either \(\phi(S_t) < \theta_T, t = T\) or \(\text{cost}(S_0, ..., S_t) > B\).

We will see later that \(\tau\) is a stopping time. The two-sided boundary of a set \(S, \delta(S),\) is defined by

\[
\delta(S) = \{y \in S \mid E(y, S^c) > 0\} \cup \{y \in S^c \mid E(y, S) > 0\}.
\]

(4.12)

The full implementation is given in Algorithm 9 and can also be found in [AP08].

---

**Input:** Graph \(G = (V, E)\), initial vertex \(x \in V\), \(T \in \mathbb{N}_+\), \(B \in \mathbb{N}_+\).

**Output:** \(S \subseteq V\), sampled from the volume-biased ESP with stopping rule \(\tau(T, B)\).

Set \(S = S_0 = \{x\}, X = x_0 = x;\)

**for** \(t = 1, ..., \tau\) **do**

1. Given \(X_{t-1} = x_{t-1}\), select \(X_t = x_t\) with probability \(p(x_{t-1}, x_t)\) and set \(X = x_t\).

2. Pick \(Z\) uniformly at random from \([0, p(x_t, S_{t-1})]\).

3. Define \(S_t = \{y \mid p(y, S_{t-1}) \geq Z\}\). The list \(D\) of vertices in \(S_t \Delta S_{t-1}\) is computed by checking for all \(y \in \delta(S_{t-1})\) whether \(p(y, S_{t-1}) \geq Z\). If so, add \(y\) to \(D\).

4. Update \(\mu(S_t)\) and \(\text{cost}(S_0, ..., S_t)\):

\[
\mu(S_t) = \mu(S_{t-1}) + \mu(D)
\]

5. If \(t = T\) or \(\text{cost}(S_0, ..., S_t) > B\), then set \(t = \tau, S = S_t\) and terminate.

6. Set \(S = S_t\) and update \(\partial(S_t)\) and \(\phi(S_t)\):

\[
\partial(S_t) = \partial(S_{t-1}) + \mu(D) - 2E(D, S_{t-1}), \quad \phi(S_t) = \partial(S_t) / \mu(S_t)
\]

7. If \(\phi(S_t) < \theta_T\), then set \(t = \tau\) and terminate.

**end**

**Algorithm 9:** Andersen and Peres’ algorithm called GenerateSample
Chapter 5

Stylised graphs

In this chapter, we apply the local clustering algorithms, Spielman-Teng and Andersen-Peres to two tractable network structures. The first is a graph with a very clear bottleneck, the second a two-dimensional grid. Although practical networks are usually more complex than the these two stylised graphs, we hope to get some insight in the way in which, and the speed at which, the algorithms spread on a network. Both examples allow us to investigate these questions in more detail analytically than in the case of general network descriptions.

5.1 Dumbbell graph

Consider the graph consisting of two cliques of size $n$ and $m$, respectively, connected by one edge. This graph, which we will refer to as a dumbbell graph, is depicted in Figure 5.1 for $n = 6$ and $m = 5$. It is easy to see that the graph can be divided into two well-connected clusters by removing the edge in the middle, connecting the two cliques. It is also obvious that this link would be the bottleneck if messages were to be transmitted between the two cliques. By tackling the clustering of this graph by the Spielman-Teng and Andersen-Peres algorithm, we hope to show that these local clustering methods indeed find the two obvious clusters and hence prove effective.

We use the following notation. Without loss of generality, assume $n \geq m$ and label the vertices in the left and right clique $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$, respectively. The edge in the middle is said to connect $a_n$ and $b_1$. We will take $a_1$ as the starting vertex in both algorithms. In Table 5.1 we list the volume, cut and conductance values for some subsets of the vertices. We will need them in the remainder of this section. From Figure 5.1 and Table 5.1 we easily derive

Figure 5.1: Dumbbell graph for $n = 6$ and $m = 5$. 

53
that the minimum conductance \( \varphi_G \) of this family of graphs is

\[
\varphi_G = \min_{S \subseteq V} \varphi(S) = \varphi(\{b_1, \ldots, b_m\}) = O\left( \frac{1}{\min\{m, n\}^2} \right).
\]  

**5.1.1 Spielman-Teng**

We have seen in Chapter 2 that the algorithm Nibble, proposed by Spielman and Teng, uses the \( t \)-step probability measure of the lazy random walk defined on the graph, in the \( t \)th iteration, to determine the order in which nodes are added to the set. By truncating the probability measure, nodes that have very low probability (relative to their degree) of being reached from the starting vertex are left out of the constructed subset. The reason for doing this is that these nodes are only weakly connected to the rest of the set, indicating that they are not part of a low-conductance set. We will illustrate how the algorithm works on the dumbbell graph. First, we need to determine the transition probabilities of the lazy random walk on the dumbbell graph. In this case the single step transition probabilities are given by

\[
p(a_i, a_j) = \begin{cases} 
\frac{1}{2} & \text{if } i = j, \\
\frac{1}{2} \frac{1}{n-1} & \text{if } i \neq n, i \neq j, \\
\frac{1}{2} \frac{1}{m} & \text{if } i = n, i \neq j, \\
0 & \text{else},
\end{cases}
\]  

(5.2)

\[
p(a_n, b_j) = \begin{cases} 
\frac{1}{2} \frac{1}{m} & \text{if } i = 1, \\
0 & \text{else},
\end{cases}
\]  

(5.3)

\[
p(b_j, b_j) = \begin{cases} 
\frac{1}{2} \frac{1}{m} & \text{if } i = 1, \\
\frac{1}{2} \frac{1}{n-1} & \text{if } i \neq 1, i \neq j, \\
\frac{1}{2} \frac{1}{m} & \text{if } i = 1, i \neq j, \\
0 & \text{else},
\end{cases}
\]  

(5.4)

\[
p(b_1, a_i) = \begin{cases} 
\frac{1}{2} \frac{1}{m} & \text{if } i = n, \\
0 & \text{else}. 
\end{cases}
\]  

(5.5)
To keep things tractable, we will neglect the truncation operation, setting \( \epsilon = 0 \) in (4.9), and work with the pure \( t \)-step transition probabilities \( p^t_a(x) \) for \( x \in \{a_1, ..., a_n, b_1, ..., b_m\} \). We choose conductance threshold \( \phi^* \) between \( \frac{1}{n^{n+1} - n} \) and \( \frac{1}{n-1} \). This choice of stopping criterion will force the procedure to terminate when the obvious cluster \( \{a_1, ..., a_n\} \) is found. Using \( a_1 \) as starting vertex, we find the single step probability

\[
p^1_{a_1}(x) = \begin{cases} 
\frac{1}{2} & \text{if } x = a_1, \\
\frac{1}{2^{(n-1)}} & \text{if } x = a_i, i \neq 1, \\
0 & \text{else}.
\end{cases}
\]

Hence the support set of the \( t \)-step distribution, sorted in ascending order of relative probability, to be checked for conductance is

\[
S^1 = \{a_1, a_2, ..., a_n\},
\]

or separately,

\[
S^1_1 = \{a_1\},
S^1_2 = \{a_1, a_2\},
\]

\[
\vdots
\]

\[
S^1_{n-1} = \{a_1, ..., a_{n-1}\},
S^1_n = \{a_1, ..., a_n\},
\]

where we use the abbreviation \( S^t_j = S_j(p^t_{a_1}) \). Using Table 5.1 we consecutively find,

\[
\begin{align*}
\phi(S^1_1) &= 1, \\
\phi(S^1_2) &= \frac{n - 2}{n - 1}, \\
\vdots \\
\phi(S^1_{n-1}) &= \frac{1}{n - 1}, \\
\phi(S^1_n) &= \frac{1}{n^2 - n - 1}.
\end{align*}
\]

None of the sets \( S^1_j, j \leq n - 1 \) satisfies the conductance constraint. However, \( S^1_n \) does, and hence the procedure is terminated and \( C = S^1_n = \{a_1, ..., a_n\} \) is returned by the algorithm. Thus, in the case of a dumbbell graph, the Spielman-Teng algorithm produces exactly the cluster we expected, namely the cluster with lowest conductance, proving its effectiveness. Note that the low-conductance cluster of the dumbbell graph is found by the Spielman-Teng algorithm in the first iteration, that is \( t = 1 \), independent of the values of \( n \) and \( m \). Nevertheless, \( n \) different subsets, namely the sets described in (5.8), need to be checked for conductance before the right cluster is found.

### 5.1.2 Andersen-Peres

In Chapter 2, we explained that the Andersen-Peres algorithm is based on the volume biased evolving set process (ESP), which is a Markov chain on the subsets of all vertices of the graph.
CHAPTER 5. STYLISED GRAPHS

Remember that the probability of the random walk being in \( y \), entering (or staying) in set \( S \) after taking one step was given by

\[
p(y, S) = \frac{1}{2} \left( \frac{E(y, S)}{d(y)} + 1_{y \in S} \right),
\]

and the next state followed by \( S_t \) was defined as

\[
S_{t+1} = \{ x \in V \mid p(x, S_t) \geq U \},
\]

where \( U \) is a uniform random variable on \([0,1]\). Starting the ESP in one vertex, it is clear we are able to construct this Markov chain for this stylised example. As we mentioned before, we choose \( S_0 = \{ a_1 \} \). First of all, we can write

\[
d(y) = \begin{cases} 
  n - 1 & \text{if } y = a_i, i \neq n, \\
  n & \text{if } y = a_n, \\
  m & \text{if } y = b_1, \\
  m - 1 & \text{if } y = b_i, i \neq 1.
\end{cases}
\]

Then,

\[
E(y, \{ a_1 \}) = \begin{cases} 
  1 & \text{if } y = a_i, i \neq 1, \\
  0 & \text{else,}
\end{cases}
\]

which yields

\[
p(y, \{ a_1 \}) = \begin{cases} 
  \frac{1}{2^{n-1}} & \text{if } y = a_i, i \neq 1, n, \\
  \frac{1}{2} & \text{if } y = a_1, \\
  \frac{1}{2n} & \text{if } y = a_n, \\
  0 & \text{else.}
\end{cases}
\]

Using (5.10), we find that

\[
S_{t+1} = \begin{cases} 
  \{ a_1, ..., a_n \} & \text{if } 0 \leq U \leq \frac{1}{2n}, \\
  \{ a_1, ..., a_{n-1} \} & \text{if } \frac{1}{2n} < U \leq \frac{1}{2(n-1)}, \\
  \{ a_1 \} & \text{if } \frac{1}{2} < U \leq \frac{1}{2}, \\
  \emptyset & \text{if } \frac{1}{2} < U \leq 1,
\end{cases}
\]

or equivalently by uniformity of \( U \),

\[
\mathbb{P}(S_{t+1} = S') = \begin{cases} 
  \frac{1}{2n} & \text{if } S' = \{ a_1, ..., a_n \}, \\
  \frac{1}{2^{n-1}} & \text{if } S' = \{ a_1, ..., a_{n-1} \}, \\
  \frac{n-2}{2^{n-1}} & \text{if } S' = \{ a_1 \}, \\
  \frac{1}{2} & \text{if } S' = \emptyset, \\
  0 & \text{else.}
\end{cases}
\]

These are the transition probabilities of the Markov chain being in \( \{ a_1 \} \). Next, we want to derive which states can be reached from \( S_1 = \{ a_1, ..., a_{n-1} \} \). First, we find

\[
E(y, \{ a_1, ..., a_{n-1} \}) = \begin{cases} 
  n - 2 & \text{if } y = a_i, i \neq n, \\
  n - 1 & \text{if } y = a_n, \\
  0 & \text{else,}
\end{cases}
\]
and hence,
\[
p(y, \{a_1, \ldots, a_{n-1}\}) = \begin{cases} 
\frac{2n-3}{2(n-1)} & \text{if } y = a_i, i \neq n, \\
\frac{n-1}{2m} & \text{if } y = a_n, \\
0 & \text{else.}
\end{cases}
\] (5.17)

From this equation, we are able to derive
\[
S_{t+1} = \begin{cases} 
\{a_1, \ldots, a_n\} & \text{if } 0 \leq U < \frac{n-1}{2m}, \\
\{a_1, \ldots, a_{n-1}\} & \text{if } \frac{n-1}{2m} < U \leq \frac{2n-3}{2(n-1)}, \\
\emptyset & \text{if } \frac{2n-3}{2(n-1)} < U \leq 1,
\end{cases}
\] (5.18)
giving transition probabilities
\[
P(S_{t+1} = S') = \begin{cases} 
\frac{n-1}{2m} & \text{if } S' = \{a_1, \ldots, a_n\}, \\
\frac{n^2-n-1}{2(m(n-1))} & \text{if } S' = \{a_1, \ldots, a_{n-1}\}, \\
\frac{2}{2(n-1)} & \text{if } S' = \emptyset, \\
0 & \text{else.}
\end{cases}
\] (5.19)

Next, we derive the probabilities for \(S_t = \{a_1, \ldots, a_n\}\):
\[
E(y, \{a_1, \ldots, a_n\}) = \begin{cases} 
n - 1 & \text{if } y = a_i, \\
1 & \text{if } y = b_1, \\
0 & \text{else,}
\end{cases}
\] (5.20)
and
\[
p(y, \{a_1, \ldots, a_n\}) = \begin{cases} 
\frac{2n-1}{2m} & \text{if } y = a_i, i \neq n, \\
\frac{1}{2m} & \text{if } y = a_n, \\
0 & \text{else,}
\end{cases}
\] (5.21)
which implies
\[
S_{t+1} = \begin{cases} 
\{a_1, \ldots, a_n, b_1\} & \text{if } 0 \leq U \leq \frac{1}{2m}, \\
\{a_1, \ldots, a_n\} & \text{if } \frac{1}{2m} < U \leq \frac{2n-1}{2m}, \\
\{a_1, \ldots, a_{n-1}\} & \text{if } \frac{2n-1}{2m} < U \leq 1.
\end{cases}
\] (5.22)

\[
P(S_{t+1} = S') = \begin{cases} 
\frac{1}{2m} & \text{if } S' = \{a_1, \ldots, a_n, b_1\}, \\
\frac{1-\frac{1}{2m}}{2m} - \frac{1}{2m} & \text{if } S' = \{a_1, \ldots, a_n\}, \\
\frac{1}{2m} & \text{if } S' = \{a_1, \ldots, a_{n-1}\}, \\
0 & \text{else.}
\end{cases}
\] (5.23)

The only subset reachable by the Markov chain we still need to check is \(S_t = \{a_1, \ldots, a_n, b_1\}\).

For this set,
\[
E(y, \{a_1, \ldots, a_n, b_1\}) = \begin{cases} 
n - 1 & \text{if } y = a_i, i \neq n, \\
n & \text{if } y = a_n, \\
1 & \text{if } y = b_1,
\end{cases}
\] (5.24)
giving
\[
p(y, \{a_1, \ldots, a_n, b_1\}) = \begin{cases} 
1 & \text{if } y = a_i, \\
\frac{n+1}{2m} & \text{if } y = b_1, \\
\frac{1}{2(n-1)} & \text{if } y = b_1, i \neq 1,
\end{cases}
\] (5.25)
so that,

\[ S_{t+1} = \begin{cases} \{a_1, ..., a_n, b_1, ..., b_m\} & \text{if } 0 \leq U \leq \frac{1}{2^{(m-1)}}, \\ \{a_1, ..., a_n, b_1\} & \text{if } \frac{1}{2^{(m-1)}} < U \leq \frac{m+1}{2^m}, \\ \{a_1, ..., a_n\} & \text{if } \frac{m+1}{2^m} < U \leq 1, \end{cases} \]  

and

\[ \mathbb{P}(S_{t+1} = S') = \begin{cases} \frac{1}{2^{(m-1)}} & \text{if } S' = \{a_1, ..., a_n, b_1, ..., b_m\} \\ \frac{m^2 - m - 1}{2m(m-1)} & \text{if } S' = \{a_1, ..., a_n, b_1\} \\ \frac{m-1}{2^m} & \text{if } S' = \{a_1, ..., a_n\} \\ 0 & \text{else.} \end{cases} \]  

Since \( \{a_1, ..., a_n, b_1, ..., b_m\} \) is an absorbing state (there are no external edges), this concludes our calculations for the evolving set process on the dumbbell graph. The transition diagram of the resulting Markov chain is displayed in Figure 5.2.

![Figure 5.2: Transition diagram of the evolving set process on the dumbbell graph.](image)

In order to derive the transition probabilities for the volume-biased ESP, which is used in the Andersen-Peres algorithm, we need to know the volumes \( \mu \) of the reachable states. We refer to Table 5.1 for these values.

Obviously, the state space of the Markov chain of the volume-biased ESP is the same as the one of the ESP, except for the fact that state \( \emptyset \) cannot be reached. The transition diagram of the volume-biased ESP with adjusted transition probabilities is given in Figure 5.3. We see that the desired cluster \( \{a_1, ..., a_n\} \) is a reachable state. Choosing a conductance threshold \( \phi^* \) such that \( \frac{1}{n^2 - n + 1} < \phi^* < \frac{1}{n-1} \) as stopping criterion the algorithm will return the desired cluster.

Because we have the exact transition probabilities, we are also able to say something about the speed of the algorithm. The number of required iterations in order to find the cluster...
\{a_1, ..., a_n\} is equal to the hitting time of state \{a_1, ..., a_n\} in the Markov chain underlying the volume-biased ESP starting from \{a_1\}. The following proposition formalises this.

**Proposition 5.1.** The number of iterations required by the volume-biased ESP, starting in \{a_1\}, in order to find cluster \{a_1, ..., a_n\} is distributed as

\[ T_{\{a_1\}} \overset{\text{d}}{=} 1 + N^*, \quad (5.28) \]

where \(N^*\) is geometrically distributed with parameter \(\frac{n^2-n-1}{2n(n-1)}\). The expected number of iterations to reach state \{a_1, ..., a_n\} is therefore

\[ \mathbb{E}T_{\{a_1\}} = \frac{q}{1-q} + 1 = \frac{2n(n-1)}{n^2-n+1} \quad (5.29) \]

where \(q = \frac{n^2-n-1}{2n(n-1)}\).

**Proof.** Let \(T_{\{a_1\}}\) denote the hitting time of \{a_1, ..., a_n\} starting from \{a_1\}, and let \(T_{\{a_1, ..., a_{n-1}\}}\) denote the hitting time of \{a_1, ..., a_n\} starting from \{a_1, ..., a_{n-1}\}. We have the relation

\[ T_{\{a_1, ..., a_{n-1}\}} \overset{\text{d}}{=} 1 + N, \quad (5.30) \]

where \(N\) is a geometrically distributed random variable with parameter \(q = \frac{n^2-n-1}{2n(n-1)}\), since the Markov chain returns to \{a_1, ..., a_{n-1}\} (failure) with probability \(q\), while making the transition to \{a_1, ..., a_n\} (success) has probability \(1 - q\). Let \(p_1 = \frac{n-2}{2n(n-1)}\) and \(p_2 = \frac{1}{2n}\). Then we furthermore find that \(T_{\{a_1\}} - 1\) equals zero with probability \(1 - p_1 - p_2\), \(T_{\{a_1\}}\) with probability \(p_1\) and \(T_{\{a_1, ..., a_{n-1}\}}\) with probability \(p_2\). Using probability generating functions, this gives

\[ P_{T_{\{a_1\}}}(z) = z \cdot \left( (1 - p_1 - p_2) \cdot P_0(z) + p_1 P_{T_{\{a_1\}}} (z) + p_2 P_{T_{\{a_1, ..., a_{n-1}\}}} (z) \right), \quad (5.31) \]

\[ = z \cdot \left( (1 - p_1 - p_2) \cdot 1 + p_1 P_{T_{\{a_1\}}} (z) + p_2 P_{T_{\{a_1, ..., a_{n-1}\}}} (z) \right), \quad (5.32) \]

\[ P_{T_{\{a_1, ..., a_{n-1}\}}}(z) = z \cdot P_N(z) = \frac{1-q}{1-qz} \cdot z. \quad (5.33) \]
Equation (5.31) can be rewritten as

\[ (1 - p_1 z) P_{\{a_1\}}(z) = z \cdot \left( 1 - p_1 - p_2 + p_2 P_{\{a_1, \ldots, a_{n-1}\}}(z) \right), \]  

or

\[ P_{\{a_1\}}(z) = \frac{z}{1 - p_1 z} \cdot \left( 1 - p_1 - p_2 + \frac{1 - q}{1 - q z} p_2 z \right). \]  

Substituting \( p_1, p_2 \) and \( q \) yields

\[ P_{\{a_1\}}(z) = \frac{(n^2 - n + 1)z}{2n(n - 1) - (n^2 - n - 1)z}, \]  

and

\[ \frac{n^2 - n + 1}{2n(n - 1)} z, \]  

where again \( q = \frac{n^2 - n - 1}{2n(n - 1)}. \) Since \( \frac{1 - q}{1 - q z} \) is the probability generating function of the geometric distribution with parameter \( q. \) This completes the proof. \( \square \)

5.1.3 Bounds on mixing time

Consider a lazy random walk defined on the graph in Figure [5.1], starting in the leftmost vertex. To escape the left clique of size \( n, \) the walk first needs to pass through \( a_n, \) the vertex with a connection to the other clique, which happens with probability \( \frac{1}{2} \frac{n - 1}{n(n - 1)}, \) after which the transition to \( b_1 \) has to be made, which has probability \( \frac{1}{2n} \) of occurring. Combining these steps implies that leaving the clique in two steps has probability \( \frac{1}{4} \frac{n - 1}{n(n - 1)} \), which is very small. We are even able to argue that the number of steps before reaching \( a_n \) is geometrically distributed (plus 1) with parameter \( \frac{2n - 2}{2n(n - 1)}, \) which is very close to 1, implying that it takes a large number of steps to reach \( a_n, \) let alone making the transition to \( b_1 \) directly afterwards. This illustrates how the bottleneck of the dumbbell graph affects the hitting times of vertices in the other clique and thereby the diffusion of the probability mass. This idea is captured in the concept of mixing times, see Definition 2.5. Because the dumbbell graph admits an explicit expression for the conductance \( \phi_G, \) we are able to derive bounds on the mixing time of the random walk defined on this type of graph, using the theory discussed in Chapter 7 and 10 of [LPW06].

The following theorem gives a lower bound for the mixing time.

**Theorem 5.1** (See [LPW06] p.55 and p.89).

\[ \tau_{\text{mix}} = \tau(1/4) \geq \frac{1}{4\phi_G}, \]  

where mixing times for general \( \varepsilon \) are derived by

\[ \tau(\varepsilon) \leq \left\lceil \log_2 \varepsilon^{-1} \right\rceil \tau_{\text{mix}}. \]  

We will use \( \tau_{\text{mix}} = \tau(1/4) \) in the remainder of our calculations, because \( \tau(\varepsilon) \) differs from \( \tau_{\text{mix}} \) only by a factor and we are mainly interested in the order of magnitude of the mixing time. Denote by \( E \) the hitting time of vertex \( y, \) starting from \( x \) and

\[ T_{\text{hit}} := \max_{x, y \in V} E_{x} T_{y}. \]  

This second theorem gives us an upper bound for the mixing time.
Theorem 5.2 (See [LPW06] p.134). Consider a lazy random walk on a graph $G$. Then

$$\tau_{mix} \leq 2T_{hit} + 1. \quad (5.42)$$

Applying these two theorems to our case yields the following result.

