Minimizing Outage Time in an Electricity Network

Reducing the SAIDI of a medium voltage network by relocation of net openings and placement of circuit breakers

Author
Myrte klein Brink BSc

Supervisors
Radboud University
Dr. W. Bosma

Liander N.V.
J. Heres MSc
A.S. Hekker MSc

July 4, 2018

Abstract

The system average interruption duration index, the SAIDI, indicates the expected outage time of medium voltage electricity networks. Circuit breakers and net openings can be placed in an MV network to reduce its SAIDI. The research question of this project is which method can be used to minimize the SAIDI of a network by determining optimal locations for circuit breakers and net openings. Answering this question uses graph theory and optimization methods. The practical problem is first defined as a mathematical optimization problem. Then this thesis gives an innovative algorithm to calculate the SAIDI of a network. The optimization problem is split in two, one part focusing on circuit breakers and one on net openings. The first part is solved by efficiently calculating the possible benefit for every one possibility and using that for a greedy and a branch-and-bound algorithm. The algorithms have been implemented in R and tested on the MV network of Alliander with positive results. For the second part, some useful definitions are given and the suggestion is made to try approximation algorithms, because an acceptable optimization algorithm has not been found. This has not been tried in this project. For the combination of these two devices some ideas are given that could help to solve this problem.

Contents

Chapter 1

Introduction

This thesis deals with reducing power outage at customers of an electricity network. This network is used to supply electric power. A network operator is in charge of the construction and upkeep of this network. When a failure occurs in this network, it can cause outage in parts of the network. This means that some customers are no longer supplied with power. One of the main objectives of the network operator is to reduce the amount of time for which customers experience power outage. This thesis investigates the placement of certain protective measures in the network to help achieve this objective.

1.1 Practical problem

This thesis is the report of a research project for the mathematical master of the Radboud University in Nijmegen. The project was done as an internship at Liander. Liander is a Dutch network operator that takes care of the distribution network of electricity and natural gas in a part of the Netherlands. This thesis focuses on the electricity distribution network. The electricity network in the Netherlands is divided into parts for different levels of voltages, ultra high, high, medium and low voltage. The network of Liander consists mostly of low and medium voltage. The largest contribution to outage time comes from the medium voltage network, as is shown in Figure 1.1. For that reason, Liander wants to implement additional protective devices in this part.

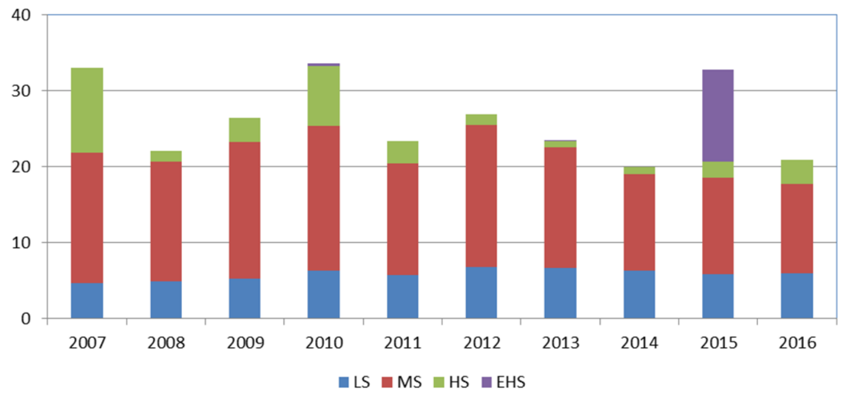


Figure 1.1: Number of minutes of outage per customer for the last ten years in the Netherlands [1] (in Dutch: LS corresponds to low voltage, MS to medium voltage, HS to high voltage and EHS to ultra high voltage)

1.1.1 System average interruption duration index

Figure 1.1 shows the number of minutes of outage per customer over periods of one year. This number can also be estimated for the coming years. This estimation is given by the SAIDI, the system average interruption duration index. This index is used to assess the reliability of the network. The network operator therefore wants this index to be as low as possible. There are several methods to reduce this index. In this thesis the optimal placement of two types of protective devices, namely circuit breakers and net openings, is investigated. These protective devices change the impact, the number of customers experiencing outage, of certain failures. The structure of the network, the SAIDI and the protective devices are further explained in Chapter 2.