**Proposition 5.2.** Consider the lazy random walk defined on the dumbbell graph with cliques of size $n$ and $m$ respectively, then

$$O(\min\{n, m\}^2) \leq \tau_{mix} \leq O(\max\{n, m\}^2). \quad (5.43)$$

Moreover, if $n$ and $m$ are of the same order, then

$$\tau_{mix} = O(n^2). \quad (5.44)$$

**Proof.** The lower bound follows directly by combining Theorem 5.1 and Equation (5.1):

$$\tau_{mix} \leq \frac{m^2 - m + 1}{4} = O(m^2) = O(\min\{n, m\}^2). \quad (5.45)$$

We use Theorem 5.2 to derive an upper bound for $\tau_{mix}$. Without proof, we state that $T_{hit} = \max\{E_{a_1}T_{b_m}, E_{b_2}T_{a_1}\}$. By symmetry, we also have the following equalities:

$$E_{a_1}T_{b_m} = E_{a_1}T_{b_m}, \quad \text{for } j = 1, \ldots, n - 1,$n
$$E_{b_2}T_{b_m} = E_{b_2}T_{b_m}, \quad \text{for } j = 2, \ldots, m - 1. \quad (5.46)$$

We can now derive $E_{a_1}T_{b_m}$ (note that $E_{b_2}T_{a_1}$ may be found by switching variables $m$ and $n$). Using the transition probabilities of the lazy random walk in (5.2)-(5.5) and eliminating the second subscript, we find

$$E_{a_1}T_{b_m} = 1 + \frac{1}{2}E_{a_1}T_{b_m} + \sum_{j=2}^{n-1} \frac{E_{a_1}T_{b_m}}{2(n-1)} + \frac{1}{2(n-1)} E_{a_1}T_{b_m},$$

$$= 1 + \frac{1}{2}E_{a_1}T_{b_m} + \frac{n - 2}{2(n-1)} E_{a_1}T_{b_m} + \frac{1}{2(n-1)} E_{a_1}T_{b_m},$$

$$= 1 + \frac{2n - 1}{2(n-1)} E_{a_1}T_{b_m} + \frac{1}{2(n-1)} E_{a_1}T_{b_m}, \quad (5.47)$$

$$E_{a_2}T_{b_m} = 1 + \sum_{j=1}^{n-1} \frac{E_{a_2}T_{b_m}}{2n} + \frac{1}{2} E_{a_2}T_{b_m} + \frac{1}{2n} E_{b_2}T_{b_m},$$

$$= 1 + \frac{n - 1}{2n} E_{a_2}T_{b_m} + \frac{1}{2} E_{a_2}T_{b_m} + \frac{1}{2n} E_{b_2}T_{b_m}, \quad (5.48)$$

$$E_{b_1}T_{b_m} = 1 + \frac{1}{2m} E_{a_2}T_{b_m} + \frac{1}{2} E_{b_1}T_{b_m} + \sum_{j=2}^{m-1} \frac{E_{b_1}T_{b_m}}{2m},$$

$$= 1 + \frac{1}{2m} E_{a_2}T_{b_m} + \frac{1}{2} E_{b_1}T_{b_m} + \frac{m - 2}{2m} E_{b_1}T_{b_m}, \quad (5.49)$$

$$E_{b_2}T_{b_m} = 1 + \frac{1}{2(m-1)} E_{b_1}T_{b_m} + \frac{1}{2} E_{b_2}T_{b_m} + \sum_{j=2}^{m-1} \frac{E_{b_2}T_{b_m}}{2(m-1)},$$

$$= 1 + \frac{1}{2(m-1)} E_{b_1}T_{b_m} + \frac{1}{2} E_{b_2}T_{b_m} + \frac{m - 3}{2(m-1)} E_{b_2}T_{b_m}. \quad (5.50)$$
Solving this system of linear equations gives us
\[ E_{a_1} T_{b_w} = 2m + 2(n^2 - 1) + \frac{4(n^2 - n + 2)}{m} = O(n^2 + m), \] (5.51)
and hence
\[ E_{b_w} T_{a_1} = 2n + 2(m^2 - 1) + \frac{4(m^2 - m + 2)}{n} = O(m^2 + n). \] (5.52)
We find
\[ T_{hit} = O(\max\{n, m\}^2) \] (5.53)
and substituting this into Theorem 5.2 yields the required upper bound.

### 5.2 Grid network

In this section we consider a 2D grid network, in which nodes are only connected to the direct left, right, upper and lower neighbours. Nodes are labeled according to their x- and y-coordinate, where the middle node of the grid is considered the origin \((0, 0)\). We let both algorithms run, starting from the origin \((0, 0)\). In particular, we are interested in sufficiently large grids, avoiding boundary scenarios. We mention beforehand that, although the 2D grid is a canonical example to many algorithms, it is not an obvious test case for clustering algorithms, because there is only one clear cluster, which is the entire grid.

#### 5.2.1 Spielman-Teng

As we have seen in Chapter 2, since the \(t\)-step distribution plays a central role in this algorithm, we will first derive an exact expression for this probability measure, which is possible for a grid. After that, some results on the spatial behaviour and the speed of the algorithm are presented.

**t-step probability distribution**

Consider a two-dimensional grid of nodes of dimension \(2n + 1\) by \(2n + 1\), where we assume \(n > t\). Since the algorithm is able to reach only nodes \((x, y)\) with \(x, y \leq t\) within \(t\) steps, this excludes boundary scenarios. Each node is labeled according to its coordinates \((x, y)\), \(x, y \in \{-n, ..., n\}\) and we take node \((0, 0)\) as our starting position. Each node is adjacent only to its (four) direct neighbours. Consider the lazy random walk defined on this grid. The transition probabilities are

\[ p((x_1, y_1), (x_2, y_2)) = \begin{cases} 
\frac{1}{2}, & \text{if } (x_1, y_1) = (x_2, y_2), \\
\frac{1}{8}, & \text{if } (x_1, y_1) \sim (x_2, y_2), \\
0, & \text{else.}
\end{cases} \] (5.54)

We next derive an explicit expression for the probability that the random walk is at node \((x, y)\) after \(t\) steps.

**Proposition 5.3.** Let \((x, y) \in \{-n, ..., n\}^2\). Then the \(t\)-step probability of the lazy random walk starting in \((0,0)\) equals

\[ p_{(0,0)}^t((x, y)) = \left(\frac{1}{4}\right)^{|x|+|y|} \left(\frac{1}{2}\right)^t \sum_{m=0}^{k} \sum_{n=0}^{k-m} \binom{t}{m+|x|, m, n + |y|, n, t - |x| - |y| - 2m - 2n} \left(\frac{1}{16}\right)^{m+n}. \]
where $k = \left\lfloor \frac{t - (|x| + |y|)}{2} \right\rfloor$.

Proof. Without loss of generality, assume that $x, y \geq 0$. This assumption can later easily be extended to negative values of $x$ and $y$ by symmetry of the grid. To determine the probability of being in $(x, y)$ after $t$ step, we need to determine all possible paths from $(0, 0)$ to $(x, y)$ of length $t$ and sum up the probabilities of these paths. Let $r, l, u, d$ denote the number of transitions made to the right, left, up, down, respectively. Let $s$ be the number of times the random walk stayed in the same position. For a path to be a valid $t$-step path from $(0, 0)$ to $(x, y)$ is has to satisfy the constraints

$$x = r - l,$$
$$y = u - d,$$
$$t = r + l + u + d + s. \quad (5.56)$$

Or,

$$l = r - x,$$
$$d = u - y,$$
$$s = t + x + y - 2r - 2u. \quad (5.57)$$

Let $k$ denote the number of redundant pairs of steps (either up and down or left and right). Since we need at least $x + y$ steps already to reach $(x, y)$, this is maximally

$$k = \left\lfloor \frac{t - (x + y)}{2} \right\rfloor.$$

The maximum number of extra steps to the right $m = r - x$ is bounded by $k$ (since you also need $m$ steps back to the left). Conditioned on $m$, the maximum number of extra steps up is hence $k - m$. The gap between $t$ and the total number of ‘real’ transitions is filled up by redundant steps (staying at the same position). The number of paths with $m$ steps to the right, $m - x$ steps to the left, $n$ steps up, $n - y$ steps down, and $t + x + y - 2m - 2n$ redundant steps is equal to the multinomial coefficient

$$\binom{t}{m, m - x, n, n - y, t + x + y - 2m - 2n}, \quad (5.58)$$

while the probability of such a path is

$$\left( \frac{1}{8} \right)^m \left( \frac{1}{8} \right)^{m-x} \left( \frac{1}{8} \right)^n \left( \frac{1}{8} \right)^{n-y} \left( \frac{1}{2} \right)^{t+x+y-2m-2n} \quad (5.59)$$

$$= \left( \frac{1}{8} \right)^{2m+2n-x-y} \left( \frac{1}{2} \right)^{x+y-2m-2n} \left( \frac{1}{2} \right)^t \quad (5.60)$$

$$= 4^{x+y} \left( \frac{1}{2} \right)^t \left( \frac{1}{16} \right)^{m+n}. \quad (5.61)$$
Hence, summing over all possible combinations yields

\[ p_{(0,0)}^t((x,y)) = 4^{x+y} \left( \frac{1}{2} \right)^t \sum_{m=x}^{x+k} \sum_{n=y}^{y+k-m} \left( m, m-x, n, n-y, t + x + y - 2m - 2n \right) \left( \frac{1}{16} \right)^{m+n}, \]

where \( k = \left\lfloor \frac{t-(x+y)}{2} \right\rfloor \). Some rewriting gives

\[ p_{(0,0)}^t((x,y)) = \left( \frac{1}{4} \right)^{x+y} \left( \frac{1}{2} \right)^t \sum_{m=0}^{k} \sum_{n=0}^{k-m} \left( m, m+x, n, n+y, t - x - y - 2m - 2n \right) \left( \frac{1}{16} \right)^{m+n}. \]

For the general case, where \( x, y \in \{-n, ..., n\} \), we find by symmetry (5.55).

Note that there is some more symmetry to this function, namely \( p_{(0,0)}^t((x,y)) = p_{(0,0)}^t((y,x)). \)

Possible outcomes

In each iteration step of the Spielman-Teng algorithm, the \( t \)-step probability from the starting vertex to any other node is computed. The nodes in the support set, that is all nodes with positive probability of being reached after \( t \) steps, are ordered by \( \Pi \) according to the following rule:

\[ \frac{p(\Pi(j))}{d(\Pi(j))} \geq \frac{p(\Pi(j+1))}{d(\Pi(j+1))}. \]

Here, we will neglect the truncation operation and work with the pure probability measure. Furthermore, in case of a grid, we find \( d(y) = 4 \), for all nodes \( y \) of the grid. Hence, the support set is sorted according to the original probability distribution. To get a feeling of how Nibble tries to construct a low-conductance set, we take a look at the case \( t = 3 \). It should be clear that the support set equals

\[ S^3 = \{ (x,y) | |x| + |y| \leq 3 \}. \]

The probability measure \( \{p^3(x,y)\}_{(x,y) \in S^3} \) is displayed in Figure 5.4. The nodes with the same probability are grouped as

\[ S^3 = S_1^3 \cup S_2^3 \cup S_3^3 \cup S_4^3 \cup S_5^3 \cup S_6^3, \]

with

\[ S_1^3 = \{ (0,0) \}, \]
\[ S_2^3 = \{ (1,0), (-1,0), (0,1), (0,-1) \}, \]
\[ S_3^3 = \{ (1,1), (1,-1), (-1,1), (-1,-1) \}, \]
\[ S_4^3 = \{ (2,0), (-2,0), (0,2), (0,-2) \}, \]
\[ S_5^3 = \{ (2,1), (2,-1), (-2,1), (-2,-1), (1,2), (1,-2), (-1,2), (-1,-2) \}, \]
\[ S_6^3 = \{ (3,0), (-3,0), (0,3), (0,-3) \} \]

and \( p(S_1^3) > p(S_{i+1}^3) \). We call this partition of the support set the rings of the \( t \)-step distribution. In the Spielman-Teng algorithm, the nodes would be included in this order. That is,
the algorithm first checks whether $S^3_1$ meets the conductance constraints, then $S^3_1 \cup S^3_2$, all the way up to $S^3$.

If one of these sets indeed has lower conductance than the set threshold, the procedure terminates. Otherwise, it proceeds with the 4-step distribution. Continuing this process numerically shows that if the rings of the $t$-step distribution are $S^t_1, \ldots, S^t_k$, then the first $k$ rings of the $t + m$-step distribution, for $m \in \mathbb{N}$, are equal to $S^t_1, \ldots, S^t_k$. Note that this is generally not the case, because truncated probabilities might change the sets. More precisely, define

$$Q(i, j) = \{ (x, y) \mid |x| + |y| \leq i \land |x| - |y| = j \},$$

and

$$R(i, j) = \bigcup_{1 \leq k \leq i - 1, j - k \equiv 0 \pmod{2}} Q(k, l) \cup \bigcup_{l \leq i, j - l \equiv 0 \pmod{2}} Q(i, l)$$

(5.69)

where $i \in \mathbb{N}_+$ and $0 \leq j \leq i$, $j$ even if $i$ even and $j$ odd if $i$ odd. Then in the $t^{th}$ iteration, the following sets are checked consecutively,

$$R(0, 0), R(1, 1), R(2, 0), R(2, 2), \ldots, R(t, t \mod{2}), \ldots, R(t, t)$$

(5.70)

Every outcome of the Spielman-Teng algorithm is of the form (5.69). This leads us to the following proposition.

**Proposition 5.4.** The number of subsets to be checked for conductance in the Spielman-Teng algorithm in the $t^{th}$ iteration equals

$$N_t = \begin{cases} \left( \frac{t}{2} + 1 \right)^2, & \text{if } t \text{ even}, \\ \frac{1}{4}(t+1)(t+3), & \text{if } t \text{ odd}. \end{cases}$$

(5.71)
Proof. For all $i \in \mathbb{N}$, the possible outcomes of the form $R(i, \cdot)$, which we call the sets at level $i$, are

$$R(i, i \mod 2), R(i, i \mod 2 + 2), ..., R(i, i). \quad (5.72)$$

For $i$ even, these are $\frac{i}{2} + 1$ sets. For $i$ odd, this number is $\frac{i + 1}{2}$, or in general

$$\left\lfloor \frac{i}{2} \right\rfloor + 1. \quad (5.73)$$

Since in the $t^{th}$ iteration, the sets of all levels up to $t$ need to be checked, this total number equals

$$N_t = \sum_{k=0}^{t} \left( \left\lfloor \frac{k}{2} \right\rfloor + 1 \right), \quad (5.74)$$

and some basic calculus yields

$$N_t = 2 \sum_{k=0}^{\lfloor t/2 \rfloor} k - \left\lfloor \frac{t}{2} \right\rfloor 1_{(t \text{ even})} + (t + 1) \quad (5.75)$$

$$= \frac{t}{2} \left( \left\lfloor \frac{t}{2} \right\rfloor + (t + 1) \right) \quad (5.76)$$

$$= \begin{cases} \left( \frac{t}{2} \right)^2 + (t + 1), & \text{if } t \text{ even}, \\ \left( \frac{t-1}{2} \right) \left( \frac{t+1}{2} \right) + (t + 1), & \text{if } t \text{ odd}, \end{cases} \quad (5.77)$$

$$= \begin{cases} \left( \frac{t}{2} + 1 \right)^2, & \text{if } t \text{ even}, \\ \frac{1}{4} (t + 1) (t + 3), & \text{if } t \text{ odd}, \end{cases} \quad (5.78)$$

which completes the proof. \qed

### Speed of the algorithm

Using the ordering given in (5.70), we can write down a recursive relation for the value of the cut and volume of a set $R(i, j)$ and therefore also the conductance. That is,

$$\partial(R(0, 0)) = \partial(\{(0, 0)\}) = 1, \quad \partial(R(1, 1)) = 12, \quad (5.79)$$

$$\partial(R(i, j)) = \partial(R(i, j)), \quad \text{if } j \geq 2, \quad (5.80)$$

$$\partial(R(i, i \mod 2)) = \partial(R(i - 1, i - 1)), \quad (5.81)$$

$$\partial(R(i, i)) = \partial(R(i, i - 2)) + 8, \quad (5.82)$$

and

$$\mu(R(0, 0)) = \mu(\{(0, 0)\}) = 4, \quad \mu(R(1, 1)) = 20, \quad (5.83)$$

$$\mu(R(i, j)) = \mu(R(i, j - 2)) + 32, \quad \text{if } 2 \leq j < i, \quad (5.84)$$

$$\mu(R(i, 0)) = \mu(R(i - 1, i - 1)) + 16 \quad \text{if } i \text{ even}, \quad (5.85)$$

$$\mu(R(i, 1)) = \mu(R(i - 1, i - 1)) + 32 \quad \text{if } i \text{ odd}, \quad (5.86)$$

$$\mu(R(i, i)) = \mu(R(i, i - 2)) + 16. \quad (5.87)$$

Hence, we can find $\phi(R(i, j)) = \partial(R(i, j))/\mu(R(i, j))$. The resulting speed of the algorithm is plotted in Figure 5.5 In the $t^{th}$ iteration, the collection $C_t = \{R(0, 0), R(1, 1), ..., R(t, t)\}$ is
Figure 5.5: Number of sets \((R(0,0), R(1,1), \ldots)\) plotted against the lowest conductance value found for the Spielman-Teng algorithm.

Figure 5.6: Number of sets \((R(0,0), R(1,1), \ldots)\) plotted against the lowest conductance value found for the Spielman-Teng algorithm without restarts.

checked for conductance, whereas in the \(t + 1^{th}\) this is \(C_{t+1} = \{R(0,0), R(t,t), \ldots, R(t + 1, t + 1)\}\). Since in this case \(C_t \subset C_{t+1}\), the speed can be improved by not checking all previous sets again, but continuing with checking \(C_{t+1} \setminus C_t\). The speed of this adapted procedure is plotted in Figure 5.6. Again note that we cannot do this in general due to the truncation operation in the original algorithm.
5.2.2 Andersen-Peres

State space of the Markov chain

Like in case of the dumbbell graph, we would like to construct the Markov chain underlying the evolving set process on the 2D grid. Starting the ESP in one node, it is clear that the current state of the Markov chain cannot just be any subset of all nodes. For instance, by symmetry of the grid, the current set $S_t$ is by definition symmetric around $(0, 0)$. That is,

$$(x, y) \in S_t \iff (-x, y) \in S_t \iff (x, -y) \in S_t \iff (-x, -y) \in S_t.$$  \hspace{2cm} (5.88)

A few possible realisations of the algorithm are presented in Figure 5.7.

More specifically, we can enumerate the exact subsets of the grid points, or equivalently the states of the Markov chain, that are reachable by the process, starting with $S_0 = \{(0, 0)\}$. The next result describes all possible configurations.

**Proposition 5.5.** Let $x = (0, 0)$ be the starting vertex and $G$ a sufficiently large grid. Denote the ESP by $\{S_t\}_{t \geq 0}$ with $S_0 = \{(0, 0)\}$. Then the state space of the Markov chain corresponding to the ESP is,

$$S = \{S(i, j) \mid i \in \mathbb{N}_+, j = 1, 2, ..., i\} \cup \emptyset,$$  \hspace{2cm} (5.89)

where

$$S(i, j) = \{(x, y) \mid |x| + |y| \leq i + j - 2, |x| \leq i - 1, |y| \leq i - 1\}.$$  \hspace{2cm} (5.90)

Furthermore, all states have positive probability of being visited.

**Proof.** Let us call $\{S(i, j)\}_{i=1}^\infty$ the states at level $i$. These can be interpreted as the configurations of radius $i - 1$ the evolving set is allowed to take. We now prove the hypothesis by induction on $i$ First, consider $i = 1, j = 1$. Then,

$$S(1, 1) = \{(x, y) \mid |x| + |y| \geq 0, |x| \leq 0, |y| \leq 0\} = \{(0, 0)\} = S_0.$$  \hspace{2cm} (5.91)
5.2. GRID NETWORK

Hence indeed \( S(1, 1) \in S \). Assume that our hypothesis is true for all levels \( i \geq k \), so the only reachable configurations of radius \( k - 1 \) are \( S(k, 1), \ldots, S(k, k) \). We prove that the only states the Markov chain can reach from level \( k \) are also of the form (5.90). Three cases can be distinguished: states reachable from \( S(k, 1) \), \( S(k, j) \) for \( 1 < j < k \) and \( S(k, k) \). First consider \( S_t = S(k, 1) \), which equals

\[
S(k, 1) = \{(x, y) \mid |x| + |y| \leq k - 1, |x| \leq k - 1, |y| \leq k - 1\} = \{(x, y) \mid |x| + |y| \leq k - 1\}. \tag{5.92}
\]

We partition the nodes of the grid into six subsets.

\[
\begin{align*}
C &= \{(x, y) \mid |x| + |y| < k - 1\}, \tag{5.93} \\
I_1 &= \{(x, y) \mid |x| + |y| = k - 1 \land (|x| = k - 1 \lor |y| = k - 1)\}, \tag{5.94} \\
I_2 &= \{(x, y) \mid |x| + |y| = k - 1 \land |x| < k - 1 \land |y| < k - 1\}, \tag{5.95} \\
O_1 &= \{(x, y) \mid |x| + |y| = k \land (|x| = k \lor |y| = k)\}, \tag{5.96} \\
O_2 &= \{(x, y) \mid |x| + |y| = k \land |x| < k \land |y| < k\}, \tag{5.97} \\
B &= \{(x, y) \mid |x| + |y| > k\}. \tag{5.98}
\end{align*}
\]

This partition of the nodes is visualised in Figure 5.8 for \( k = 5 \). Note that \( S_t = (k, 1) = C \cup I_1 \cup I_2 \). It is easily seen that

\[
E(v, S_t) = \begin{cases} 
4 & \text{if } v \in C, \\
2 & \text{if } v \in I_1, \\
1 & \text{if } v \in I_2, \\
2 & \text{if } v \in O_1, \\
1 & \text{if } v \in O_2, \\
0 & \text{else.}
\end{cases} \tag{5.99}
\]
CHAPTER 5. STYLISED GRAPHS

By the fact that \( d(v) = 4 \) for all nodes \( v \) and \( (5.9) \),

\[
p(v, S_t) = \begin{cases} 
1 & \text{if } v \in C, \\
\frac{3}{4} & \text{if } v \in I_1, \\
\frac{1}{2} & \text{if } v \in I_2, \\
\frac{1}{4} & \text{if } v \in O_1, \\
0 & \text{if } v \in O_2,
\end{cases}
\]

(5.100)

Using the description of \( S_{t+1} \) in \( (5.10) \), this yields

\[
S_{t+1} = \begin{cases} 
C \cup I_1 \cup I_2 \cup O_1 \cup O_2, & \text{if } 0 < U \leq \frac{1}{8}, \\
C \cup I_1 \cup I_2 \cup O_1, & \text{if } \frac{1}{8} < U \leq \frac{1}{4}, \\
C \cup I_1 \cup I_2, & \text{if } \frac{1}{4} < U \leq \frac{3}{8}, \\
C \cup I_1, & \text{if } \frac{3}{8} < U \leq \frac{1}{2}, \\
C, & \text{if } \frac{1}{2} < U \leq 1.
\end{cases}
\]

(5.101)

By rewriting these unions, we get

\[
C \cup I_1 \cup I_2 \cup O_1 \cup O_2 = \{ \{x, y\} \mid |x| + |y| \leq k \land |x| \leq k \land |y| \leq k \} = S(k + 1, 1), \\
C \cup I_1 \cup I_2 \cup O_1 = \{ \{x, y\} \mid |x| + |y| \leq k \land |x| \leq k - 1 \land |y| \leq k - 1 \} = S(k, 2), \\
C \cup I_1 \cup I_2 = \{ \{x, y\} \mid |x| + |y| \leq k - 1 \land |x| \leq k - 1 \land |y| \leq k - 1 \} = S(k, 1), \\
C \cup I_1 = \{ \{x, y\} \mid |x| + |y| \leq k - 1 \land |x| \leq k - 2 \land |y| \leq k - 2 \} = S(k - 1, 2), \\
C = \{ \{x, y\} \mid |x| + |y| \leq k - 2 \land |x| \leq k - 2 \land |y| \leq k - 2 \} = S(k - 1, 1).
\]

It follows that all configurations that can be reached directly after \( S(k, 1) \) are of the form \( (5.9) \). In the second case we consider \( S(k, j) \), where \( 1 < j < k \). Again, a division of the nodes can be made:

\[
C = \{ \{x, y\} \mid |x| + |y| < k + j - 2 \land |x| < k - 1 \land |y| < k - 1 \} ,
\]

(5.102)

\[
I_1 = \{ \{x, y\} \mid |x| + |y| < k + j - 2 \land (|x| = k - 1 \lor |y| = k - 1) \} ,
\]

(5.103)

\[
I_2 = \{ \{x, y\} \mid |x| + |y| = k + j - 2 \land |x| < k - 1 \land |y| < k - 1 \} ,
\]

(5.104)

\[
O_1 = \{ \{x, y\} \mid |x| + |y| = k + j - 1 \land |x| \leq k - 1 \land |y| \leq k - 1 \} ,
\]

(5.105)

\[
O_2 = \{ \{x, y\} \mid |x| + |y| = k \land (|x| = k \lor |y| = k) \} ,
\]

(5.106)

\[
B = \{ \{x, y\} \mid |x| + |y| > k + j - 1 \} .
\]

(5.107)

An example for \( k = 5, j = 3 \) is presented in Figure 5.9. Similarly,

\[
E(v, S_t) = \begin{cases} 
4 & \text{if } v \in C, \\
3 & \text{if } v \in I_1, \\
2 & \text{if } v \in I_2, \\
2 & \text{if } v \in O_1, \\
1 & \text{if } v \in O_2, \\
0 & \text{else},
\end{cases}
\]

\[
p(v, S_t) = \begin{cases} 
1 & \text{if } v \in C, \\
\frac{3}{4} & \text{if } v \in I_1, \\
\frac{1}{2} & \text{if } v \in I_2, \\
\frac{1}{4} & \text{if } v \in O_1, \\
0 & \text{if } v \in O_2.
\end{cases}
\]

(5.108)

Therefore,

\[
S_{t+1} = \begin{cases} 
C \cup I_1 \cup I_2 \cup O_1 \cup O_2 & \text{if } 0 < U \leq \frac{1}{8}, \\
C \cup I_1 \cup I_2 \cup O_1 & \text{if } \frac{1}{8} < U \leq \frac{1}{4}, \\
C \cup I_1 \cup I_2 & \text{if } \frac{1}{4} < U \leq \frac{3}{8}, \\
C \cup I_1 & \text{if } \frac{3}{8} < U \leq \frac{1}{2}, \\
C & \text{if } \frac{1}{2} < U \leq 1.
\end{cases}
\]

(5.109)
five different kinds of nodes, as depicted in Figure 5.10. Finally, we need to check which states can be reached from $S$.

Figure 5.9: Configuration $S(5,3)$. The small black dots, large black dots, black squares, white squares, white dots and small gray dots represent $C$, $I_1$, $I_2$, $O_1$, $O_2$ and $B$, respectively.

$C \cup I_1 \cup I_2 \cup O_1 \cup O_2 = \{(x,y) \mid |x| + |y| \leq k + j - 1 \wedge |x| \leq k \wedge |y| \leq k \} = S(k+1,j),$
$C \cup I_1 \cup I_2 \cup O_1 = \{(x,y) \mid |x| + |y| \leq k + j - 1 \wedge |x| \leq k - 1 \wedge |y| \leq k - 1 \} = S(k,j+1),$
$C \cup I_1 \cup I_2 = \{(x,y) \mid |x| + |y| \leq k + j - 2 \wedge |x| \leq k - 1 \wedge |y| \leq k - 1 \} = S(k,j),$
$C \cup I_1 = \{(x,y) \mid |x| + |y| \leq k + j - 3 \wedge |x| \leq k - 1 \wedge |y| \leq k - 1 \} = S(k,j-1),$
$C = \{(x,y) \mid |x| + |y| \leq k + j - 3 \wedge |x| \leq k - 2 \wedge |y| \leq k - 2 \} = S(k-1,j).$

Hence, taking one step from $S(k,j)$ also results in a set $S_{t+1}$ of the form (5.90). Finally, we need to check which states can be reached from $S_t = S(k,k)$. Here, we only have five different kinds of nodes, as depicted in Figure 5.10.

$$C = \{(x,y) \mid |x| + |y| < 2k - 2 \wedge |x| < k - 1 \wedge |y| < k - 1 \},$$
$$I_1 = \{(x,y) \mid |x| + |y| < 2k - 2 \wedge (|x| = k - 1 \vee |y| = k - 1) \},$$
$$I_2 = \{(x,y) \mid |x| = k - 1 \wedge |y| = k - 1 \},$$
$$O_1 = \{(x,y) \mid |x| + |y| < 2k \wedge (|x| = k \vee |y| = k) \}.$$

Then,

$$E(v,S_t) = \begin{cases} 
4 & \text{if } v \in C, \\
3 & \text{if } v \in I_1, \\
2 & \text{if } v \in I_2, \\
1 & \text{if } v \in O_1, \\
0 & \text{else}, 
\end{cases} \quad \Rightarrow p(v,S_t) = \begin{cases} 
1 & \text{if } v \in C, \\
0 & \text{else}. 
\end{cases} \quad (5.110)$$

So,

$$S_{t+1} = \begin{cases} 
C \cup I_1 \cup I_2 \cup O_1 & \text{if } 0 < U \leq \frac{1}{5}, \\
C \cup I_1 \cup I_2 & \text{if } \frac{1}{5} < U \leq \frac{2}{5}, \\
C \cup I_1 & \text{if } \frac{2}{5} < U \leq \frac{3}{5}, \\
C & \text{if } \frac{3}{5} < U \leq 1, 
\end{cases} \quad (5.111)$$
CHAPTER 5. STYLISED GRAPHS

Figure 5.10: Configuration $S(5, 5)$. The small black dots, large black dots, black squares, white dots and small gray dots represents $C$, $I_1$, $I_2$, $O_1$ and $B$, respectively.

where $U$ is a uniform random variable on $[0, 1]$. Again rewriting the unions gives,

$$C \cup I_1 \cup I_2 \cup O_1 = \{ (x, y) \mid |x| + |y| \leq 2k - 1 \land |x| \leq k \land |y| \leq k \} = S(k + 1, k),$$

$$C \cup I_1 \cup I_2 = \{ (x, y) \mid |x| + |y| \leq 2k - 2 \land |x| \leq k - 1 \land |y| \leq k - 1 \} = S(k, k),$$

$$C \cup I_1 = \{ (x, y) \mid |x| + |y| \leq 2k - 3 \land |x| \leq k - 1 \land |y| \leq k - 1 \} = S(k, k - 1),$$

$$C = \{ (x, y) \mid |x| + |y| \leq 2k - 3 \land |x| \leq k - 2 \land |y| \leq k - 2 \} = S(k - 1, k - 1).$$

The states reachable from $S(k, k)$ are $S(k + 1, k), S(k, k), S(k, k - 1)$ and $S(k - 1, k - 1)$, and are all of form (5.90). Hence, the only configurations reachable from states on level $k$ are of this form, proving our claim.

**Transition probabilities**

Using the analysis of the possible transitions of the Markov chain described in the previous section, we can easily derive the transition probabilities of the Markov chain. Given that $S_t = S(i, j)$, $i \in \mathbb{N}_+$, $j \in \{1, ..., i\}$, (5.101), (5.109) and (5.111) and the fact that $U$ is uniformly distributed on $[0, 1]$, imply that the transition probabilities for the original ESP on a grid are as follows.

- If $(i, j) = (1, 1)$,

$$\mathbb{P}(S_{t+1} = S') = \begin{cases} 
\frac{1}{7}, & \text{if } S' = \emptyset, \\
\frac{3}{5}, & \text{if } S' = S(1, 1), \\
\frac{1}{8}, & \text{if } S' = S(2, 1), \\
0, & \text{else.}
\end{cases}$$

(5.112)

- If $(i, j) = (2, 1)$,

$$\mathbb{P}(S_{t+1} = S(k, l)) = \begin{cases} 
\frac{3}{5}, & \text{if } (k, l) = (1, 1), \\
\frac{3}{5}, & \text{if } (k, l) = (2, 1), \\
\frac{1}{7}, & \text{if } (k, l) = (2, 2), \\
\frac{1}{8}, & \text{if } (k, l) = (3, 1), \\
0, & \text{else.}
\end{cases}$$

(5.113)
5.2. GRID NETWORK

• If $i > 2$, $j = 1$,

\[
P(S_{t+1} = S(k,l)) = \begin{cases} 
\frac{1}{4}, & \text{if } (k,l) = (i-1,1), \\
\frac{1}{4}, & \text{if } (k,l) = (i-1,2), \\
\frac{3}{8}, & \text{if } (k,l) = (i,1), \\
\frac{1}{4}, & \text{if } (k,l) = (i,2), \\
\frac{1}{8}, & \text{if } (k,l) = (i+1,1), \\
0, & \text{else.} 
\end{cases}
\] (5.114)

• If $i > 2$, $1 < j < i$,

\[
P(S_{t+1} = S(k,l)) = \begin{cases} 
\frac{1}{8}, & \text{if } (k,l) = (i-1,j), \\
\frac{1}{8}, & \text{if } (k,l) = (i,j-1), \\
\frac{1}{8}, & \text{if } (k,l) = (i,j), \\
\frac{1}{8}, & \text{if } (k,l) = (i,j+1), \\
\frac{1}{8}, & \text{if } (k,l) = (i+1,j), \\
0, & \text{else.} 
\end{cases}
\] (5.115)

• If $i = j > 2$,

\[
P(S_{t+1} = S(k,l)) = \begin{cases} 
\frac{1}{8}, & \text{if } (k,l) = (i-1,i-1), \\
\frac{1}{8}, & \text{if } (k,l) = (i,i-1), \\
\frac{1}{8}, & \text{if } (k,l) = (i,i), \\
\frac{1}{8}, & \text{if } (k,l) = (i+1,i), \\
0, & \text{else.} 
\end{cases}
\] (5.116)

This results in the transition diagram depicted in Figure 5.11. As described in the previous chapter, the AP algorithm Andersen uses the volume-biased variant of the ESP. The state space, does not change in this case, but the transition probabilities do. We already saw that the adjusted transition probabilities $P^*$ follow from the transition probabilities of the normal ESP $P$ by the relation

\[
P^*(S_{t+1} = S'|S_t = S) = \frac{\mu(S')}{\mu(S)} P(S_{t+1} = S'|S_t = S).
\] (5.117)

Furthermore,

\[
P^*(S_{t+1} = \emptyset|S_t = S(1,1)) = 0
\] (5.118)

so state $\emptyset$ cannot be reached by the volume-biased ESP. Because we are working with a grid, and $d(v) = 4$ for all nodes $v$, and hence $\mu(S) = 4|S|$, this simplifies to

\[
P^*(S_{t+1} = S'|S_t = S) = \frac{|S'|}{|S|} P(S_{t+1} = S'|S_t = S).
\] (5.119)

It may be verified that

\[|S(k,j)| = (2k-1)^2 - 2(k-j)(k-j+1).\] (5.120)

Hence, the transition probabilities corresponding to the volume-biased ESP are:
CHAPTER 5. STYLISED GRAPHS

Figure 5.11: Part of the transition diagram of the evolving set process.

- If \((i, j) = (1, 1)\),
  \[
  \mathbb{P}(S_{t+1} = S(k, l)) = \begin{cases} 
    \frac{1}{2}, & \text{if } (k, l) = (1, 1), \\
    \frac{1}{2}, & \text{if } (k, l) = (2, 1), \\
    0, & \text{else.}
  \end{cases}
  \]  
  (5.121)

- If \((i, j) = (2, 1)\),
  \[
  \mathbb{P}(S_{t+1} = S(k, l)) = \begin{cases} 
    \frac{3}{4}, & \text{if } (k, l) = (1, 1), \\
    \frac{3}{8}, & \text{if } (k, l) = (2, 1), \\
    \frac{1}{4}, & \text{if } (k, l) = (2, 2), \\
    \frac{3}{8}, & \text{if } (k, l) = (3, 1), \\
    0, & \text{else.}
  \end{cases}
  \]  
  (5.122)

- If \(i > 2, j = 1\),
  \[
  \mathbb{P}(S_{t+1} = S(k, l)) = \begin{cases} 
    \frac{2^{i+1} - 6i + 5}{2^{i+1} - 2i + 1}, & \text{if } (k, l) = (i - 1, 1), \\
    \frac{2^{i+1} - 2i - 3}{2^{i+1} - 2i + 1}, & \text{if } (k, l) = (i - 2, 1), \\
    \frac{2^{i+1} - 2i - 3}{2^{i+1} - 2i + 1}, & \text{if } (k, l) = (i, 1), \\
    \frac{2^{i+1} - 2i - 3}{2^{i+1} - 2i + 1}, & \text{if } (k, l) = (i, 2), \\
    \frac{2^{i+1} + 2i + 1}{2^{i+1} - 2i + 1}, & \text{if } (k, l) = (i + 1, 1), \\
    0, & \text{else.}
  \end{cases}
  \]  
  (5.123)
5.2. GRID NETWORK

- If \( i > 2, 1 < j < i \),
  
  \[
  \mathbb{P}(S_{t+1} = S(k,l)) = \begin{cases} 
  \frac{(2i-3)^2 - 2(i-j-1)(i-l)}{(2i-1)^2 - 2(i-j)(i-l+1)} \cdot \frac{1}{8}, & \text{if } (k,l) = (i-1,j), \\
  \frac{(2i-1)^2 - 2(i-j+1)(i-1)}{(2i-1)^2 - 2(i-j)(i-1+1)} \cdot \frac{1}{8}, & \text{if } (k,l) = (i,j), \\
  \frac{1}{8}, & \text{if } (k,l) = (i,j), \\
  \frac{(2i-1)^2 - 2(i-j-1)(i-l)}{(2i-1)^2 - 2(i-j)(i-l+1)} \cdot \frac{1}{8}, & \text{if } (k,l) = (i,j+1), \\
  \frac{(2i-1)^2 - 2(i-j+1)(i-1)}{(2i-1)^2 - 2(i-j)(i-1+1)} \cdot \frac{1}{8}, & \text{if } (k,l) = (i+1,j), \\
  0, & \text{else.} 
  \end{cases}
  \tag{5.124}
  \]

- If \( i = j > 2 \),
  
  \[
  \mathbb{P}(S_{t+1} = S(k,l)) = \begin{cases} 
  \frac{(2i-3)^2}{(2i-1)^2} \cdot \frac{1}{8}, & \text{if } (k,l) = (i-1,i-1), \\
  \frac{4i^2-4i-3}{(2i-1)^2} \cdot \frac{1}{8}, & \text{if } (k,l) = (i,i), \\
  \frac{4i^2+4i-3}{(2i-1)^2} \cdot \frac{1}{8}, & \text{if } (k,l) = (i+1,i), \\
  0, & \text{else.} 
  \end{cases}
  \tag{5.125}
  \]

**Expected hitting times**

With the help of the transition probabilities described in the previous section, we can derive some results on the speed of both the normal and volume-biased evolving set process on a grid. This can be done in terms of expected hitting times on the levels of the Markov chain. Let \( \mathbb{E}T_n(i,j) \) be the expected number of steps it takes the Markov chain starting from state \( S(i,j) \) to reach either one of the states \( S(n,1), ..., S(n,n) \). We are interested in the expected number of steps it takes the (volume-biased) ESP, starting from one node to grow into a configuration in level \( n \in \mathbb{N} \), that is \( \mathbb{E}T_n(1,1) \). To simplify the notation, define

\[
\mathbb{P}_{S(i,j)}(S(k,l)) = \mathbb{P}(S_{t+1} = S(k,l) \mid S_t = S(i,j)). \tag{5.126}
\]

The following system of linear equations fixes the relations between \( \mathbb{E}T_n(i,j) \) for all \( 1 \leq i < n, 1 \leq j \leq i \):

\[
\begin{align*}
\mathbb{E}T_n(i,j) &= \mathbb{P}_{S(i,j)}(S(i,j)) \cdot \mathbb{E}T_n(i,j) + \mathbb{P}_{S(i,j)}(S(i-1,j)) \cdot \mathbb{E}T_n(i-1,j) \\
&+ \mathbb{P}_{S(i,j)}(S(i-1,j+1)) \cdot \mathbb{E}T_n(i-1,j+1) + \mathbb{P}_{S(i,j)}(S(i,j-1)) \cdot \mathbb{E}T_n(i,j-1) \\
&+ \mathbb{P}_{S(i,j)}(S(i,j+1)) \cdot \mathbb{E}T_n(i,j+1) + \mathbb{P}_{S(i,j)}(S(i+1,j)) \cdot \mathbb{E}T_n(i+1,j) \\
&\forall 1 \leq i \leq n-2, 1 \leq j \leq i.
\end{align*}
\]

\[
\begin{align*}
\mathbb{E}T_n(n-1,j) &= \mathbb{P}_{S(n-1,j)}(S(n-1,j)) \cdot \mathbb{E}T_n(n-1,j) + \mathbb{P}_{S(n-1,j)}(S(n-2,j)) \cdot \mathbb{E}T_n(n-2,j) \\
&+ \mathbb{P}_{S(n-1,j)}(S(n-2,j+1)) \cdot \mathbb{E}T_n(n-2,j+1) \\
&+ \mathbb{P}_{S(n-1,j)}(S(n-1,j-1)) \cdot \mathbb{E}T_n(n-1,j-1) \\
&+ \mathbb{P}_{S(n-1,j)}(S(n-1,j+1)) \cdot \mathbb{E}T_n(n-1,j+1) \\
&\forall 1 \leq j \leq n-1.
\end{align*}
\]

Since the normal ESP contains an absorbing state, namely \( \emptyset \), the probability measure \( \mathbb{P}_{S(1,1)} \) needs to be adjusted, conditioning on the event that the Markov chain never enters state \( \emptyset \). Hence, instead of the transition probabilities given in (5.112), we take

\[
\mathbb{P}_{S(1,1)}(S(i,j)) = \begin{cases} 
  \frac{3}{4} & \text{if } (i,j) = (1,1), \\
  \frac{1}{4} & \text{if } (i,j) = (2,1), \\
  0 & \text{else.} 
  \end{cases}
  \tag{5.127}
\]
Likewise, the system of equations for the hitting times of the volume-biased ESP can be written down by replacing all probability measures $P$ by $P^*$. In this case, we do not need to adjust $P^*_{S(1,1)}$ for state $\emptyset$ cannot be reached. Solving this system of equations for different values of $n$ and for both the normal and volume-biased evolving set process, we find the hitting times as plotted in Figure 5.12. It is clear that the volume-biased is much faster in generating an $n$-level set. This of course makes sense, because in the volume-biased ESP transitions to larger sets are preferred above transitions to smaller sets.

### 5.3 Conclusions

Comparing the analytic results of the Spielman-Teng and Andersen-Peres algorithm for both graph types, we can make some observations. First of all, we note that both algorithms only allow a certain subset of all possible subsets of the vertices of the graph. For the dumbbell graph, all reachable subsets are connected, while for the grid, the algorithms find only solutions that are symmetric around the origin. Furthermore, it can be seen that the Spielman-Teng algorithm checks the same subsets as the Andersen-Peres algorithm, and a few more, like $R(5,3)$ in the grid. Since ST is a deterministic method, the expected time until we hit a set of certain conductance can be computed exactly, as seen in Figures 5.5 and 5.6. On the other hand, the AP algorithm is a random process. Therefore, we can only derive the distribution of the time before hitting a set of certain conductance, as we have done for the dumbbell graph. For the grid, we were only able to derive expected hitting times. Last, we find that our intuition is right and that the volume biased ESP is indeed much faster in generating a large set instead of the normal ESP.
Chapter 6

General observations on local algorithms

In the previous chapter we obtained insight in the processes behind the Spielman-Teng and Andersen-Peres clustering algorithms. In this chapter, we present some further observations on both algorithms. We will discuss the properties of monotonicity and connectivity of clusters found by the algorithms. We further describe how we are able to extend and modify the algorithms to fit our application by the incorporation of constraints, changing the stopping criteria and inclusion of spatial data.

6.1 Properties of Spielman-Teng

6.1.1 Simplified algorithm

In the Nibble algorithm, the following procedure is executed until a set of sufficiently low conductance, say $\phi^*$ is found, neglecting the truncation operation described in (2.93).

1. Compute the $t$-step distribution of the lazy random walk on the graph $G = (V, E)$, starting from $x_0 \in V$, say

   \[ p^t_{x_0} = \{p^t_{x_0}(1), ..., p^t_{x_0}(n)\}. \]  

   (6.1)

2. Define the support set of $p^t_{x_0}$:

   \[ S^t = \{i \in V | p^t_{x_0}(i) > 0\} \]  

   (6.2)

3. For all $i \in V$, compute

   \[ r^t(i) = \frac{p^t_{x_0}(i)}{d(i)}, \]  

   (6.3)

   and let $\Pi^t$ be the permutation defined on $S^t$ such that

   \[ r^t(\Pi^t(i)) \ge r^t(\Pi^t(i + 1)). \]  

   (6.4)

4. For $j = 1, ..., |S^t|$, set $S^j = \{\Pi^t(1), ..., \Pi^t(j)\}$. If $\phi(S^j) \le \phi^*$, terminate procedure.

5. If none of $S^1, ..., S^{|S^t|-1}, S^t$ meet the conductance requirements, repeat step 1 to 4, but then with the $(t + 1)$-step distribution.
6. If cluster not found after $t_{\text{max}}$ iterations, terminate and return $\emptyset$.

We can deduce some results from this description. One first remark we can make on $r^t$ is the following:

$$
\lim_{t \to \infty} r^t(x) = \lim_{t \to \infty} \frac{p^t_{x_0}(x)}{d(x)} = \frac{\pi(x)}{d(x)} = \frac{d(x)}{\sum_{y \in V} d(y)} \cdot \frac{1}{d(x)} = \frac{1}{\mu(V)}. \quad (6.5)
$$

independent of $x$ and starting vertex $x_0$. Hence for all $x \in V$, $r^t(x)$ converges to a constant, provided that the graph is connected (see Subsection 2.1). Vertices that are easily accessible from $x_0$ are likely to have a larger value of $r^t$, whereas relatively isolated vertices will most probably have small $r^t$. These differences even out over time, also because the probabilities have to sum up to 1.

Another result presented in Spielman-Teng [ST08] is the following.

**Proposition 6.1.** Let $q \in \mathbb{R}_+^n$ and $D$ the diagonal matrix with degrees of the vertices. Then

$$
\|D^{-1} Mp\|_\infty \leq \|D^{-1} p\|_\infty. \quad (6.6)
$$

**Proof.** With $z = D^{-1} p,$

$$
D^{-1} Mp = D^{-1} MDz = D^{-1} \left( \frac{1}{2} (I + AD^{-1}) \right) Dz = \frac{1}{2} D^{-1} (A + D)z - \frac{1}{2} (D^{-1}A + I)z = M^T z. \quad (6.7)
$$

Then,

$$
\|D^{-1} Mp\|_\infty = \|M^T z\|_\infty \leq \|M^T\|_\infty \|z\|_\infty. \quad (6.8)
$$

The last inequality hold because $\| \cdot \|_\infty$ is an induced matrix norm, see [Mat07]. Note that $M$ is a transition probability matrix, so that each column of $M$ sums up to 1. The infinite matrix norm is given by the maximum row sum of the matrix, so that

$$
\|D^{-1} Mp\|_\infty \leq 1 \cdot \|z\|_\infty = \|D^{-1} p\|_\infty. \quad (6.9)
$$

If we choose $p = p^t_{x_0} = M^t e_{x_0}$, $(6.6)$ translates into

$$
\max_{x \in V} \frac{p^t_{x_0}(x)}{d(x)} \leq \max_{x \in V} \frac{p^t_{x_0}(x)}{d(x)} \quad \text{or} \quad \max_{x \in V} r^{t+1}(x) \leq \max_{x \in V} r^t(x). \quad (6.10)
$$

This also confirms the converging behaviour of $r^t$. For clarity, we omit the subscript in $p^t_{x_0}$ if the starting vertex is fixed.

### 6.1.2 Monotonicity

In Section 5.2.1 we have established a monotonicity property for the Spielman-Teng algorithm on a two-dimensional grid. More precisely, for the Spielman-Teng algorithm, if we construct two clusters with conductance threshold $\varphi_1$ and $\varphi_2$ respectively, with $\varphi_1 > \varphi_2$ as stopping criterion, the cluster constructed with $\varphi_1$ is a subset of the cluster constructed with $\varphi_2$, under the assumption that the same starting vertex is used. This follows from the fact that in this particular graph, in each iteration the vertices are included in the same order. Hence, the
6.1. PROPERTIES OF SPIELMAN-TENG

Figure 6.1: Realisation of the Poisson point process with $\lambda = 1.6$ and connectivity distance $R_{\text{com}} = 1.2$. (a), (b), (c) and (d) represent the clusters produced by the ST algorithm with conductance threshold $\phi = 1, 0.2, 0.1, 0.075$, respectively.

cluster constructed with conductance $\phi_2$ is equal to the cluster constructed with threshold $\phi_1$ plus some additional vertices. We could say that the formed cluster is a ‘monotone’ function of the conductance. We shall now investigate whether this monotonicity property extends to a larger class of graphs and whether it perhaps even holds in much generality. As a first test case, we construct graphs by sampling points in a square of 10 by 10 according to a Poisson point process with $\lambda = 1.6$ and applied a connectivity distance $R_{\text{com}} = 1.2$. One outcome is depicted in Figure 6.1. We selected the vertex coloured red in Figure 6.1(a) as the starting vertex. Repeating the algorithm for conductance thresholds 0.2, 0.1 and 0.075 yields the clusters $C_1, C_2, C_3$ depicted in Figure 6.1(b)-(d) respectively. Indeed, we observe that $C_1 \subseteq C_2 \subseteq C_3$. By repeating this experiment for other samples, we find that this monotonicity seems to hold in general. This leads us to formulate the following conjecture.

**Conjecture 6.1.** Let $\text{Nibble}(G, x, \phi, b)$ be the outcome of the ST algorithm applied to graph $G$ with starting vertex $x$, conductance threshold $\phi$ and parameter $b$ (as in Algorithm 8. Let $\phi_1, \phi_2 \in (0, 1]$, $\phi_1 > \phi_2$ and $C_1 = \text{Nibble}(G, x, \phi_1, b)$, $C_2 = \text{Nibble}(G, x, \phi_2, b)$. If $C_2 \neq \emptyset$, then

$$C_1 \subseteq C_2.$$  

(6.11)

The difficulty of proving Conjecture 6.1 depends strongly on which of the following assumptions we make.
1. $C_1$ and $C_2$ are found in the same iteration $t$,
2. $C_1$ is found in iteration $t_1$, while $C_2$ is found in iteration $t_2$ with $t_1 < t_2$.

Case 1 is easy to prove. Namely, let $C_1 = S^I_{j_1}$ and $C_2 = S^I_{j_2}$ for some $j_1, j_2 \in \{1, ..., |S|^l\}$. Then it clearly needs to hold that $j_1 \leq j_2$, because otherwise

$$\phi(S^I_{j_2}) \leq \phi_2 < \phi_1,$$

and the first run would have terminated at this point. Hence

$$C_1 = \{\Pi^I(1), ..., \Pi^I(j_1)\} \subseteq \{\Pi^I(1), ..., \Pi^I(j_2)\} = C_2.$$  

(6.13)

Case 2 needs a lot more thought. When the difference between $\phi_1$ and $\phi_2$ is relatively large, this is the most occurring case. This is because the support set iteration $t_1$ of is simply not large enough to accommodate a cluster of conductance $\phi_2$. Assume without loss of generality that $t_1 = t$ and $t_2 = t + 1$ and let

$$C_1 = S^I_{j_1} = \{\Pi^I(1), ..., \Pi^I(j_1)\},$$

(6.14)

$$C_2 = S^{I+1}_{j_2} = \{\Pi^{I+1}(1), ..., \Pi^{I+1}(j_2)\}.$$  

(6.15)

Conjecture 6.1 is equivalent with $S^I_{j_1} \subseteq S^{I+1}_{j_2}$. Showing this is not trivial like in case 1, since the permutations $\Pi^I$ and $\Pi^{I+1}$ might differ in the first $|S^I|$ elements. Another reformulation of Conjecture 6.1 is:

$$\phi(\{\Pi^I(1), ..., \Pi^I(j_1)\}) \leq \phi_1,$$

then

$$\phi(\{\Pi^{I+1}(1), ..., \Pi^{I+1}(k)\}) > \phi_1$$

for all $k$ such that $\{\Pi^I(1), ..., \Pi^I(j_1)\} \not\subseteq \{\Pi^{I+1}(1), ..., \Pi^{I+1}(k)\}$.

Some observations that may help in the proof are given below.

**Lemma 6.1.** If a vertex is $t + 1$ steps away from the starting node, in the $t + 1^{th}$ iteration, at least one of its neighbours at $t$ hops from the starting vertex is added before the new vertex is added. I.e.

$$r^{t+1}(y) \leq \max_{x \sim y} r^{t+1}(x).$$

(6.18)

**Proof.** Because the random walk is lazy, we find that for all $x \in S^I$

$$p^{t+1}(x) \geq \frac{1}{2} p^t(x) > 0,$$

(6.19)

which implies $x \in S^{t+1}$ and hence $S^I \subseteq S^{t+1}$.

Let $y \in S^{t+1} \backslash S^I$, hence $p^I(y) = 0$ and $p^{t+1}(y) > 0$.

$$p^{t+1}(y) = \frac{1}{2} p^I(y) + \sum_{x \sim y} p^t(x) p(x, y),$$  

(6.20)

$$= \sum_{x \sim y} p^t(x) \cdot \frac{1}{2d(x)},$$  

(6.21)

$$= \frac{1}{2} \sum_{x \sim y} r^t(x).$$  

(6.22)
6.1. PROPERTIES OF SPIELMAN-TENG

Clearly, \(x \in S^t \setminus S^{t-1}\) for all \(x \sim y\) such that \(p^t(x) > 0\), since \(S^t\) contains all vertices within \(t\) hops of the starting vertex. Reformulating \((6.20)\) gives

\[
p^t(x) = 2p^{t+1}(x) - 2 \sum_{z \sim x} p^t(z)p(z,x) \leq 2p^{t+1}(x). \tag{6.23}\]

Combining \((6.21)\) and \((6.23)\) yields

\[
p^{t+1}(y) \leq \sum_{x \sim y} \frac{p^{t+1}(x)}{d(x)} \leq d(y) \max_{x \sim y} \frac{p^{t+1}(x)}{d(x)}. \tag{6.24}\]

This yields \((6.18)\).

The statement that \(r^{t+1}(y) \leq r^{t+1}(x)\) for all \(x \sim y\) is in general not true. This also implies that in general

\[
S^{t+1}_{|S_t|} \neq S^t, \tag{6.25}\]

That is, the first \(|S_t|\) vertices of the ordered support set of the \((t+1)\)-step distribution not necessarily coincides with with the support set of the \(t\)-step distribution. See Appendix A for some examples. Hence Lemma 6.1 does not provide sufficient ground for proving the conjecture, because in the \((t+1)\)th iteration, one of the ‘new’ vertices may be added to the cluster before all vertices in the previous support set are included. Therefore, a cluster of lower conductance may be found before all previous vertices (and consequently the vertices in \(C_1\)) are included. However, the latter does not occur, and deeper research on the reason needs to be done, possibly leading to the complete proof of Conjecture 6.1.

6.1.3 Connectivity

Another observation that can be made on the outcomes of algorithm applied to the grid network is the following.

**Conjecture 6.2.** For all \(x \in V, \varphi \in (0, 1], \) the cluster \(C(x, \varphi)\) is connected.

For the grid we have shown this Subsection 5.1, where we derived the explicit expression of the \(t\)-step distribution. To prove Conjecture 6.2 it suffices to show that

\[
\exists x \sim y : r^t(x) \geq r^t(y) \quad \forall y \in S^t \setminus \{\Pi^t(1)\}. \tag{6.26}\]

In words, this statement says that for a given vertex \(x\), at least one of its neighbours is added to the set before the vertex itself is added (except for the first vertex). Equivalently, we have to prove

\[
r^t(y) \leq \max_{x \sim y} r^t(x) \quad \forall y \in S^t \setminus \{\Pi^t(1)\}. \tag{6.27}\]

Although we are unable to prove the general statement, we can give the proof for a special type of graph, namely a \(k\)-regular tree.

**Proposition 6.2.** Let \(G\) be a \(k\)-regular tree (up to level \(T > t\)) with root vertex \(x_0\). Then for all \(t \in \mathbb{N}\)

\[
r^t(x) \leq r^t(p_x) \quad \forall x \in V \setminus \{x_0\}, \tag{6.28}\]

where \(p_x\) is the parent of \(x\) and

\[
r^t(x_0) = \max_{x \in V} r^t(x). \tag{6.29}\]
Proof. Since we are dealing with a $k$-regular graph, $r^t(x) \leq r^t(p_x)$ is equivalent to $p^t(x) \leq p^t(p_x)$. We prove the statement by induction.

For $t = 0$, we have
\[
p^0(x) = \begin{cases} 1, & \text{if } x = x_0, \\ 0, & \text{else}. \end{cases} \tag{6.30}
\]

Hence,
\[
r^0(x) = \begin{cases} 1/k, & \text{if } x = x_0, \\ 0, & \text{else}. \end{cases} \tag{6.31}
\]
Clearly, $p^0(x_0) = \max_{x \in V} p^0(x)$ and for all $x \sim x_0$
\[
0 = p^0(x) \leq p^0(x_0) = 1, \tag{6.32}
\]
while
\[
0 = p^0(x) \leq p^0(p_x) = 0, \tag{6.33}
\]
for $x \sim x_0$, provides the basis for our hypothesis. Now suppose indeed $p^{t-1}(x) \leq p^{t-1}(p_x)$ for all $x \in V$ with parent $p_x$. Then if $p_x \neq x_0$,
\[
p^t(p_x) = \frac{1}{2}p^{t-1}(p_x) + \frac{1}{2k}p^{t-1}(p_{p_x}) + \frac{1}{2k} \sum_{c \text{ child of } p_x} p^{t-1}(c), \tag{6.34}
\]
where the last equality follows from symmetry.
\[
p^t(x) = \frac{1}{2}p^{t-1}(x) + \frac{1}{2k}p^{t-1}(p_x) + \frac{1}{2k} \sum_{c \text{ child of } x} p^{t-1}(c), \tag{6.36}
\]
\[
= \frac{1}{2}p^{t-1}(x) + \frac{1}{2k}p^{t-1}(p_x) + \frac{k-1}{2k}p^{t-1}(c_x), \tag{6.37}
\]
where $c_x$ is a child of $x$. Then,
\[
p^t(p_x) - p^t(x) = \frac{1}{2} \left[ p^{t-1}(p_x) - p^{t-1}(x) \right] + \frac{1}{2k} \left[ p^{t-1}(p_{p_x}) - p^{t-1}(p_x) \right] \tag{6.38}
\]
\[+ \frac{k-1}{2k} \left[ p^{t-1}(x) - p^{t-1}(c_x) \right]. \tag{6.39}
\]
Here, all three terms between the brackets are positive by the induction assumption. Hence $p^t(p_x) - p^t(x) \geq 0$, yielding the required statement. For $x_0$ and $x \sim x_0$, we find
\[
p^t(x_0) - p^t(x) = \left[ \frac{1}{2}p^{t-1}(x_0) + \frac{1}{2}p^{t-1}(x) \right] - \left[ \frac{1}{2}p^{t-1}(x) + \frac{1}{2k}p^{t-1}(x_0) + \frac{k-1}{2k}p^{t-1}(c_x) \right] \tag{6.40}
\]
\[= \frac{1}{2} \left[ p^{t-1}(x_0) - p^{t-1}(x) \right] + \frac{1}{2k} \left[ p^{t-1}(x) - \frac{1}{k}p^{t-1}(x_0) - \frac{k-1}{k}p^{t-1}(c_x) \right], \tag{6.41}
\]
\[\geq \frac{1}{2} \left[ p^{t-1}(x_0) - p^{t-1}(x) \right] + \frac{1}{2} \left[ p^{t-1}(x) - p^{t-1}(x_0) \right] = 0. \tag{6.42}
\]
This gives
\[
p^t(x_0) \geq p^t(x), \tag{6.43}
\]
for all $x \sim x_0$, and hence $p^t(x_0) = \max_{x \in V} p^t(x)$. \qed
6.2 Properties of Andersen-Peres

In this special case, the order of inclusion of the vertices in the support set does not change as \( t \) increases. This is in general not the case. Note that the \( t \)-step distribution on each of the vertices of one level is equal by symmetry. By Proposition 6.2, we find that monotonicity as in Conjecture 6.1 indeed holds for \( d \)-regular tree, due to the fixed ordering in the \( t \)-step distribution for all \( t \). This means that in each iteration of the Spielman-Teng algorithm, first the root vertex is included, then its neighbours, then the vertices at level 2, etcetera, until level \( t \).

6.2 Properties of Andersen-Peres

As discussed in Chapter 4, the algorithm by Andersen and Peres simulates the volume-biased evolving set process. The process can be seen as a Markov chain on the subsets of the vertices. Vertices are included (or excluded) according to the degree in which they are connected to the current set. We are able to do some observations on this algorithm.

6.2.1 Randomness

Since the Andersen-Peres algorithm implements a random process, the outcome of each execution is not deterministically defined. Hence different runs might produce different clusters. This is illustrated in Figure 6.2, where we ran three times the AP algorithm on the graph we also used in Subsection 6.2.1 and the same starting vertex. Although all three clusters have conductance smaller than 0.1, they are clearly different. Also note that none of these clusters coincides with the cluster constructed by Nibble with \( \phi = 0.1 \) in Figure 6.1(c).

Not surprisingly, the monotonicity in subsets that occurs in the Spielman-Teng algorithm does not hold in the Andersen-Peres algorithm, due to the probabilistic nature of the algorithm. In contrast to Nibble, we cannot guarantee that a cluster with lower conductance threshold also results in a cluster containing more vertices.

6.2.2 Inclusion of streets

Figure 6.2 provides some evidence that the AP algorithm does not easily include vertices that have weak connections to the current set. Furthermore, we already argued that the likelihood of AP including long ‘streets’ is small. In this subsection, we will show that this is indeed the case by establishing the order of the lower bound on the hitting time of including a line of length \( n \) into the constructed cluster. The situation we discuss is depicted in Figure 6.3.

We assume that vertex 0 is never excluded from the set, so that the connection to the line remains intact. The hitting times we find here will hence be a lower bound for the actual hitting times without this condition. Let vertex 0 and \( n \) have degree \( d(0) \) and \( d(n) \) respectively. We focus on the original ESP, described in Section 2.3, first and derive results for the volume-biased version, described in Section 2.4, afterwards.

Proposition 6.3. The expected number of iterations needed by the ESP to include a full street of \( n \) vertices is at least \( O(n^2) \).

Proof. Since we are not interested in what happens on the rest of the boundary of \( S_0 \) during the execution of the algorithm, we analyse the path of the Markov chain on classes of sets, defined as:

\[
S_k^* = \{ S \subseteq V | S_0 \cup \{1, ..., k\} \subseteq S, j \notin S \forall j = k + 1, ..., n \}.
\] (6.44)
CHAPTER 6. GENERAL OBSERVATIONS ON LOCAL ALGORITHMS

Figure 6.2: Realisation of the Poisson point process with \( \lambda = 1.6 \), and connectivity distance \( R_{\text{com}} = 1.2 \). Three outcomes of the AP algorithm with conductance threshold \( \varphi = 0.1 \).

Figure 6.3: Starting set \( S_0 \) with street of length \( n \) connected to it.
In this way, we can ignore the transitions made on the boundary of $S_0$ other than the line. Clearly,

\[ p(0, S_0) = \frac{1}{2} + \frac{1}{2} \cdot \frac{d(0) - 1}{d(0)} = \frac{2d(0) - 1}{2d(0)}, \]

\[ p(1, S_0) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}. \]

Hence, let $S'$ be the next set and $P_i$ denote the probability measure conditioned on the event that the current set is in $S^*_i$.

\[ P_0(S' \in S_0 \setminus \{0\}) = P(U \leq \frac{2d(0) - 1}{2d(0)}) = \frac{2d(0) - 1}{2d(0)}. \]

\[ P_0(S' \in S^*_0) = \frac{3d(0) - 2}{4d(0) - 2}, \]

\[ P_0(S' \in S^*_1) = \frac{d(0)}{4d(0) - 2}. \]

For $1 < i < n - 1$, we find

\[ p(i, S^*_i) = \frac{1}{2} + \frac{1}{2} \cdot \frac{2}{2} = \frac{3}{4}, \]

\[ p(i + 1, S^*_i) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}. \]

This yields

\[ P_i(S' \in S^*_{i-1}) = 1 - \frac{3}{4} = \frac{1}{4}, \]

\[ P_i(S' \in S^*_i) = \frac{3}{4} - \frac{1}{4} = \frac{1}{2}, \]

\[ P_i(S' \in S^*_i+1) = \frac{1}{4}. \]

Last, for $i = n - 1$, we have

\[ p(n - 1, S^*_n - 1) = \frac{1}{2} + \frac{1}{2} \cdot \frac{2}{2} = \frac{3}{4}, \]

\[ p(n, S^*_n) = \frac{1}{2} \cdot d(n), \]

which gives

\[ P_{n-1}(S' \in S^*_n - 2) = 1 - \frac{3}{4} = \frac{1}{4}, \]

\[ P_{n-1}(S' \in S^*_n - 1) = \frac{3}{4} - \frac{1}{2d(n)} = \frac{3d(n) - 2}{4d(n)}, \]

\[ P_{n-1}(S' \in S^*_n) = \frac{1}{2d(n)}. \]

The full transition diagram is depicted in Figure 6.4. This diagram allows us to derive the
expected number of iterations needed to include the entire street of $n$ hops. That is, the number of iterations to reach a set in $S_n^*$ starting in $S_0^*$. Let $ET_{i,j}$ denote the expected time to reach $S^*_i$ starting in $S^*_j$. The requested hitting time is $ET_{0,n}$. The following equality holds:

$$ET_{0,n} = \sum_{i=0}^{n-1} ET_{i,i+1}. \quad (6.46)$$

For $i = 1, ..., n - 2$ we find

$$ET_{i,i+1} = 1 + \frac{1}{2} ET_{i,i+1} + \frac{1}{4} (ET_{i-1,i} + ET_{i,i+1}), \quad (6.47)$$

$$ET_{i,i+1} = 4 + ET_{i-1,i} = 4i + ET_{0,1}. \quad (6.48)$$

Furthermore,

$$ET_{0,1} = 1 + \frac{3d(1) - 2}{4d(1) - 2} ET_{0,1},$$

$$ET_{0,1} = \frac{4d(1) - 2}{d(1)}$$

and

$$ET_{n-1,n} = 1 + \frac{3d(n) - 2}{4d(n)},$$

$$ET_{n-1,n} = \frac{4d(n)}{d(n) + 2}.$$

By Equation (6.46) it follows that

$$ET_{0,n} = ET_{0,1} \sum_{i=1}^{n-2} ET_{i,i+1} + ET_{n-1,n}$$

$$= ET_{0,1} + 4 \sum_{i=1}^{n-2} i + (n - 2)ET_{0,1} + ET_{n-1,n}$$

$$= (n - 1)ET_{0,1} + 2(n - 2)(n - 1) + ET_{n-1,n}$$

$$= (n - 1) \frac{4d(1) - 2}{d(1)} + 2(n - 2)(n - 1) + \frac{4d(n)}{d(n) + 2} = O(n^2)$$

Note that we considered the unbiased ESP in this calculation, while the Andersen-Peres algorithm implements the volume-biased version. Luckily, Andersen and Peres provided a useful result in their paper that allows the extension of the statement of Proposition 6.1 to the volume-biased ESP.
Proposition 6.4 ([AP08] p. 5). For any function $f$ and starting set $S_0 \neq \emptyset$,
\[
\hat{E}[f(S_0, \ldots, S_t)] = \mathbb{E} \left[ \frac{\mu(S_t)}{\mu(S_0)} f(S_0, \ldots, S_t) \right],
\]
(6.49)
where $\hat{E}[-]$ denotes the expectation under the volume-biased transition kernel.

This gives us the following corollary.

Corollary 6.1. The expected number of iterations needed by the volume-biased ESP to include a full street of $n$ vertices is at least $O(n^2)$.

Proof. If we choose
\[
f(S_0, \ldots, S_t) = \min\{k | S_k \in S_n\} \left( = \frac{\mathcal{T}_{0,n}}{\mu(S_t)} \right),
\]
(6.50)
we get
\[
\hat{E}\mathcal{T}_{0,n} \geq \hat{E} \left[ \frac{\mathcal{T}_{0,n}}{\mu(S^1_{0,n})} \right] = \mathbb{E} \left[ \frac{\mu(S^1_{0,n})}{\mu(S_0)} \cdot \frac{\mathcal{T}_{0,n}}{\mu(S^1_{0,n})} \right] = \frac{1}{\mu(S_0)} \mathbb{E}\mathcal{T}_{0,n} = O(n^2).
\]
(6.51)

Hence, in order to cross a street of length $n$, it takes at least $O(n^2)$ number of iterations to succeed. This quadratic order shows that it is fairly difficult for such long roads to be included in the cluster constructed by the Andersen-Peres algorithm.

6.3 Adding constraints

The main motivation for using local clustering algorithms instead of global algorithms, like spectral clustering, was the fact that the local algorithms allow for incorporation of constraints. In this section, we will show how to do so.

6.3.1 Restricting construction

In the algorithm of Spielman-Teng, a cluster is constructed vertex by vertex by analysis of the $t$-step distribution of the random walk defined on the graph. This step-wise procedure clearly allows us to influence the way in which the cluster is formed. Remember from Algorithm 10 that the support set $S^t$ of the (truncated) $t$-step distribution defines the order of construction in each iteration step. Starting from the empty set, vertices are added to the cluster one by one. After each addition, the intermediary subsets of $S^t$ are checked for conditions posed in Algorithm 10. If all of these conditions are met, the procedure terminates and the constructed subset of vertices is returned. However, instead of just checking the conditions posed by Spielman-Teng, we can easily include additional desired constraints in this step, causing the algorithm to only return clusters which are feasible according to some constraints defined by ourselves.

A similar modification can be made to the Andersen-Peres algorithm. Remember that this algorithm essentially simulates the volume-biased evolving set process, which could be represented by a Markov chain on all subsets of the vertices. After each transition, the evolving set is checked for conductance, and the process is terminated once a set of sufficiently low conductance is found. Again, we note that there is a point in the process in which we can evaluate the current solution. In this case, this can be done when the transition to a new subset has been made. If the new set does not meet all the constraints we defined, we simply prohibit the transition and move on to the next iteration.


6.3.2 Possible constraints

Size

Probably the first constraint that comes to mind when thinking of a good quality cluster is its size. Naturally, small clusters will perform well in network simulations, because the offered load is typically low. On the other hand, producing small clusters will result in a large number of segments and thereby a large number of the expensive segment controllers. Hence, it is likely that there is a minimum value for the cluster sizes we would like to obtain. Of course, large clusters tend to exhibit high offered load and high loss rates. Introducing a constraint that sets an upper bound for the size solves this issue. At each step of the ST and AP algorithm, the constraint can be checked.

Hop count bounding

By intuition we can tell that the number of nodes a particular packet has to be routed through is strongly correlated with the expected delay of the packet and, to a lesser extend, also the failure probability. This last claim may be justified by observing that the more intermediary transmissions a packet needs to take, the higher the probability that one of these transmissions fails, will be. Therefore, we might want to pose a constraint on the number of hops between the segment controller and each of its assigned OLCs. This may be done by fixing a upper limit on the average hop count within the segment. Another option is to bound the hop count of all vertices in the cluster individually.

However, the position of the segment controller is not known during the execution of the algorithm. Without redefining the segment controller position in each iteration, calculating the distances between vertices and the segment controller is hence not possible. This is issue can be solved in the following way. As we argued before, the medoid of a cluster would be a reasonable choice as the segment controller position. Hence, recalculating the medoid of the cluster in each iteration is an option. However, we also showed that computing the medoid of a set of vertices of size \( n \) takes \( O(n^3) \) calculations. Since this procedure needs to be done in each iteration, this will be a very time-consuming task. We are able to do this somewhat more efficiently by computing and storing the distance matrix once before the execution of the ESP, we are able to save time. Also, we do not need to store the matrix of the entire graph, but just the vertices that are within a certain range of the starting vertex.

6.4 Stopping criteria

Both the Spielman-Teng and Andersen-Peres algorithm use a stopping criterion based on conductance. The procedure is terminated once a cluster of sufficiently low conductance is found. Also, both algorithms set a maximum number of iterations, both to limit the complexity of the algorithm and to terminate when it becomes clear that the starting vertex is not contained in a subset of sufficiently low conductance. However, we might ask ourselves whether asking for a cluster of certain conductance is desired in our setting. Indeed, we would like to construct clusters that contain many internal edges. Also, the number of internal edges usually correlates with the size of the constructed cluster. But setting a conductance constraint does not guarantee that a cluster of a certain size is found.

Since we argued that we like the clusters of the partition to be similar in size, we might want to make this our primary stopping criterion. Note that both local clustering algorithms search for low conductance clusters by design. We may therefore assume that in each iteration
of the algorithms, a good quality cluster is found, in terms of conductance. Therefore, we choose to replace the conductance criterion by a minimum size criterion. That is, the cluster is constructed in the same way as in the original setting, but the procedure is terminated once the size of the cluster exceeds a certain value, say \( m^* \). The constraint on the maximum number of iterations is kept in order to limit the running time of the algorithm and to stop the search if it is likely that a good quality cluster cannot be constructed using the chosen vertex as the starting point.

6.5 Including spatial information

Until now, we only used the undirected, unweighted connectivity matrix in the local algorithms. The two algorithms as described in [ST08] and [AP08] in fact only consider the partitioning of unweighted graphs. Nevertheless, we found a way to incorporate weights on edges in both algorithms. This will be discussed in Subsection 6.4.1. By introducing the possibility to define a measure on links between vertices, we are able to include the valuable information on the geographical structure of the graph. Subsection 6.4.2 studies how to choose the weights on the edges of the graph while accounting for distances between the vertices.

6.5.1 Introducing weights

We follow the terminology of Chapter 2, and denote the weight on edge \((i, j)\) by \( w(i, j) \). The matrix \( W \) stores all values of \( w(i, j) \). The total weight of a vertex is given by \( w(i) = \sum_i w(i, j) \) and the diagonal matrix \( D_w \) is defined by \((D_w)_{ii} = w(i)\). The natural extension of the lazy random walk considered before, is the biased lazy random walk discussed in Subsection 2.1.2 with transition matrix

\[
M_w = \frac{1}{2} \left( I + WD_w^{-1} \right).
\]  

(6.52)

The transition probabilities hence become

\[
p_w(x, y) = \begin{cases} 
\frac{1}{2w(x)}, & \text{if } x = y, \\
\frac{w(x, y)}{2w(x)}, & \text{if } x \sim y \\
0, & \text{else.}
\end{cases}
\]

(6.53)

This choice results in a preference of the random walk to pass along an edge with relatively high weight. This adaption forms the basis of the modifications we make to the local algorithms by Spielman-Teng and Andersen-Peres.

Spielman-Teng

In order to make the Spielman-Teng algorithm suitable for finding clusters in weighted graphs, we need to make two adaptions. The first adaption is to replace the unbiased transition matrix \( M \) by \( M_G \) defined in (6.52). This implies that the \( t \)-step distribution of the biased lazy random walk will be used to determine the order of inclusion. The second modification is in the calculation of \( r^t \). Since the degree \( d(x) \) of a vertex is not relevant in the biased version, we take its equivalent, namely \( w(x) \). Hence,

\[
p_{x_0}^{t+1} = M_w p_{x_0}^t = \{p_1^{t+1}, ..., p_n^{t+1}\}
\]

(6.54)

and

\[
r^t(x) = \frac{p^t(x)}{w(x)}.
\]

(6.55)
The truncation operation becomes
\[
[p]_\varepsilon(x) = \begin{cases} 
  p(x), & \text{if } p(x) \geq w(x)\varepsilon, \\
  0, & \text{else}.
\end{cases}
\] (6.56)

Since we already omitted the constraints posed on the generated cluster in the previous section, where we stated one single stopping criterion based on the size, we do not need to adjust those to the weighted case. The full description of the modified Spielman-Teng algorithm (with size threshold) is given in Algorithm 10.

**Algorithm 10**: Modified version of Spielman and Teng’s algorithm called Nibble

**Input**: Graph \( G = (V, E) \), starting vertex \( x \in V \), size threshold \( m^* \in \mathbb{N}_+ \), maximum number of iterations \( T \in \mathbb{N}_+ \), truncation threshold \( \varepsilon > 0 \).

**Output**: \( S \subseteq V \), such that either \( |S| \geq m^* \) or \( S = \emptyset \).

Set \( p_0 = \chi_v \), \( q_0 = [p_0]_\varepsilon \);
Set \( M_W = \frac{1}{2}(I + W_GD_w^{-1}) \);

for \( t = 1, \ldots, T \) do

1. Set \( p_t = M_W q_{t-1} \);
2. Set \( q_t = [p_t]_\varepsilon \);
3. Let \( S'_t \) be the support set of \( q_t \). If \( |S'_t| \geq m^* \), then return

\[ S = S_{m^*}(q_t) = \{\Pi_t(1), \ldots, \Pi_t(m^*)\}, \]
and terminate.
4. If \( t = T \), return \( S = \emptyset \).

**Andersen-Peres**

In the original Andersen-Peres algorithm, the probability of a vertex being included (or excluded) in the next set in the Markov chain, was determined by the probability of the random walk entering (or staying inside) the current set. However, because the transition probabilities of the random walk are modified, we also need to adjust the entering probability as originally defined in (2.55) according. Define for \( y \in V \),
\[
E_W(y, S) = \sum_{x \in S, x \sim y} w(x, y)
\] (6.57)

and
\[
p_W(y, S) = \frac{1}{2} \left( 1_{(y \in S)} + \frac{E_W(y, S)}{w(y)} \right). \tag{6.58}
\]

The set \( S' \) following the current set \( S \) hence equals
\[
S' = \{y \in V | p_W(y, S) \geq U\}, \tag{6.59}
\]
where $U$ is uniformly distributed on $[0,1]$, just as in the original case. It follows that the observations we made on original algorithm still hold. Namely, for $y \in S$

$$p_W(y,S) = \frac{1}{2} \left( 1 + \frac{E_W(y,S)}{w(y)} \right) \geq \frac{1}{2},$$

(6.60)

and for $y \in S^c$,

$$p_W(y,S) = \frac{1}{2} E_W(y,S) \leq \frac{1}{2}.$$  

(6.61)

This implies that the current set does not grow if $U \geq 1/2$ and does not shrink if $U < 1/2$. Furthermore, if $y \in S \setminus \partial S = \text{int } S$ we find

$$p_W(y,S) = \frac{1}{2} \left( 1 + \sum_{x \sim y} \frac{w(x,y)}{w(y)} \right) = \frac{1}{2} (1 + 1) = 1,$$

(6.62)

which shows that $\text{int } S \subseteq S'$ for all outcomes of $U$. These observations imply that the overall structure of the algorithm stays intact when incorporating weights. Our own weighted version of AP with alternative stopping criterion is fully described in Algorithm 11.

| **Input:** Graph $G = (V, E)$, initial vertex $x \in V$, $T \in \mathbb{N}_+$, size threshold $m^* \in \mathbb{N}_+$. |
| **Output:** $S \subseteq V$, sampled from the weighted volume-biased ESP. |
| Set $S = S_0 = \{x\}$, $X = x_0 = x$; |
| **for** $t = 1, ..., T$ **do** |
| 1. Given $X_{t-1} = x_{t-1}$, select $X_t = x_t$ with probability $p_W(x_{t-1},x_t)$ and set $X = x_t$; |
| 2. Pick $U$ uniformly at random from $[0, p_W(x_t,S_{t-1})]$; |
| 3. Define $S_t = \{y \mid p_W(y,S_{t-1}) \geq Z\}$; |
| 4. If $t = T$ or $|S_t| \geq m^*$, set $S = S_t$ and terminate; |
| **end** |

Algorithm 11: Modified version of Andersen and Peres’ algorithm called GenerateSample.

### 6.5.2 Weight function

As the local algorithms allow extension to weighted graphs, we can use this possibility to include spatial information in the construction of the clusters. What remains to be seen is how to represent the geographical information by the weights. Of course, the free space distance between two vertices plays an important role in this. The weight function $w : E \mapsto \mathbb{R}$ should hence be based on this distance. Furthermore, it should have the following properties:

1. $w(i,j) \geq 0$ for all $v_i, v_j \in V$.

2. $w(i,j) = 0$ for all pairs of vertices that are not within communication range.

3. The function $w$ is decreasing in the distance between the Euclidean distance between the two vertices incident to the edge.
CHAPTER 6. GENERAL OBSERVATIONS ON LOCAL ALGORITHMS

Requirement one is necessary to avoid division by zero (if the total weight of a vertex sums up to zero). The second requirement avoids that the local algorithm produces a disconnected cluster. The third requirement is desired, since we want the weight of an edge to represent the quality of the connection between the vertices. It does not make sense to assign large weight, that is a good link quality, to distant vertices.

We propose some functions that fit the requirements. A first option is a linearly decreasing function in terms of distance:

\[
\begin{align*}
    w_{\text{lin}}(i,j) &= \begin{cases} 
        \frac{-\|v_i - v_j\|}{R_c} + 1, & \text{if } i \neq j \text{ and } \|v_i - v_j\| \leq R_c, \\
        0, & \text{else},
    \end{cases} 
\end{align*}
\]

where \( R_c \) is the communication range and \( \| \cdot \| \) the Euclidean distance function. For the next option, we refer back to Section 3.2, in which the affinity of two vertices was defined. Remember that the affinity between vertices \( v_i \) and \( v_j \) was defined as

\[
\begin{align*}
    w_{\text{NJW}}(i,j) &= \begin{cases} 
        \exp\left(-\frac{\|v_i - v_j\|^2}{\sigma^2}\right), & \text{if } i \neq j, \\
        0, & \text{else},
    \end{cases} 
\end{align*}
\]

where \( \sigma > 0 \) is the scaling parameter. This affinity function given by Ng et al. [NJW01] in fact satisfies the first and third requirement stated above. Nevertheless, we can slightly adjust it in order to also satisfy the second requirement, namely

\[
\begin{align*}
    w_{\text{NJW}}(i,j) &= \begin{cases} 
        \exp\left(-\frac{\|v_i - v_j\|^2}{\sigma^2}\right), & \text{if } i \neq j \text{ and } \|v_i - v_j\| \leq R_c, \\
        0, & \text{else},
    \end{cases} 
\end{align*}
\]

A third weight function also comes from Subsection 3.2. The locally scaled affinity function defined as

\[
\begin{align*}
    w_{\text{loc}}(i,j) &= \begin{cases} 
        \exp\left(-\frac{\|v_i - v_j\|^2}{\sigma_i \sigma_j}\right), & \text{if } i \neq j \text{ and } \|v_i - v_j\| \leq R_c, \\
        0, & \text{else},
    \end{cases} 
\end{align*}
\]

where \( \sigma_i = \|v_i - v_i(K)\| \) and \( v_i(K) \) the \( K \)th closest neighbour of \( v_i \). For a more elaborate description of this function we refer to Subsection 3.2.2.

In Section 3.2, we saw that the local scaling affinity matrix very well accounted for homogeneities in the graph. Since we are typically dealing with graphs in which the vertices are inhomogeneously dispersed, this last weight function seems a good choice to apply in the weighted versions of the two local algorithms. Hence, we will use this function in the comparison of segmentations in Chapter 9.
Chapter 7

Auxiliary routines

7.1 Framework

This chapter completes the partitioning algorithms built around the cluster generating algorithms studied in Chapter 4, 5 and 6. Remember from Section 4.2 that the general scheme for the clustering is as follows:

1. Seed node selection
2. Cluster generation
3. Cluster selection
4. Covering to partition

We extensively studied the second step and modified the algorithms by Spielman-Teng and Andersen-Peres to fit our application in the previous two chapters. Although this was the most important step in the total procedure, the generated clusters alone do not directly provide a partition. In the last three sections of this chapter, we elaborate on each of the other three subroutines, but first we discuss the method Spielman-Teng proposed in [ST08] to partition a graph into low-conductance subgraphs with the help of their cluster generating algorithm. Because this method turns out to be unsuited when applied to our connectivity graphs, we designed three subroutines of our own, which will be described in Section 7.3 to 7.5.

7.2 First approach

In [ST08] Spielman and Teng introduce a complete method to partition a graph into subgraphs of low conductance. Andersen and Peres also refer to this method in their paper. The method does not depend on the choice of using either Nibble by Spielman-Teng or GenerateSample by Andersen-Peres. Therefore, in Algorithm 12 we use the notation \textit{Cluster}, which can be substituted for either Nibble of GenerateSample. The idea behind the procedure is simple. Vertices with a high degree are likely to be contained in a well-connected subgraph. Hence we prefer to grow our clusters from these vertices instead of from relatively isolated vertices. In each iteration, the vertex of highest degree not yet assigned to any cluster is selected. From this vertex, a cluster is grown according to one of the generating schemes until a particular stopping criterion is reached. Next, the vertices contained in this cluster are removed from the graph and the procedure is repeated on the graph induced by the remainder of the vertices.
The procedure terminates if no vertices are left. The more detailed mathematical description is given in Algorithm 12. Here $E|_V$ denotes the subset of the edges $E$ that have both endpoints in $V$. Even though the idea behind the procedure makes sense and the computation time of this procedure including the cluster generation algorithms is very short compared to the global algorithms, it has a downside for our application. Initially the algorithm produces good quality clusters indeed. However, as time progresses, the graph consists of several leftovers, containing relatively few vertices, which cannot be considered as good clusters. Also, we are not able to set the desired number of clusters beforehand. Hence we resort to our own methods for incorporating the cluster generating algorithms by Spielman-Teng and Andersen-Peres.

### 7.3 Seed selection

Since both the Spielman-Teng and the Andersen-Peres algorithm construct their clusters from one single vertex, we need to select at least $k$ of such starting vertices in order to produce a $k$-way partition of the graph. However, choosing exactly $k$ of such seeds might not result in the best covering of the vertices. Therefore, we choose to select many more starting vertices, say $m$, and pick the $k$ ‘best’ generated clusters of these $m$ afterwards. How we select the $k$ best exactly will be shown in the next section.

There is some similarity between selecting seed vertices and the selection of initial medoids in the $k$-medoids algorithm, as studied in Subsection 3.2.2. We noted that in choosing these initial vertices, it was important to select vertices with large degree, because these are likely to be at the center of well-connected clusters of vertices. Another observation we made is that the initial medoids would have to be evenly dispersed across the graph.

These two observations will also play a role in selecting the seeds for a local clustering algorithm. Vertices with a large degree are likely to be contained in low-conductance subgraph, which is exactly what the algorithms of Spielman-Teng and Andersen-Peres are designed to search for. Secondly, if two nearby seeds are selected, both contained in the same low-conductance cluster, the two resulting clusters will most probably be very similar or even equal. To reduce running time, it is therefore beneficial to select seeds that are well-dispersed across the graph in order to avoid duplicates.

The inspiration for the method we designed to select the seeds comes from the algorithm presented in [GLMY11]. Label the vertices of a graph in order of descending degree. Then, the first vertex in this list is added to the set of seed vertices. Next, all neighbours of this first seed

| Input: Graph $G = (V, E)$, conductance threshold $\varphi^*$. |
| Output: Partition $P = \{P_1, ..., P_l\}$ of the graph $G$. |

Set $i = 1$;

while $V \neq \emptyset$ do

| Select vertices $x \in V$ with probability $d(x)/\mu(V)$; |
| $P_i = \text{Cluster}(G, x, \varphi^*)$; |
| $V \leftarrow V \setminus P_i$; |
| $G \leftarrow (V, E|_V)$; |
| $i \leftarrow i + 1$; |

end

Algorithm 12: Algorithm Partition by Spielman-Teng
vertex are removed from the set of vertices, and the vertex of highest degree in the remaining list is added to the list, etcetera. This process is iterated until no vertices remain. Clearly, this method considers both of the requirements stated before, that is, vertices with high degree are favoured and neighbouring vertices cannot both be chosen as seeds. Nevertheless, applying this method to the street light network of deployment A with connectivity distance equal to 300 meters, yields 1736 seed vertices. Growing one cluster around each of these seeds is very time consuming and still results in (almost) duplicate clusters. Nevertheless, we used the idea behind this algorithm to select seeds, but instead of removing only neighbours of selected seeds from the list of vertices, we remove all nodes within a certain distance from the selected vertex. Since the coordinates of the vertices are available, this can be done easily, resulting in the seed selection algorithm described in Algorithm 13.

\begin{algorithm}
\textbf{Input:} Graph $G = (V, E)$, distance $D$.
\textbf{Output:} Seed vertex set $S \subseteq V$.
 Relabel the vertices in $V$ according to their degree in descending order.;
 Set $S = \emptyset$.;
\textbf{while} $V \neq \emptyset$ do
\begin{itemize}
  \item Select the first vertex $x \in V$.;
  \item $S \leftarrow S \cup \{x\}$;
  \item $V \leftarrow V \setminus \{v \in V \mid \|v - x\| \leq D\}$;
\end{itemize}
end
\end{algorithm}

Algorithm 13: Seed selection algorithm

In the algorithm, $\| \cdot \|$ denotes the Euclidean distance measure. The value of $D$ depends greatly on the topology of the deployment. Naturally, in dense networks, a small value of $D$ is preferred, in order to find sufficiently many seed points. Also, decreasing the value of $D$ increases the number of seed vertices and thereby the number of constructed clusters and overall running time. For our application where $k$ typically ranges from 10 to 25, we find that generating $m = 50$ clusters is sufficient. The corresponding value of $D$ does not follow immediately from this, but we can easily derive this value by doing some fine tuning. For instance, selecting 50 seeds in deployment A translates to $D = 2300$ meters, while the same number of seeds translates to $D = 600$ meters in the deployment B.

### 7.4 Choosing clusters for covering

Suppose we have selected $m$ seed vertices and constructed a cluster starting from each of these vertices. Accordingly, these clusters have size equal to (in Spielman-Teng) or slightly larger than size (in Andersen-Peres) $(1 + \epsilon)|V|/k$ with $\epsilon > 0$ small. In the optimal case, we are able to select $k$ of the $m$ clusters that cover all vertices of the graph. In this scenario, we could apply a so-called set covering algorithm.

The set covering problem is defined as follows. Given a universe $U$, that is, a set of objects, like vertices, a collection $C$ of subsets of $U$, such that $\bigcup_{S \in C} S = U$ and a cost function $c$, which assigns a value to each $S \in C$, find the subset $T \subseteq U$ minimising $\sum_{S \in T} c(S)$ such that $\bigcup_{S \in T} S = U$. Here, we would set $U = V$, $C$ equal to the set generated clusters and $c(S) = 1$ for all clusters $S$. However, we are unlucky in the sense that in most outcomes of the cluster generation algorithms, the union of all clusters does not cover the entire graph. A relaxation of the set covering problem is the maximum coverage problem, which is defined
CHAPTER 7. AUXILIARY ROUTINES

as follows. Given a universe \( \mathcal{U} \), a collection \( \mathcal{C} \) of subsets of \( \mathcal{U} \) and an integer \( k \), find a subset \( \mathcal{T} \subset \mathcal{C} \) maximising \( \sum_{S \in \mathcal{T}} |S| \), such that \( |\mathcal{T}| = k \). This problem setting resembles our problem much better, since we are able to choose the number of clusters beforehand.

Compared to the set covering problem, much less attention has been devoted in the literature on the maximum coverage problem. Both problems are proven to be NP-hard, and hence no polynomial time algorithm is known (and does not exist, unless P=NP) that solves them optimally. The most basic algorithm to find a solution for the maximum coverage problem is the greedy algorithm, as given in Algorithm 14.

\[
\begin{align*}
\text{Input:} & \quad \text{Universe } \mathcal{U}, \text{ collection } \mathcal{C} \text{ of subsets of } \mathcal{U}, \text{ integer } k. \\
\text{Output:} & \quad \text{Subset } \mathcal{T} \subset \mathcal{C} \text{ such that } |\mathcal{T}| = k. \\
& \quad \text{Set } \mathcal{T} = \emptyset; \\
& \quad \text{Set } \mathcal{A} = \emptyset; \\
& \quad \text{for } i = 1, \ldots, k \text{ do} \\
& \quad \quad \text{Let } C \in \mathcal{C} \text{ such that } |C \setminus \mathcal{A}| \text{ maximal}; \\
& \quad \quad \mathcal{T} \leftarrow \mathcal{T} \cup \{C\}; \\
& \quad \quad \mathcal{A} \leftarrow \mathcal{A} \cup C; \\
& \quad \text{end}
\end{align*}
\]

Algorithm 14: Greedy algorithm for maximum coverage problem.

In [Vaz01], it is proved that the greedy algorithm is an \((1 - 1/e)\)-approximation algorithm. That is, the number of vertices covered by collection \( \mathcal{T} \) outputted by the greedy algorithm is at least \((1 - 1/e)\) times the number of vertices covered by the optimal solution. The greedy algorithm hence provides a reasonably good partial cover of the vertices. Nevertheless, we are able to improve this covering with the help of an iterative improvement algorithm. We choose to use the \(m\)-Opt heuristic. The idea behind this algorithm (applied to the maximum coverage problem) is simple. In each iteration, chose \(m\) sets of the current solution \( \mathcal{T} \) and remove them from the covering. Then, select the \(m\) sets of \( \mathcal{C} \) not in \( \mathcal{T} \) such that the size of the union of the \( \mathcal{T} \) with the \(m\) sets is maximal. This process is repeated until \( \mathcal{T} \) does not change or a maximum number of iterations is reached. The mathematical formulation of the \(m\)-Opt heuristic is given in Algorithm 15.

Naturally, the larger the value of \(m\), the longer the running time of the heuristic. We will work with \(m = 3\). Note that this algorithm does not guarantee an optimal solution, since it might get stuck in a local optimum. However, since \(k\) is typically between 10 and 25, choosing \(m = 3\) does give a large enough set of so-called neighbouring solutions. Two solutions are said to be neighbours if they differ in at most \(m\) sets. Therefore, it seems reasonable to assume that the \(m\)-Opt heuristic provides a good quality solution with the greedy cover as initial solution.

7.5 Covering to partition

After we have derived a subset of the constructed clusters of size \(k\) that cover most of the vertices of the graph, we need to obtain a valid partition from this partial cover. The chosen clusters form the basis for this partition. However, there will be vertices in the graph assigned to two or more of the selected clusters. Since each vertex can only be assigned to one cluster, we need to decide on an assignment rule for vertices contained in two of more clusters. Furthermore, the graph will most probably contain vertices that are not covered by any of the clusters. We also need to account for this.
Input: Universe $U$, collection $C$ of subsets of $U$, integers $k$ and $t_{\text{max}}$, initial solution $\mathcal{T}_0$. 

Output: Subset $\mathcal{T} \subset C$ such that $|\mathcal{T}| = k$ and $|\bigcup_{R \subseteq \mathcal{T}_0} R| \leq |\bigcup_{R \subseteq \mathcal{T}} R|$.

Set $\mathcal{T} \leftarrow \mathcal{T}_0$;

for $i = 1, \ldots, t_{\text{max}}$ do

| Randomly select $\{R_1, \ldots, R_m\} \subseteq \mathcal{T}$; |
| $R \leftarrow \mathcal{T} \setminus \{R_1, \ldots, R_m\}$; |
| Select $\{T_1, \ldots, T_m\} \subseteq C$ such that $\mathcal{R} \cup \{T_1, \ldots, T_m\}$ gives maximal coverage; |
| Set $\mathcal{T}^* \leftarrow \mathcal{R} \cup \{T_1, \ldots, T_m\}$; |
| if $\mathcal{T}^* = \mathcal{T}$ then |
| | return $\mathcal{T}$ |
| else |
| | Set $\mathcal{T} \leftarrow \mathcal{T}^*$; |
| end |
end

Algorithm 15: $m$-Opt heuristic for maximum coverage problem.

Little to no research has been done on methods to transform a partial cover to a partition. Therefore, we had to design our own algorithm. The idea behind the algorithm is simple. First, we remove all vertices that are contained in multiple clusters. After that, we ‘inflate’ each of the reduced clusters, until all vertices have been assigned. A concise statement of the method is given in Algorithm 16. In the algorithm $B(S)$ denotes the boundary of a set of vertices $S$, and $\mathcal{N}(x)$ denotes the set of neighbours of a vertex $x$.

Experiments show that this method may produce a partition with disconnected clusters. This is a result of the deletion of vertices assigned to more than one cluster. We will illustrate this issue with an example in the next section. Even though this scenario is unlikely, and the number of disconnected nodes is usually small, we need to correct the partition in order to satisfy our connectivity requirement. We do this in a similar fashion as in Algorithm 16. First, we detect which clusters are disconnected. Next we keep only the largest component of the graph and remove the smaller components from the cluster. Again, we grow all sets of $\mathcal{P}$, until all removed vertices have been reassigned. This procedure evidently results in a connected partition of the graph. We present this correction algorithm formally in Algorithm 17.

The complete procedure to construct a partition of size $k$ using local algorithms is hence:

1. Select seeds $S = \{s_1, \ldots, s_m\}$ (Algorithm 13),

2. Generate clusters $\mathcal{C} = \{C_1, \ldots, C_m\}$ with $s_1, \ldots, s_m$ as starting vertices using either Spielman-Teng or Andersen-Peres,

3. Construct an initial partial cover $\mathcal{T} = \{T_1, \ldots, T_k\}$ (Algorithm 14),

4. Optimise the partial cover using 3-Opt heuristic (Algorithm 15),

5. Transform the partial cover into a partition $\mathcal{P} = \{P_1, \ldots, P_k\}$ (Algorithm 16),

6. Correct the partition (Algorithm 17).
CHAPTER 7. AUXILIARY ROUTINES

Algorithm 16: Partial cover to partition algorithm.

\textbf{Input:} Graph $G = (V, E)$, partial cover $\mathcal{T} = \{T_1, ..., T_k\}$.
\textbf{Output:} Partition $\mathcal{P} = \{P_1, ..., P_k\}$ of $G$.

Set $B \leftarrow V \setminus \left( \bigcup_{i=1}^{k} T_i \right)$;

\textbf{for} $i = 1, ..., k$ \textbf{do}
    Set $P_i \leftarrow T_i \setminus \left( \bigcup_{j \neq i} T_j \right)$;
    $B \leftarrow A \cup (T_i \setminus P_i)$
\textbf{end}

\textbf{while} $B \neq \emptyset$ \textbf{do}
    \textbf{for} $i = 1, ..., k$ \textbf{do}
        \textbf{for} $\forall x \in B(P_i)$ \textbf{do}
            $P_i \leftarrow P_i \cup (B \cap \mathcal{N}(x))$;
            $B \leftarrow B \setminus \mathcal{N}(x)$;
        \textbf{end}
    \textbf{end}
\textbf{end}

Algorithm 17: Partition correction algorithm.

\textbf{Input:} Graph $G = (V, E)$, partition $\mathcal{P} = \{P_1, ..., P_k\}$.
\textbf{Output:} Partition $\mathcal{P}^* = \{P_1^*, ..., P_k^*\}$ of $G$ without disconnections.

Set $B \leftarrow V$ \textbf{for} $i = 1, ..., k$ \textbf{do}
    Set $P_i^* \leftarrow \text{largestComponent}(P_i)$;
    $B \leftarrow B \setminus P_i^*$
\textbf{end}

\textbf{while} $B \neq \emptyset$ \textbf{do}
    \textbf{for} $i = 1, ..., k$ \textbf{do}
        \textbf{for} $\forall x \in B(P_i^*)$ \textbf{do}
            $P_i^* \leftarrow P_i^* \cup (B \cap \mathcal{N}(x))$;
            $B \leftarrow B \cup \mathcal{N}(x)$;
        \textbf{end}
    \textbf{end}
\textbf{end}
7.6 Example

We illustrate all steps in a stylised example. Consider the graph depicted in Figure 7.1. Suppose we want to partition this graph into four subsets. First, we apply the seed selection algorithm and sort the vertices in descending order according to their degrees. There are several vertices in the graph with the maximum degree equal to 7. We may select any of these as our first seed. We choose to take the vertex labeled $s_1$ in Figure 7.2. Excluding the vertices within the range depicted by the circular area around $s_1$, we search for the remaining vertex of maximum degree and find $s_2$. Again, we exclude the vertices laying too close by $s_2$. Repeating this yields the seed vertices $s_1, \ldots, s_8$ as shown in Figure 7.2.

Suppose the local algorithm produced the eight clusters presented in Figure 7.3 with $s_1, \ldots, s_8$ as starting vertices. Next, we apply the greedy algorithm in Algorithm 14 to find a initial partial cover. Starting with the initial cover $C = \emptyset$, in each iteration, we include the cluster that adds the largest number of vertices to the cover. Hence, in the first iteration we select the largest cluster, that is cluster (a), which consists of 15 vertices, $T = \{a\}$. In the second iteration, we find that cluster (b) to (h) add 9, 13, 11, 9, 11, 6 and 5 vertices to the union. We therefore select (c) and set $T = \{a,c\}$. Continuing this procedure gives the greedy
Figure 7.3: Eight sample clusters generated from the seed vertices $s_1, ..., s_8$.

(a) Greedy partial cover. (b) Partition before correction. (c) Partition after correction.

Figure 7.4: Phases of the local clustering routine.

The solution $T = \{a,c,d,b\}$, depicted in Figure 7.4a.

The next step is to apply the 3-Opt heuristic to $T$. However, after performing a few iterations, we find that no improvement can be made and $T$ is the partial cover with maximal coverage. We then remove the one vertex which is part of both cluster $a$ and $c$ and expand the boundaries of all clusters. In case of a tie, assign the vertex under consideration to the cluster of smallest size. This yields the partition given in Figure 7.4b.

This partition clearly contains a disconnected cluster, namely cluster $a$. Removing the smallest component, which in this case is a single vertex, and repeating the ‘inflating’ procedure gives the connected partition in Figure 7.4c.
Chapter 8

Simulation

8.1 Wireless networking

At this point in time it is not possible to compare the different segmentations of the street light deployments in the real environment, for the simple fact that the system has not been fully deployed for any of the data sets we considered. Therefore, we need to resort to simulation in order to compare the quality of the segmentation produced by spectral clustering, $k$-medoids and the two local algorithms.

In Chapter 1, we already discussed the routing procedures used by the network, the so-called flooding for SC-to-OLC communication and SinkDV for OLC-to-SC communication. However, this is not the full story. The transmission of packets is steered by the IEEE 802.15.4 standard. Though the structure of IEEE 802.15.4 is very complex, we will only discuss the dominant features that are most relevant for the performance of the multi-hop wireless networks we are studying. The standard specifies the so-called medium access control (MAC) and is used for low-rate wireless networking. Basically, the MAC data communication protocol provides channel access control mechanisms and decides in a distributed fashion when nodes are allowed to transmit. It relies on the carrier sense multiple access with collision avoidance (CSMA/CA) principle in order to reduce interference and thereby loss rates. The main idea behind CSMA/CA is to refuse a node access to the channel when nearby nodes are currently transmitting. A more elaborate description is given in the next section.

For the traffic directed towards the segment controller is believed to be the dominant over traffic towards the individual OLCs, we will focus on this first kind of packets only in our simulation model.

8.2 Physical model

This section describes the main concepts of the data transmission between OLCs and their segment controller as also presented in [IEE06]. Although we try to give a short description that is as concise as possible, we stress that the full specification of the IEEE 802.15.4 is far more detailed than we are able to explain here. Nevertheless, we will try to give a realistic overview of the procedures.

8.2.1 Packet generation and uplink selection

The only type of data that is sent to the segment controller by an OLC are log reports. These reports contain information on the performance of the individual street light, like energy
consumption and strength of the light produced. These log reports are generated every 15 minutes at every OLC. We will refer to these 15 minutes as the intergeneration time of the packets. In realistic scenarios, this intergeneration time can be assumed deterministic, with only a very small uniformly distributed error. The packets are assumed to have a fixed length.

Once a packet has been generated, an uplink, that is a node that is within the communication range of the first node and that has hop count one less than the originator, is selected. Naturally, there are several of such uplinks available. However, in Philips’ application, only one or two are selected for each node, due to storage limitations. The choice of these uplinks depends on the link quality, but also on the load already accumulated at the possible uplink. By routing packets to a node with relatively small offered load, load is balanced more evenly. In cases in which the assigned uplinks are not available, for instance if one of the uplinks or nodes further upstream fails, other potential uplinks are allowed to be selected. Also, at each OLC, a blacklist is stored, consisting of neighbouring nodes that the OLC is forbidden to choose as uplink, due to poor link quality or high previously detected failure rates on the link.

8.2.2 Medium access

When a valid uplink has been selected and the packet is ready for transmission, the node will attempt to get access to the channel. Here, the CSMA/CA protocol comes into play. A visualisation of this process is presented in Figure 8.1. First, we clarify the meaning of the parameters given in the scheme, which is commonly used in literature, such as [DF11]. \( NB \) denotes the number of back-off periods that have taken place for this particular transmission. The definition of a back-off period will become clear in a moment. Before each new transmission attempt, this value is set to 0. The value of \( \text{maxNB} \) specifies the maximum number of back-off periods before dropping a packet and may be fixed by the user. \( \text{BE} \) represents the current back-off exponent, which is related to the length of back-off periods. \( \text{minBE} \) is the minimum and initial value of \( \text{BE} \), while \( \text{maxBE} \) is the maximum. \( \text{BE}\text{slot} \) denotes the length of a back-off slot in seconds. Last, the counter \( c \) keeps track of the number of consecutive times the channel has been sensed idle.

The first step in the protocol is to start a back-off period. Each back-off period consists of a number of slots of length \( \text{BE}\text{slot} \). The number of slot in a back-off period is uniformly distributed on \([0, 2^{\text{BEmin}} - 1]\). Hence, in the first stage a back-off period of length \( U[0, 2^{\text{BEmin}} - 1] \cdot \text{BE}\text{slot} \) is started. When this back-off period expires, a clear channel assessment (CCA) is performed. During an CCA, the node checks whether surrounding, potentially transmitting, nodes cause too much interference. This is expressed most realistically by the SINR model. SINR stands for signal-to-interference-and-noise-ratio and reflects the ratio between power actually intended for the receiving node, and interference and background noise. Here is a short introduction to this notion. Whenever a node transmits a packet at a certain power \( P_s \), the power of the signal fades with distance. This is commonly referred to as path loss. The power left at a distance \( d \) from the source is given by the following relation:

\[
P_{sf} = P_s G_s G_r \left( \frac{\lambda}{4\pi d} \right)^\alpha,
\]

where \( P_r \) is the deteriorated power received by the receiver node at distance \( d \), \( G_s \) and \( G_r \) are parameter inherent to the devices, \( \lambda \) is the wavelength at which transmission takes place and \( \alpha \) is believed to be within the interval \([2,3]\). If multiple devices around a node generate noise, the received power simply adds up. Going back to the CCA procedure, the node senses the total amount of power it receives. It this power exceeds a prespecified threshold,
8.2. PHYSICAL MODEL

Figure 8.1: Transmission scheme.

the node concludes a neighbouring transmission is taking place, which could cause too much
interference for its own transmission. If this is the case, $NB$ and $BE$ are both increased by
one, and a new random back-off period is initiated before checking the channel again. If
at a certain point in time $NB$ exceeds the value $maxNB$, the packet is dropped. If the power
received did not exceed the threshold, the channel is considered idle. If two consecutive CCAs
(separated by a back-off period) sensed the channel idle, the transmission is considered secure
and the packet is sent to the selected uplink.

8.2.3 Receiving the packet

Suppose a node $r$ is receiving a packet from a transmitting node $s$. During the transmission,
interference or background noise might cause the transmission to fail, even though the CCA
attempted to avoid this. Node $r$ checks whether the packet it received was changed due to
interference in the following way. The packet is divided into say $m$ equally sized pieces.
During each of this time slots, the signal-to-interference-and-noise-ratio is calculated, which
is defined as

$$SINR_{s,r} = \frac{P_{s,r}}{I + N},$$

(8.2)

where $P_{s,r}$ is the power of the signal of $s$ received by $r$ as in Equation (8.1), $I = \sum_{v \in \mathcal{N}(r)} P_{v,r}$
where $\mathcal{N}(r)$ denotes the nodes surrounding $r$ and $N$ is the background power, which is
usually fixed. If the amount of interference detected in these slots is too large, the packet is
discarded and considered erroneous. If on the other hand, the check for errors came back
negative, the packet is assumed to be correct and an acknowledgement is sent back to the
transmitting node (without the CSMA/CA protocol). Accordingly, the newly arrived packet is placed in the queue of packets waiting to be transmitted towards the next uplink and the whole procedure is repeated at this node. If the transmitting node does not receive an acknowledgement within a certain time limit after the transmission finished, the transmitter node assumes the packet did not arrive correctly at the receiver node. Consequently, the node may initiate a retrial, according to user preference.

8.3 Abstracted model

8.3.1 Model in low traffic

We mentioned earlier that each node generates one packet every 15 minutes. This translates to an intergeneration rate of 1/900 packet per sec per node. Considering the size of a segment to be of order 2000, the number of packet generated per second in the whole segment is roughly $20/9 \approx 2.22$. The typical length of one packet is 100 bits, while the data rate is fixed to 100 kbit/s for this application. Hence the transmission time of packet is equal to $100/(100 \cdot 128) = 0.0078125$ seconds. These figure indicate that we are dealing with a low-traffic wireless network. In these low-traffic regimes, the probability of a one nodes interfering with the signal of another transmission is low, hence the event of more than one nodes being active in the neighbourhood of a receiving node is very unlikely. By this observation, we can simplify the complex SINR model to a more convenient model. The most important change is that interference levels no longer play a role. Instead, a packet is immediately considered erroneous if one neighbour is active while the transmission is taking place. Although this modification might seems like a major abstraction from the realistic situation, we believe that this abstracted simulation model still resembles the main features of the wireless network in low-traffic by the reasoning above. We next discuss how this modification acts out through the rest of the model.

8.3.2 Modifications

Instead of detecting whether the accumulated power of signals received by a node that is trying to initiate a transmission, is higher than a certain threshold for the CCA, we introduce a parameter $\beta$, which we refer to as the sensing range. During a CCA, the node checks whether one or more nodes within Euclidean distance $\beta$ is currently active and transmitting. If so, the channel is considered busy, if not CCA is returned true. This is the only change we make concerning the protocol for initiating the transmission described in Figure 8.1 is still followed accordingly.

Another difference concerns the choice of the uplink. In practice, each node distributively decides by itself on its one or two active uplinks. In this choice, each node takes into account the packet load already accumulated at the potential uplinks. Because the choice of these uplinks is strongly depends on the history of the performance of the network, it is therefore hard to retrieve before starting the simulations. Instead, we randomly choose one of the nodes that is one hop closer to the SC (a parent node) instead. We assume for every packet, one of the parents is selected with equal probability. Applying this randomised selection procedure implies that we also neglect the blacklisting of parents.

As we mentioned before, threshold values for the SINR model are typically unknown. That means that also in the process of receiving a packet we need to make an abstraction. Instead of checking the interference level during the intervals of the transmission in order to determine the probability of a packet being erroneous, we introduce a parameter $\eta$, which we
will call the interference range. In our simpler model, during the time that a node is receiving a packet, it continuously checks whether any node other than the node whose packet it is receiving, is active within distance $\eta$. If not, the packet is assumed to be received correctly and placed in the packet queue. If interference was detected, the packet is assumed to be erroneous and discarded. Either way, no acknowledgement is sent back to the transmitter node and hence we also do not consider retrials.

### 8.3.3 Parameter settings and performance measures

For all future simulation runs, we fix the network parameters to be able to fairly compare the results of different network segmentations. In Table 8.1, an exhaustive list of the network parameters is given. The parameters $minBE$, $maxBE$, $BEslot$ and $maxNB$ are set to the default values of the IEEE 802.14.5 standard, as given in [DF11]. The packet length and data rate were beforehand specified by [GSA+], whereas in the real deployment, a node normally generates one packet every 15 minutes, giving a traffic rate of $1/900$ packet per second per node.

From the simulation application, we are able to retrieve any desired performance measure. During a full simulation run of a particular segmentation of the deployment, the wireless network induced by each segment separately is simulated. Of each of the segments, following performance measures and characteristics are stored:

- **Size,**
  The number of OLCs contained in the segment.

- **Hop count,**
  The number of intermediary nodes a packet has to pass through before reaching the SC. Both the average and maximum hop count are stored for each segment.

- **Delay,**
  The amount of time between the time the packet was generated and the time it reached the SC.

- **Failure rate,**
  The fraction of packets that did not reach the SC.

- **Maximum duty cycle,**

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$</td>
<td>300 meters</td>
</tr>
<tr>
<td>$\eta$</td>
<td>300 meters</td>
</tr>
<tr>
<td>$\beta$</td>
<td>450 meters</td>
</tr>
<tr>
<td>Packet length</td>
<td>100 bits</td>
</tr>
<tr>
<td>Data rate</td>
<td>100 kbyte per sec</td>
</tr>
<tr>
<td>Generation rate</td>
<td>$1/900$ packet per sec per node</td>
</tr>
<tr>
<td>$minBE$</td>
<td>3</td>
</tr>
<tr>
<td>$maxBE$</td>
<td>5</td>
</tr>
<tr>
<td>$BEslot$</td>
<td>$320 \cdot 10^{-6}$ sec</td>
</tr>
<tr>
<td>$maxNB$</td>
<td>6</td>
</tr>
<tr>
<td>Simulation time</td>
<td>100000 sec</td>
</tr>
</tbody>
</table>

Table 8.1: Parameter settings for simulation.
The fraction of time the OLC with the highest load, spent transmitting.

We note that only the delay, that is the time between the generation of the packet and the arrival of the packet at the SC, of successful packets is taken into account. Once these values are obtained for all segments separately, we derive the overall averages and maxima of the measures listed above. The quality of a segmentation of the network will be evaluated on these overall values.
Chapter 9

Comparison

In this chapter, we will investigate the performance of our new partitioning methods in the context of the wireless messages that need to be spread over the light network. We will compare the following algorithms:

1. Spectral clustering;
2. $k$-medoids with spectral clustering initialisation;
3. Spielman-Teng (both weighted and unweighted);
4. Andersen-Peres (both weighted and unweighted).

We mainly focus on comparing the global algorithms (spectral clustering and $k$-medoids) and the local algorithms (Spielman-Teng and Andersen-Peres). We hope to empirically show that our new algorithms outperform the existing algorithms. First in Section 9.1, we will compare the results for the three deployments discussed in Chapter 1 in terms of easily computable graph characteristics, such as sizes of the segments and hop count. In Section 9.2, we simulate the wireless subnetworks that follow from the segmentations to see whether there are significant differences in performance. After that, in Section 9.3 we will study some specific structures in segments, which can be linked to bad network performance, as we will verify by simulation. In Section 9.4, the number of clusters to be constructed is varied, so that we can investigate whether one algorithm generally outperforms all other algorithms for a particular deployment and performance measure. In Section 9.5, we will give some concluding remarks.

9.1 Graph-related properties

In the remainder of this chapter, we mainly focus on three settings:

- Deployment A, 14 segments;
- Deployment B, 10 segments;
- Deployment C, 18 segments.

We choose these particular numbers of clusters, to achieve an average number of nodes per cluster of about two thousand, which is within the expected capacity range of a segment controller. The segmentations resulting from the six algorithms are given in Figure 9.1, 9.2 and 9.3 respectively, for these three deployments. Each which each segment is depicted with a different colour.
Figure 9.1: Segmentations for deployment A, 14 clusters.
9.1. GRAPH-RELATED PROPERTIES

(a) Spectral clustering

(b) $k$-medoids

(c) ST (unweighted)

(d) ST (weighted)

(e) AP (unweighted)

(f) AP (weighted)

Figure 9.2: Segmentations for deployment A, 10 clusters.
Figure 9.3: Segmentations for deployment C, 18 clusters.
9.1. GRAPH-RELATED PROPERTIES

9.1.1 Connectivity

One vital requirement for the segmentation of the wireless network is that each segment is connected, so that each segment controller is able to communicate with all assigned OLCs. If we look carefully at the segmentation of deployment A constructed by spectral clustering, we see that it contains two disconnected clusters, one of which is the dark green coloured cluster, which consists of a larger component in the middle, and a smaller piece in the south of the deployment. In case of the deployment B we also detect a few nodes in the north east and south which are not connected to the larger component of their cluster in the segmentation constructed by spectral clustering. The segmentation of deployment C by spectral clustering contains multiple disconnections.

Although the procedure behind spectral clustering is largely a black box, because the clustering based on the eigenvectors is far from transparent, we have some intuition about why the disconnections can occur. In the three cases of a disconnected cluster discussed above, the smaller component consisted of nodes that are relatively isolated. In view of the eigenvectors of the Laplacian, the associated points in $\mathbb{R}^k$, as explained in Algorithm 2 on page 33, are not similar to any other group of these points. Therefore, the $k$-means algorithm, which is a subroutine of the spectral clustering algorithm, is obliged to assign this small group of points to one of the larger clusters of points, which is possibly not the cluster it is actually connected to.

In fact, the spectral clustering routine is the only method out of the six algorithms discussed in this thesis that might produce disconnected segments. Segmentations by $k$-medoids are connected by construction, because each node is assigned to its closest medoid based on hop count, which translates to each OLC being able to communicate with its SC. In the local algorithms, single clusters are constructed by including well-connected nodes one by one, clearly resulting in connected clusters. The partition-correcting algorithm described in Algorithm 17 ensures that each cluster remains connected after the partitioning routine. Because spectral clustering (without additional heuristics) in many cases fails to provide a feasible segmentation, we omit the results of this method from our analysis in the remainder of this chapter.

9.1.2 Size

Another desired property concerns the size of the clusters. We argued that a balanced partition, i.e., one where all segments contain approximately the same number of nodes, is desired in general, provided that the segments are ‘well-connected’ within. The offered load per segment controller would then be divided nicely, so that the segmentation does not contain one huge cluster deteriorating the performance of the full network. The sizes of the clusters of each segmentation are given in Table 9.1, 9.2 and 9.3 for deployments A, B and C, respectively.

If we use as benchmark the perfectly balanced segmentation, the clusters would all be of size $30547/14 \approx 2181.2$ for deployment A, $19846/10 = 1984.6$ for deployment B and $32379/18 \approx 1798.8$ for deployment C. If we study the sizes of the segments for A for the global algorithms compared to the local algorithms, we immediately notice that the variance of the sizes for the spectral clustering and $k$-medoids algorithm is relatively large. This is reflected in the standard deviation and the box plots in Figure 9.4a. The same observation can be made for deployment B. Hence, for these deployments, the modifications we made in order to produce a balanced partitioning paid off. Although most sizes by spectral clustering and $k$-medoids are of the same order, the segmentations by both algorithms contain one cluster that
### Table 9.1: Sizes of the 14 segments produced by the algorithms for deployment A.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Spectral</th>
<th>(k)-Medoids</th>
<th>ST</th>
<th>ST (w)</th>
<th>AP</th>
<th>AP (w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>111</td>
<td>3448</td>
<td>1973</td>
<td>2576</td>
<td>1688</td>
<td>1774</td>
</tr>
<tr>
<td>2</td>
<td>2372</td>
<td>2327</td>
<td>2298</td>
<td>2027</td>
<td>2217</td>
<td>2036</td>
</tr>
<tr>
<td>3</td>
<td>2432</td>
<td>3534</td>
<td>1985</td>
<td>2012</td>
<td>2530</td>
<td>3498</td>
</tr>
<tr>
<td>4</td>
<td>1219</td>
<td>1537</td>
<td>2632</td>
<td>1685</td>
<td>2229</td>
<td>2493</td>
</tr>
<tr>
<td>5</td>
<td>3212</td>
<td>2797</td>
<td>2512</td>
<td>2335</td>
<td>2448</td>
<td>1907</td>
</tr>
<tr>
<td>6</td>
<td>3753</td>
<td>2557</td>
<td>1481</td>
<td>2213</td>
<td>1646</td>
<td>2464</td>
</tr>
<tr>
<td>7</td>
<td>3459</td>
<td>2573</td>
<td>1989</td>
<td>2359</td>
<td>2179</td>
<td>2030</td>
</tr>
<tr>
<td>8</td>
<td>1430</td>
<td>1378</td>
<td>2312</td>
<td>2410</td>
<td>2177</td>
<td>2657</td>
</tr>
<tr>
<td>9</td>
<td>908</td>
<td>1732</td>
<td>2138</td>
<td>2326</td>
<td>1964</td>
<td>1634</td>
</tr>
<tr>
<td>10</td>
<td>3400</td>
<td>2136</td>
<td>2596</td>
<td>2050</td>
<td>2684</td>
<td>1836</td>
</tr>
<tr>
<td>11</td>
<td>2306</td>
<td>2127</td>
<td>1979</td>
<td>2373</td>
<td>1781</td>
<td>1574</td>
</tr>
<tr>
<td>12</td>
<td>2327</td>
<td>930</td>
<td>2021</td>
<td>2203</td>
<td>2001</td>
<td>2617</td>
</tr>
<tr>
<td>13</td>
<td>2063</td>
<td>2231</td>
<td>2742</td>
<td>1959</td>
<td>2510</td>
<td>1764</td>
</tr>
<tr>
<td>14</td>
<td>1545</td>
<td>1216</td>
<td>1879</td>
<td>2009</td>
<td>2483</td>
<td>2253</td>
</tr>
</tbody>
</table>

| Minimum | 111      | 930            | 1481| 1685   | 1646| 1574   |
| Maximum | 3753     | 3534           | 2742| 2576   | 2684| 3498   |
| St. dev. | 1099    | 828            | 368 | 248    | 344 | 609    |

### Table 9.2: Sizes of the 10 segments produced by the algorithms for deployment B.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Spectral</th>
<th>(k)-Medoids</th>
<th>ST</th>
<th>ST (w)</th>
<th>AP</th>
<th>AP (w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1178</td>
<td>1664</td>
<td>2076</td>
<td>2139</td>
<td>1917</td>
<td>2121</td>
</tr>
<tr>
<td>2</td>
<td>3978</td>
<td>3117</td>
<td>2105</td>
<td>2319</td>
<td>2795</td>
<td>1541</td>
</tr>
<tr>
<td>3</td>
<td>1645</td>
<td>2066</td>
<td>1925</td>
<td>2134</td>
<td>1702</td>
<td>1634</td>
</tr>
<tr>
<td>4</td>
<td>2179</td>
<td>2305</td>
<td>2187</td>
<td>2093</td>
<td>2051</td>
<td>2305</td>
</tr>
<tr>
<td>5</td>
<td>2396</td>
<td>2563</td>
<td>2363</td>
<td>1712</td>
<td>2241</td>
<td>2117</td>
</tr>
<tr>
<td>6</td>
<td>1279</td>
<td>2371</td>
<td>1994</td>
<td>1682</td>
<td>1944</td>
<td>2245</td>
</tr>
<tr>
<td>7</td>
<td>129</td>
<td>1179</td>
<td>2074</td>
<td>2011</td>
<td>2019</td>
<td>2238</td>
</tr>
<tr>
<td>8</td>
<td>1945</td>
<td>847</td>
<td>1801</td>
<td>1904</td>
<td>2206</td>
<td>1760</td>
</tr>
<tr>
<td>9</td>
<td>3188</td>
<td>1807</td>
<td>1732</td>
<td>1788</td>
<td>1743</td>
<td>1909</td>
</tr>
<tr>
<td>10</td>
<td>1929</td>
<td>1917</td>
<td>1589</td>
<td>2064</td>
<td>1228</td>
<td>1976</td>
</tr>
</tbody>
</table>

| Minimum | 129      | 847            | 1589| 1682   | 1228| 1541   |
| Maximum | 3978     | 3117           | 2363| 2319   | 2795| 2305   |
| St. dev. | 1181   | 716            | 246 | 221    | 457 | 272    |
Figure 9.4: Box plots of the segment sizes.
### 9.1.3 Hop count

A third characteristic that can be observed almost immediately is the average hop count. As we stated before, we choose the segment controller position equal to the medoid of the cluster under consideration. In Table 9.4, 9.5 and 9.6, we give the average and maximum hop count per cluster for deployment A, B and C, respectively. In general, we can argue that a small average hop count leads to a low average end-to-end delay, since the delay mostly depends on the number of nodes a packet has to travel through before reaching the segment controller. Furthermore, in low traffic, the fewer hops a packet needs to make, the smaller the probability of losing the packet due to either interference or medium access failures.

Table 9.4 clearly shows that \( k \)-medoids produces the segmentation with the smallest hop count statistics, in the sense of both average and maximum. This is no coincidence, since the objective of the \( k \)-medoids method is to minimise the average hop count. Both Spielman-Teng and Andersen-Peres do not explicitly take the hop count into account. We see a similar
### 9.1. GRAPH-RELATED PROPERTIES

<table>
<thead>
<tr>
<th>$k$-medoids</th>
<th>ST</th>
<th>ST (w)</th>
<th>AP</th>
<th>AP (w)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Max</td>
<td>Average</td>
</tr>
<tr>
<td>1</td>
<td>5.9597</td>
<td>16</td>
<td>5.3311</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>7.5105</td>
<td>16</td>
<td>6.8685</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>6.8295</td>
<td>18</td>
<td>10.1303</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>7.9106</td>
<td>16</td>
<td>7.3002</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>6.061</td>
<td>13</td>
<td>5.2789</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>6.9953</td>
<td>14</td>
<td>6.385</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>8.7883</td>
<td>21</td>
<td>8.4165</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>7.0758</td>
<td>17</td>
<td>6.2515</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>6.7183</td>
<td>17</td>
<td>8.0638</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.3416</td>
<td>17.07</td>
<td>7.6082</td>
</tr>
</tbody>
</table>

Table 9.4: Hop count values for the segmentations with 14 clusters for deployment A.

<table>
<thead>
<tr>
<th>$k$-medoids</th>
<th>ST</th>
<th>ST (w)</th>
<th>AP</th>
<th>AP (w)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Max</td>
<td>Average</td>
</tr>
<tr>
<td>3.0222</td>
<td>5</td>
<td>3.3825</td>
<td>7</td>
<td>3.5245</td>
</tr>
<tr>
<td>4.0035</td>
<td>7</td>
<td>3.1344</td>
<td>11</td>
<td>3.3001</td>
</tr>
<tr>
<td>3.7318</td>
<td>8</td>
<td>3.2639</td>
<td>7</td>
<td>3.3777</td>
</tr>
<tr>
<td>3.2403</td>
<td>8</td>
<td>4.1687</td>
<td>9</td>
<td>3.9804</td>
</tr>
<tr>
<td>3.5244</td>
<td>8</td>
<td>3.4329</td>
<td>6</td>
<td>2.9533</td>
</tr>
<tr>
<td>3.6837</td>
<td>9</td>
<td>3.2513</td>
<td>7</td>
<td>3.2658</td>
</tr>
<tr>
<td>3.2358</td>
<td>8</td>
<td>3.4614</td>
<td>6</td>
<td>3.0622</td>
</tr>
<tr>
<td>2.4345</td>
<td>9</td>
<td>3.6047</td>
<td>7</td>
<td>3.2862</td>
</tr>
<tr>
<td>3.1339</td>
<td>7</td>
<td>2.9723</td>
<td>5</td>
<td>3.6779</td>
</tr>
<tr>
<td>3.3698</td>
<td>7</td>
<td>2.7678</td>
<td>4</td>
<td>3.2737</td>
</tr>
<tr>
<td></td>
<td>3.33799</td>
<td>7.6</td>
<td>3.3394</td>
<td>6.9</td>
</tr>
<tr>
<td></td>
<td>4.0035</td>
<td>9</td>
<td>4.1687</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 9.5: Hop count values for the segmentations with 10 clusters for deployment B.
result for deployment B in Table 9.5. Nevertheless, for deployment C, k-medoids is clearly outperformed by AP based on the maximum hop count in the segmentations. This is due to the fact that the segmentation produced by k-medoids contains one huge segment of size 4902. This observation supports our claim that a segmentation in which the sizes of the segments are more of less equal is desirable.

### 9.2 Wireless properties

In this section we analyse the outcomes of the simulations we ran on the segmentations of the deployments A, B and C. We first compare the segmentations with respect to delay. After that failure rates are analysed and finally we take a look at the duty cycle of the nodes, which is defined as the fraction of time a node is transmitting. The summarised results of the simulations are given in Tables 9.7, 9.8 and 9.9.

#### 9.2.1 Delay

The delay of a packet is defined as the amount of time between the generation of the packet at the source and its reception at the SC. We remind the reader that we only take into account the delay of the packets that were successfully received by the SC. For the outcome of each algorithm, we computed the average delay in seconds for each of the segments. The row ‘Average’ in Tables 9.7, 9.8 and 9.9 refers to the overall average of the average delay of all segments, whereas ‘Maximum’ refers to the maximum average delay among all segments. We are also able to identify the worst performing segment, that is the segment with highest delay. The summarised results of the simulations are given in Tables 9.7, 9.8 and 9.9.
### Table 9.7: Simulation results for all segmentations with 14 clusters of deployment A.

<table>
<thead>
<tr>
<th></th>
<th>k-Medoids</th>
<th>ST</th>
<th>ST (w)</th>
<th>AP</th>
<th>AP (w)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Delay (sec)</strong></td>
<td><strong>Average</strong> 0.064815516</td>
<td><strong>0.069650833</strong></td>
<td><strong>0.070522535</strong></td>
<td><strong>0.067034822</strong></td>
<td><strong>0.067034822</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Maximum</strong> 0.078655631</td>
<td><strong>0.100279331</strong></td>
<td><strong>0.103227403</strong></td>
<td><strong>0.096463798</strong></td>
<td><strong>0.096463798</strong></td>
</tr>
<tr>
<td><strong>Failure rate</strong></td>
<td><strong>Average</strong> 0.039822674</td>
<td><strong>0.0376375</strong></td>
<td><strong>0.03566073</strong></td>
<td><strong>0.036341816</strong></td>
<td><strong>0.038355946</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Maximum</strong> 0.056871458</td>
<td><strong>0.048945449</strong></td>
<td><strong>0.047286978</strong></td>
<td><strong>0.047441702</strong></td>
<td><strong>0.051952453</strong></td>
</tr>
<tr>
<td><strong>Max DC</strong></td>
<td><strong>Average</strong> 0.0004430117</td>
<td><strong>0.004456065</strong></td>
<td><strong>0.004425772</strong></td>
<td><strong>0.004288466</strong></td>
<td><strong>0.00450108</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Maximum</strong> 0.006339844</td>
<td><strong>0.008661238</strong></td>
<td><strong>0.008319453</strong></td>
<td><strong>0.007013828</strong></td>
<td><strong>0.007118203</strong></td>
</tr>
</tbody>
</table>

### Table 9.8: Simulation results for all segmentations with 10 clusters of deployment B.

<table>
<thead>
<tr>
<th></th>
<th>k-Medoids</th>
<th>ST</th>
<th>ST (w)</th>
<th>AP</th>
<th>AP (w)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Delay (sec)</strong></td>
<td><strong>Average</strong> 0.030938978</td>
<td><strong>0.030148767</strong></td>
<td><strong>0.030749577</strong></td>
<td><strong>0.031308761</strong></td>
<td><strong>0.031308761</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Maximum</strong> 0.035913154</td>
<td><strong>0.037342497</strong></td>
<td><strong>0.03851315</strong></td>
<td><strong>0.037135969</strong></td>
<td><strong>0.037135969</strong></td>
</tr>
<tr>
<td><strong>Failure rate</strong></td>
<td><strong>Average</strong> 0.026382774</td>
<td><strong>0.02354078</strong></td>
<td><strong>0.02398904</strong></td>
<td><strong>0.024769014</strong></td>
<td><strong>0.024718627</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Maximum</strong> 0.039900527</td>
<td><strong>0.028434683</strong></td>
<td><strong>0.03202757</strong></td>
<td><strong>0.030071638</strong></td>
<td><strong>0.030071638</strong></td>
</tr>
<tr>
<td><strong>Max DC</strong></td>
<td><strong>Average</strong> 0.001575183</td>
<td><strong>0.001521308</strong></td>
<td><strong>0.001572407</strong></td>
<td><strong>0.001839369</strong></td>
<td><strong>0.001731753</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Maximum</strong> 0.002320547</td>
<td><strong>0.002965625</strong></td>
<td><strong>0.002815156</strong></td>
<td><strong>0.004423203</strong></td>
<td><strong>0.003026641</strong></td>
</tr>
</tbody>
</table>

### Table 9.9: Simulation results for all segmentations with 18 clusters of the deployment C.

<table>
<thead>
<tr>
<th></th>
<th>k-Medoids</th>
<th>ST</th>
<th>ST (w)</th>
<th>AP</th>
<th>AP (w)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Delay (sec)</strong></td>
<td><strong>Average</strong> 0.108412491</td>
<td><strong>0.107049389</strong></td>
<td><strong>0.107522537</strong></td>
<td><strong>0.12790613</strong></td>
<td><strong>0.121298196</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Maximum</strong> 0.177940856</td>
<td><strong>0.203534097</strong></td>
<td><strong>0.1818248032</strong></td>
<td><strong>0.205034115</strong></td>
<td><strong>0.22486887</strong></td>
</tr>
<tr>
<td><strong>Failure rate</strong></td>
<td><strong>Average</strong> 0.051383934</td>
<td><strong>0.041720764</strong></td>
<td><strong>0.046873241</strong></td>
<td><strong>0.0500575421</strong></td>
<td><strong>0.044470472</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Maximum</strong> 0.103623391</td>
<td><strong>0.081543835</strong></td>
<td><strong>0.105197034</strong></td>
<td><strong>0.09433972</strong></td>
<td><strong>0.07545674</strong></td>
</tr>
<tr>
<td><strong>Max DC</strong></td>
<td><strong>Average</strong> 0.006717427</td>
<td><strong>0.005943525</strong></td>
<td><strong>0.006410957</strong></td>
<td><strong>0.007914312</strong></td>
<td><strong>0.006452695</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Maximum</strong> 0.011080859</td>
<td><strong>0.011837266</strong></td>
<td><strong>0.013282813</strong></td>
<td><strong>0.018883906</strong></td>
<td><strong>0.011021563</strong></td>
</tr>
</tbody>
</table>
average delay, in each segmentation.

For deployment A, we immediately see that the \( k \)-medoids algorithm performs better than all other algorithms in both the average and maximum delay. Although the difference in average delay between the worst (weighted ST) and the best (\( k \)-medoids) is small (58 milliseconds), this is a relative increase of 8.8 percent. A vast difference is shown in the maximum delay. As we argued before, the delay of a packet strongly depends on the number of hops it needs to make before reaching the SC. Since \( k \)-medoids’ objective is to minimise the average hop count, this translates to a minimisation of the average delay under this assumption. The fact that \( k \)-medoids performs best for minimising delay is hence not surprising. Our claim that delay is intimately related to hop count is verified by the scatter plot in Figure 9.5, in which we plotted the average hop count of a number of clusters against the average delay within the cluster. The figure clearly shows a linear relation between the two variables, also indicating that congestion does not play a significant role in these low-traffic networks. Nevertheless, \( k \)-medoids is outperformed by ST in deployment B. A similar observation can be made for deployment C. The reason is the poor initialisation provided by the spectral clustering routine. Apparently, the \( k \)-medoids algorithm gets stuck in a local optimum, which is not the global optimum. This suggests that initialising the \( k \)-medoids algorithm with the spectral clustering algorithm is not optimal, since this initialisation does not converge to the optimal medoid positions for this deployment. However, if one is able to initialise the medoids such that the optimal solution in terms of average hop count is found, \( k \)-medoids indeed provides the segmentation with optimal average delay.

9.2.2 Failure rate

Next, we analyse the performance of the network under different segmentations by the loss probabilities (failure rates). In the simulations, we keep track of the number of lost packets per segment and thereby calculated the loss probabilities for each segment separately. In Table 9.7, 9.8 and 9.9 we display the average and maximum failure rate among the segments. What stands out is the maximum failure rate of the segmentation by \( k \)-medoids compared to the local algorithms. In all three deployments, this value is relatively high. The segments
attaining this maximum failure rate in \( k \)-medoids are segment 3 (blue in Figure 9.1(b)) for deployment A, segment 2 (green in Figure 9.2(b)) for deployment B and segment 10 (dark green in Figure 9.3(b)) for deployment C. If we compare this to the values in Tables 9.1, 9.2 and 9.3, we see that these segments coincide with the largest segment of each segmentation. This suggests that there is a link between the size of a cluster and the corresponding failure rate. In order to verify this relation, we plotted in Figure 9.6 the size of a number of segments against their failure rates. Although the relation between the size and failure rate is not as evident as it is between hop count and delay, we definitely see a positive correlation between the two variables. Naturally, in a larger segment more data is generated and the load accumulated at nodes close to the segment controller is increased. This causes more activation of nodes in these areas, inducing more interference and hence higher loss probabilities due to collisions. Also, the probability of a node not being able to access the channel before the maximum number of back-off periods is exceeded is increased in these larger clusters.

This observation implies that balancing the sizes of the segments minimises the overall maximum failure rate. Since the local algorithms attempt to construct segments of equal size, they outperform the global methods on this network performance measure for these three deployments.

**9.2.3 Duty cycle**

The last network characteristic we discuss is the duty cycle (DC), which is the fraction of time the node actually spends transmitting. For each segment, we select the duty cycle of the busiest node, which is usually one of the neighbours of the SC. This value is referred to as the maximum duty cycle of the segment. Over all segments we computed the average and maximum of these values, as they are given in Table 9.7, 9.8 and 9.9. The reason we analyse the duty cycle is because of the requirement set by European law, stating that each node should not be active more than 1 percent of the time.

In the deployment A and B, we see that this requirement is met. However, in deployment C, we see that for all outcomes the maximum exceeds 0.01. Note however, that the sizes of the segments of deployment C are on average of the same order as the segments of the other deployments.
two deployments. The structure of deployment C is therefore inherently worse for wireless communication than the more well-knitted networks of A and B. Increasing the number of segments solves this problem. This issue will be discussed in more detail in the next section.

In Appendix B, we present the simulation results of two more deployments.

### 9.3 Special structures

In this section we zoom into some of the segments in isolation in order to find out which structural properties affect the performance of the segment either positively or negatively. We will do so by analysing some illustrative segments. We will use three segments of deployment A (Figure 9.1) of approximately the same size, namely segment number 8 of ST (weighted), segment number 3 of AP (unweighted) and segment number 10 of ST (unweighted). These three segments are depicted in Figure 9.7 together with their medoids (the SC positions), and will be referred to as segment (a), (b) and (c). Segment (a) contains a few long roads, which we expect to be bottlenecks. Segment (b) is part of the densely populated city centre of deployment A. Segment (c) represents an area of average density in the deployment.

#### 9.3.1 Overall measures

In Table 9.10 the simulation results of each of the three segments are presented. Observe that the long roads in (a) cause the average and maximum hop count to be very large. The high density of segment (b) is reflected in the relatively low average hop count, whereas the average hop count of segment (c) is indeed somewhere in the middle of the two. Note that even though segment (a) is smaller than (b) and (c), the hop count is much larger, highlighting the dispersed structure of the network even more. As we noted in the previous section, the average hop count is linked directly to the average delay. These three cases are no different.

![Figure 9.7: Three segments of deployment A.](image)

<table>
<thead>
<tr>
<th>Segment</th>
<th>Size</th>
<th>Av. hop count</th>
<th>Max hop count</th>
<th>Delay</th>
<th>Failure rate</th>
<th>Max duty cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>2213</td>
<td>11.5495</td>
<td>24</td>
<td>0.10322</td>
<td>0.04728</td>
<td>0.00831</td>
</tr>
<tr>
<td>(b)</td>
<td>2530</td>
<td>4.8960</td>
<td>14</td>
<td>0.04384</td>
<td>0.03463</td>
<td>0.00278</td>
</tr>
<tr>
<td>(c)</td>
<td>2569</td>
<td>6.2515</td>
<td>13</td>
<td>0.05595</td>
<td>0.03685</td>
<td>0.00427</td>
</tr>
</tbody>
</table>
Therefore, segment (a) performs bad compared to both segment (b) and (c) in terms of delay. When looking at the failure rates, we immediately note the relatively high loss probability of segment (a). As expected, congestion seems to occur in long streets where the street light connectivity follows a line pattern. This leads to a busy channel in these areas and hence many packet losses. Segment (b) has the lowest failure rate among these three segments, even though it is the most dense one. This can be understood by the fact that we are dealing with a low traffic network. Collisions are hence rare. Since segments (b) and (c) are almost equal in size and a higher hop count implies more possibilities of failing, the number of hops a packet has to make before reaching the SC has the upper hand in determining the failure probability.

The high value of the maximum duty cycle of segment (a) is again caused by the presence of long roads. Since all traffic on one end of the road needs to be routed along the road towards the SC, packets accumulate in these streets. This naturally leads to a high occupation rate of the nodes in these streets, and the maximum of these duty cycles will also be very high.

We again see a lower value for segment (b) compared to (c). The reason is the SC position. Obviously, all traffic intended for the SC has to pass through one of the neighbours of the SC itself. Thus in general, the more neighbours a SC controller has, the better the load is spread out across these neighbouring nodes and the lower the fraction of time each of these neighbour spends transmitting. With that said, the medoid of segment (b) has 21 neighbours, while the medoid of segment (c) has 10. This explains the difference in maximum duty cycles between segments (b) and (c).

### 9.3.2 Individual measures

In this subsection, we zoom in even more, to the level of individual nodes. For each of the three segments, we will see how the load is spread across the network, which nodes are heavily loaded or not, and at which nodes many packets are dropped.

In Figure 9.8 the duty cycle of each node is depicted by colour for all three segments. Dark blue indicates a low duty cycle, while dark red indicate a large duty cycle. From these figures, we are able to draw some conclusions. First, we see that in general, the duty cycle of a node increases if the node is closer to the segment controller. However, we see a certain wave pattern in all figures. This be explained as follows. The heavily loaded nodes are typically 300 meters apart, starting from the segment controller. Once a packet arrives at one of these nodes, it keeps following this pattern. We may also say that the nodes in this wave pattern are contains in many shortest paths between nodes and the segment controller.

The increased duty cycles of nodes in the long streets are clearly visible in segment (a), verifying our claim of highly accumulated load along these roads. This in contrast to segments (b) and (c), where the load is spread out much more evenly, leading to lower duty cycles.

Another measure we are able to depict by colour is the failure probability. Figure 9.9 shows the probability that a packet is dropped at each node. Clearly, the failure probability increases when getting closer to the segment controller. This is due to the fact that load accumulates towards the segment controller, thereby increasing interference and hence loss probabilities. In segment (b) and (c) the failure probability decreases with the hop count, as one would expect. In segment (a), we again see that the failure probability is increased along the streets. This confirms that the streets indeed act as bottlenecks in the network. The observation that all packets generated in the left part of (a) have to pass through the narrow

121
CHAPTER 9. COMPARISON

Figure 9.8: Three segments of deployment A with nodes coloured by their duty cycle.
road also confirms that the failure probability strongly depends on the offered load and so does the duty cycle of a node.

9.4 Varying the number of segments

We have seen how the outcomes of the six algorithms differ in terms of size, hop count and wireless performance measures for a fixed number of segments. In this section we will study whether the differences in delay and failure rate remain when varying the number of clusters. Also, we would like to see whether one of the local algorithms significantly outperforms the other algorithms and whether the inclusion of weights really does make a difference.

9.4.1 Delay

As we argued in the Section 9.3, the delay of a packet is mainly determined by the hop count of the node which generated it. Therefore, $k$-medoids performs very well on minimising the average delay for the three instances we extensively discussed before. We will investigate here whether this is also the case for different number of segments. In Figure 9.10(a), which depicts the average delay under the different segmentations for various number of segments, we indeed see that the dark blue line of $k$-medoids lays mostly below the other line in the left figure. The exception is $k = 16$. This peak is caused by one dense cluster of size 6028, which shows a lot of congestion. In Figure 9.10(b), where the delay of a packet in the worst performing segment is depicted, the advantage over the other algorithms is evident. In Figure 9.11, the results are shown for deployment B. Here, $k$-medoids definitely does not perform better than the other algorithms in terms of hop count. Again, this is most probably caused by the poor initialisation provided by spectral clustering. Note however that the delay of a packet is a factor 4 larger in deployment C than it is in deployment B, even though the average number of nodes per segment is of the same order. The reason for this is the structure of the network, as discussed in Section 9.2. Based on these observations, we may state that under the condition that $k$-medoids is well-initialised, the $k$-medoids in most cases produces the best segmentation regarding the delay of packets.

9.4.2 Failure rate

In Subsection 9.2.2 we saw that the local algorithms outperform $k$-medoids on both average and maximum failure rate. We will now check whether this is a coincidence or whether the local algorithms indeed produce better segmentations regarding the failure rate and duty cycle of the network. If we analyse the results depicted in Figures 9.13, 9.14 and 9.15 we immediately notice that in almost all segmentations, the $k$-medoids method performs worst regarding the failure rate. Furthermore, we observe that, with a few exceptions, the average failure rate decreases with the number of clusters as expected. Although the difference in performance between $k$-medoids and the other algorithms is evident, there is not a clear distinction to be made among the local algorithms themselves.

9.4.3 Weights

Figures 9.10, 9.15 do not provide clear evidence for the fact that the weighted versioned of our local algorithms perform better than the unweighted versions of the algorithms. In fact, the original versions tend to be more stable than the weighted ones. We have no clear explanation for this. One reason might be that the simulation we built does not take the real path losses,
Figure 9.9: Three segments of deployment A with nodes coloured by their failure probability.
which do depend on physical distances between senders and receivers, into account, whereas they might play a significant role in the link quality between nodes. Another reason might be our choice of distance function, that is the local scaling similarity function. It is possible that a different function provides better segmentations.

When deciding on one local algorithm to use, AP and ST (unweighted) both seem to perform well in terms of loss rates. However, we do not have enough evidence to state that one definitely outperforms the other.

### 9.5 Conclusion

This chapter provided a simulation-based comparison of the segmentations constructed by the algorithms. We highlighted undesired structural properties that negatively impact the performance of a segment. Our main findings are the following:

- $k$-medoids indeed minimises the overall average hop count of the segmentation, provided that it is well-initialised. The initialisation of the medoids by the spectral clustering algorithm was shown to be poor.
In general, the local algorithms produce segmentations that are more balanced in size than $k$-medoids, which does not consider size at all in the construction of clusters.

We found that congestion plays a negligible role in these low-traffic networks, which implies that there is an (almost) one-to-one relation between the hop count and the delay of a packet. Because $k$-medoids attempts to minimise the average hop count, this algorithm also performs best in minimising the average delay, again provided that it is well-initialised.

The average failure rate of a segment strongly depends on the size of the segment. Large clusters exhibit higher failure rates than small clusters in general. The failure rate is minimised if all segment sizes are of the same order. Therefore, the local algorithms outperform $k$-medoids on this measure.

The introduction of weights to the local algorithms did not provide a tangible improvement. A different weight function might improve the performance of the weighted versions of ST and AP.

We have not detected enough evidence to claim that one of the algorithms indeed out-
Figure 9.14: Average (a) and maximum (b) failure rate of a packet in deployment B for variable number of segments.

Figure 9.15: Average (a) and maximum (b) failure rate of a packet in deployment C for variable number of segments.

• Within a segment, long roads close to the segment controller position affect the performance severely. It is preferred to only include streets on the boundary of a segment, so that no large amount of traffic needs to be routed through these bottlenecks.

• To reduce the maximum duty cycle and spread the offered load more evenly, it is preferred to place the segment controller in a position where the number of first-hop neighbours is high.

Although we carefully built our simulation program, we stress that the abstractions we made in order to keep things tractable may affect the results stated in this chapter. Also, we are not able to include some real worlds features, like third party interference, reflection by buildings, hardware failures, etcetera. Therefore, we cannot give a guarantee on the error on the actual values presented in this chapter. Nevertheless, the qualitative results provided here have given insight in the performance of particular segmentations.
Chapter 10

Conclusion

In this thesis we studied the problem of segmenting deployments consisting of street lights equipped with wireless networking capabilities. This concluding chapter provides a brief recap of the topics of our study and reflects on the three central research questions posed in Section 1.4. Also, we provide suggestions for future research.

10.1 Summary of contributions

The segmentation problem we attempted to solve considered a spatial wireless network, consisting of so-called outdoor lighting controllers (OLCs), of which the coordinates are known, and segment controllers (SCs), coordinating the wireless communication within its assigned segment. Because SCs are relatively expensive compared to OLCs, SCs are greatly outnumbered by OLCs. Fixing the number of SCs, our task was to choose the position of each SC, and assign each OLC to one SC such that each OLC is able to communicate with its SC and vice versa by multi-hop communication. While doing so, the goal was to optimise the performance of the wireless network induced by the segmentation.

We started off with deriving a mathematical model from the practical setting. Information on the deployment was captured by the so-called connectivity graph, indicating which pairs of OLCs are able to communicate directly. Based on this graph, we investigated two conventional graph partitioning algorithms, namely spectral clustering and $k$-medoids. We provided some illustrative examples in an attempt to clarify the process behind both so-called global algorithms. Since these type of algorithms did not seem to deliver the desired output, we resorted to some recently developed local methods that are based on rather different mathematical methods. These local clustering algorithms by Spielman-Teng and Andersen-Peres, extensively discussed in Chapters 4 to 6, attempt to construct clusters by only considering the local structure of the graph. This has the advantage of reducing the running time and the opportunity to steer the process while constructing the clusters. We exploited this last option by modifying the local algorithm so that it suits our application. Because the partitioning routine by Spielman-Teng discussed in their paper was proven inefficient, we developed a series of algorithms which do provide a connected segmentation, completing the full local clustering method for segmentation.

We further developed a software application that simulates the wireless network given a segmentation of a street light deployment. Using this simulation program, we were able to compare the novel modified local algorithms to the previously studied global algorithms in terms of network performance. Although we did not have enough sample deployments to give definite empirical statements on the differences between the outcomes of both kinds
of algorithms, we have seen evidence that $k$-medoids tends to construct segmentations with best average delay, but performs poorly on failure rates, compared to the local algorithms. Furthermore, the local algorithms construct segments that are clearly more balanced in size. Besides the comparison of algorithms, we also used the simulation program to distinguish bad and desirable graph structures. Altogether, we were able to provide a list of graph properties which could serve as goals or objectives when partitioning a deployment of street lights.

### 10.2 Answers to research questions

The three questions posed in Section 1.4 played a central role in our research. Chapter 2 to 9 were dedicated to finding answers to these questions. We now summarise our findings.

**A) How do graph characteristics relate to the performance of the network?**

This question considers the internal structure of one segment only. In Chapter 2, we discussed a few arguments that support the claim that we prefer segments that are well-connected within and weakly connected to the outside. The concept of conductance was introduced. This notion specifies the ratio between the number of external and internal edges of a subset of the vertices of a graph. By this definition, we searched for segments of low conductance and this remained the key graph characteristic in the algorithms we investigated in Chapters 3 to 6. From the comparison by simulation in Chapter 9, we deduced the following qualitative observations on the relation between graph structure and network performance:

- **Long roads in the interior of a segment are disastrous for the performance of the wireless network.** Intuition already told us that if a segment contains a long narrow line structure of OLCs in the interior, traffic needs to be routed though this bottleneck, causing packets to accumulate along this line. This increases congestion in these regions of the graph, usually leading to interference and packet losses. Simulations in Chapter 9 confirmed this assertion, showing a large increase of failure rates in OLCs in these roads. Furthermore, the duty cycle of these nodes was relatively high. Since a line structure is a typical example of a high conductance feature, this supports our goal to search for low-conductance clusters.

- **Delay is directly related to the hop count.**
  
The delay of a packet is defined as the time between generation at the origin and reception at the segment controller. It is equal to the sum of transmission time, back-off periods and waiting time. The simulation showed that the network exhibits very low traffic rates. This implies that congestion rarely occurs, and waiting times are negligible. Furthermore, the occurrence of a busy channel is not likely, indicating the weak influence of the back-off periods on the delay. The number of transmissions necessary to reach the segment controller, i.e. the hop count, is hence the dominant factor in the determination of the delay. Figure 9.5 indeed shows a linear relation between the hop count and delay.

- **The size of a segment correlates positively with the failure rate.**
  
The number of OLCs contained in a segment defines the number of packets generated per time unit. Therefore, the larger segments tend to have a higher offered load per OLC. This translates into a busier channel, causing more packets to be dropped. This reasoning underlines the relation between the size of a segment and the failure rates it
10.2. ANSWERS TO RESEARCH QUESTIONS

exhibits. Figure 9.6 shows that there is indeed a positive correlation between these two network characteristics.

- The number of direct neighbours of the segment controller is decisive for the maximum duty cycle of the segment.

Naturally, the busiest OLCs are close to the segment controller. Each packet needs to be routed through one of the direct neighbours of the segment controller for a successful transmission. The maximum duty cycle of a segment is largely determined by the spread of offered load among these neighbours. Preferably, each neighbour receives approximately the same number of packets for forwarding per time unit. The number of neighbours of the segment controller hence also plays a big role in the spread of offered load, and thereby the maximum duty cycle.

B) Can we modify known partitioning algorithms such that they are suitable for the application of wireless networks?

In Chapter 3 we studied the spectral clustering and $k$-medoids algorithm. It became evident that these algorithms did not provide stable desired solutions for the segmentation problem. Unfortunately, these global methods do not allow enough flexibility in the construction process for us to modify them and adapt them to serve our purposes. That is why we resorted to the two local clustering techniques of Spielman-Teng and Andersen-Peres. Since these methods construct a cluster step by step, we have much more control over the construction process and may even interfere in this procedure. This allowed us to make two modifications.

The first modification is the incorporation of constraints. In Spielman-Teng, individual vertices are added one by one, which gives us the opportunity to pose restrictions on these inclusions. In Andersen-Peres, we are able to refuse transitions in the evolving set process to clusters that do not meet certain requirements. This way, the resulting cluster will always satisfy the restriction we pose on the segments. Possible restrictions we discussed in Chapter 6 are upper and lower limits on the (average) hop count and size of a cluster.

Secondly, we chose to replace the stopping criteria in the local algorithms based on conductance by a size threshold. Basically, the local algorithm are terminated once a cluster of a certain size is found. We argued that a conductance threshold is not of interest in our application. Still, the Spielman-Teng and Andersen-Peres algorithms are designed to search for low-conductance sets in each iteration, implying that each intermediary cluster is in principle well-connected within. This led us to believe that adjusting the stopping criterion does not affect the performance of the algorithms in terms of finding low-conductance clusters.

C) How does the fact that we have spatial information come into play in the segmenting the network?

The first usage of the coordinates of the OLCs appeared in the construction of the connectivity matrix, that defined which OLCs are able to communicate, i.e. which pairs of OLCs are less than 300 meters apart. In the spectral clustering algorithm discussed in Subsection 3.2, the real-valued (locally scaled) affinity matrix was also based on distances. Thereby, the spectral clustering algorithm implicitly uses distances to partition graphs. However, the most important application of spatial information is in the local clustering algorithms. The idea we came up with is to introduce a weighted connectivity matrix, and extend the Spielman-Teng and Andersen-Peres algorithm so that they can also handle weighted graphs. We did not find literature on these local algorithms applied to weighted graphs, and therefore we
adjusted them ourselves. We modified transition probabilities accordingly and found that the extended algorithms exhibit similar properties as the original versions. We did not make a definite choice on the weight function on the edges, but used the locally scaled affinity function in the comparison of Chapter 9. Despite our efforts, the inclusion of weights did not show a significant improvement in the network performance.

10.3 Suggestions for future research and final remarks

Although we did an attempt to develop methods that segments the wireless street light network close to optimally, there is always room for further research. We name a few possible topics:

- Implement a more realistic and detailed simulation program, so that the provided outcomes present the real world setting more faithfully. This might affect the results presented in Chapter 9 in favour of the weighted versions of the local algorithms, because our simulation model does not take distances into account in the calculation of interference levels, link quality, etcetera.

- Think of a better weight function in the weighted versions of Spielman-Teng and Andersen-Peres to resemble the link quality and test this choice via simulation. Although the locally scaled weights did not significantly improve the segmentation of the network, there is a wide variety of possible weight functions. If one is able to find a weight function that captures the essence of the local wireless properties, this has the potential to find a close-to-optimal segmentation regarding network performance.

- The adjustments to the Spielman-Teng and Andersen-Peres algorithms to handle weighted graphs as well reserved the overall structure of the algorithms. However, since the local algorithms were mainly designed for finding low-conductance sets, the translation to weighted graphs is not directly evident. Also, we do not know how the proofs in [ST08] and [AP08] on the efficiency and complexity change when weights are introduced. A study could be done on this modification. One might also ask whether Conjectures 6.1 and 6.2 still hold for these weighted versions.

- Apply the algorithms to more deployments in order to strengthen the observations on the differences between the segmentations by all algorithms described in Chapter 9 with some statistical evidence.

In the study that led to this thesis, we have seen that segmenting a wireless network differs from conventional graph partitioning in many ways. The difficulty in partitioning the connectivity graph comes from the fact that there is not one clear objective function to optimise for. Instead, we have derived a collection of desirable and undesirable properties of a segmentation. The novel methods we developed account for these preferred features and we found evidence in the simulation results that support the claim that these new methods improve the segmentations on certain performance measures. Therefore, we believe that these newly developed methods provide a promising new direction towards the solution of the problem of segmenting wireless street light networks.
Appendix A

Examples monotonicity

Figure A.1: Two examples of clusters constructed by Spielman-Teng in which the first $|S^i|$ elements of $S^{i+1}$ do not coincide with $S^i$. The blue dots represent the vertices in $S^i$. The red dots represent the first $|S^i|$ elements of $S^{i+1}$. 

133
Appendix B

Additional simulation results

B.1 Deployment D

Deployment D is a very large deployment, containing a total of 116952 nodes. However, it consists of a large number of disconnected components. We analyse the three biggest of these here, which are depicted in Figure B.1. These three deployments, denoted by D1, D2 and D3, have size 12479, 15685 and 19448, respectively. This section contains the simulation results for these three deployments, based on the segmentations constructed by the five different algorithms.

Figure B.1: Three components of deployment D.
Figure B.2: Average (a) and maximum (b) delay of a packet in deployment D1 for variable number of segments.

Figure B.3: Average (a) and maximum (b) failure rate of a packet in deployment D1 for variable number of segments.

**B.2 Deployment E**

Deployment E consists of 48456 nodes. Again, this deployment can be divided into multiple connected components. We analyse the two biggest, denoted by E1 and E2. These are depicted in [B.8] and contain 10669 and 33137 nodes, respectively. This section contains the simulation results for these deployments, based on the segmentations constructed by the five different algorithms.
B.2. DEPLOYMENT E

Figure B.4: Average (a) and maximum (b) delay of a packet in deployment D2 for variable number of segments.

Figure B.5: Average (a) and maximum (b) failure rate of a packet in deployment D2 for variable number of segments.

Figure B.6: Average (a) and maximum (b) delay of a packet in deployment D3 for variable number of segments.
Figure B.7: Average (a) and maximum (b) failure rate of a packet in deployment D3 for variable number of segments.
B.2. DEPLOYMENT E

Figure B.8: Two components of deployment E.
Figure B.9: Average (a) and maximum (b) delay of a packet in deployment E1 for variable number of segments.

Figure B.10: Average (a) and maximum (b) failure rate of a packet in deployment E1 for variable number of segments.

Figure B.11: Average (a) and maximum (b) delay of a packet in deployment E2 for variable number of segments.
Figure B.12: Average (a) and maximum (b) failure rate of a packet in deployment E2 for variable number of segments.
Appendix C

List of symbols

\[ \| \cdot \|_{TV} \quad \text{Total variation distance} \]
\[ \partial(S) \quad \text{Cut of set } S \]
\[ [\cdot]_e \quad \text{Truncation operator} \]
\[ \Delta \quad \text{Symmetric difference} \]
\[ \beta \quad \text{Sensing range} \]
\[ \delta(S) \quad \text{Two-sided boundary of set } S \]
\[ \eta \quad \text{Interference range} \]
\[ \lambda_i \quad i^{th} \text{ smallest eigenvalue of } L \]
\[ \mu(S) \quad \text{Volume of set } S \]
\[ \Pi^t \quad \text{Ordering permutation on } r^t \]
\[ \pi \quad \text{Stationary distribution random walk} \]
\[ \sigma \quad \text{Scaling parameter} \]
\[ \sigma_i \quad \text{Local scaling parameter} \]
\[ \tau_{\text{mix}} \quad \tau(1/4) \]
\[ \tau(\epsilon) \quad \epsilon\text{-mixing time} \]
\[ \varphi(S) \quad \text{Conductance of set } S \]
\[ \varphi_G \quad \text{Conductance of graph } G \]
\[ A_G \quad \text{Adjacency matrix of graph } G \]
\[ D \quad \text{Diagonal matrix with degrees} \]
\[ D_W \quad \text{Diagonal matrix with total weights} \]
\[ d(v) \quad \text{Degree of vertex } v \]
\[ d(u,v) \quad \text{Hop distance between vertices } u \text{ and } v \]
\[ E \quad \text{Set of edges} \]
\[ E(S,T) \quad \text{Number of edges between sets } S \text{ and } T \]
\[ E_W(S,T) \quad \text{Total weight between sets } S \text{ and } T \]
\[ G \quad \text{Graph} \]
\[ K(\cdot,\cdot) \quad \text{Transition kernel ESP} \]
\[ \hat{K}(\cdot,\cdot) \quad \text{Transition kernel volume-biased ESP} \]
\[ L \quad \text{Unnormalised Laplacian} \]
\[ L_{\text{sym}} \quad \text{Normalised symmetric Laplacian} \]
\[ M \quad \text{Transition matrix random walk} \]
\[ M_w \quad \text{Transition matrix weight-biased random walk} \]
\[ N(v) \quad \text{Neighbours of vertex } v \]
\[ p(u,v) \quad \text{Transition probability from vertex } u \text{ to vertex } v \]
\[ p(u,S) \quad \text{Transition probability from vertex } u \text{ to set } S \]
\[ p_W(\cdot,\cdot) \quad \text{Transition probability based on weights} \]
\[ p_t^u \quad t\text{-step distribution random walk starting in } u \]
\[ r^t(v) = p^t(v)/d(v) \quad \text{Relative } t\text{-step distribution} \]
\[ R_c \quad \text{Connectivity range} \]
\[ S^t \quad \text{Ordered support set } t\text{-step distribution} \]
\[ S^t_j \quad \text{First } j \text{ elements of } S^t \]
\[ S^t_i \quad t^{th} \text{ set in (volume-biased) ESP} \]
\[ U \quad \text{Uniform}[0,1] \text{ random variable} \]
\[ V \quad \text{Set of vertices} \]
\[ w(u,v) \quad \text{Weight of edge } (u,v) \]
\[ w(v) \quad \text{Total weight of vertex } v \]
Bibliography