1.1.2 Research question

The network operator wants to know what the optimal locations are for these protective devices. In this case optimal means that it minimizes the SAIDI. However, not all solutions are allowed, because of restrictions on the network, but also because implementing these protective measures costs money and takes time. There are several objective functions to consider that minimize the SAIDI or the SAIDI together with the costs; three of those are treated in this thesis.

The research question is the following:
How do you minimize the SAIDI of a given MV network by placing circuit breakers and relocating net openings, given constraints on the structure, the number of circuit breakers in series and the cost and time needed for the implementation?
Alternatively, minimization of the SAIDI together with the cost is investigated.

The first goal of the project is to model this problem and define the objective function and the constraints that a feasible solution has to satisfy. After this optimization problem is defined, the goal is to design algorithms to solve it. These algorithms can be implemented and tested on the network of Liander.

1.1.3 Implementation

Many of the algorithms defined in this thesis have been implemented and tested on parts of the network of Liander. Two of these test cases are introduced in Chapter 2. The implementation is done in R [2], using the igraph package [3]. The execution times of these algorithms have also been documented in this thesis.

1.2 Chapter overview

The practical problem that has been introduced in this chapter is further explained in Chapter 2. All necessary information is contained there.

In Chapter 3 this practical problem is translated to a mathematical optimization problem. This translation models the electricity network as a graph and the allocation of the protective devices as an adjustment of this graph. The objective function and constraints are defined on this adjustment.

In order to solve the optimization problem, first the objective function is investigated. Chapter 4 gives an important definition and several algorithms to calculate the value of the objective function.

The solutions of the optimization problem give the locations of net openings and circuit breakers. In this thesis those two measures are treated separately. Chapter 5 treats the placement of circuit breakers. This problem is first solved for the case where only one addition is allowed. Then several algorithms are given to find the (approximate) optimal locations for multiple circuit breakers, including a greedy and a branch and bound algorithm. The resulting solution has to satisfy the constraints. In this chapter several methods are discussed to eventually obtain the (approximate) optimal feasible solution. In the last section of this chapter alternative objective functions are minimized. These are the functions that minimize the SAIDI together with the restricted costs. The algorithms defined before in this chapter are modified to solve the alternative optimization problems.

The other part of the optimization problem is the positioning of net openings. This part is discussed in Chapter 6. First the constraints are examined, and it becomes clear that the number of feasible solutions is relatively small. Therefore this chapter focuses on finding the feasible solutions first. Again, the relocation of one opening is examined first, and algorithms are given to solve this problem. Then the problem of finding optimal locations for multiple net openings is treated. It becomes clear that the number of solutions grows very large and the approaches that have been used so far will be inefficient to find an optimal solution. The suggestion is therefore made to try approximation algorithms for this specific problem.

Chapter 7 gives the conclusion of this thesis. It gives the answer to the research question as far as this project can and suggestions on further research. Among these suggestions is a section about the combination of the two protective measures. Finding an optimal combination of additional circuit breakers and relocated net openings has not been solved in this project, but several ideas that could be used for further research are given in this chapter.

Chapter 2

Network background

In this chapter an extensive description of the practical problem is given. All necessary information of the electricity network and the relevant protective devices is contained in this chapter. This information is based on the article of A. Middag [4] and the knowledge available at Liander. The goal in this thesis is to minimize the SAIDI, the expected outage time for customers, so it is explained how an outage occurs and what happens in the network at that time.

The electricity network consists of substations with distribution lines between them. Most substations contain a transformer to change the voltage of the electric power. This is necessary because the power is generated at some generating site and transported along the country at high voltage, to limit the power loss due to resistance[1]. It is however not distributed to customers at high voltage, but first transformed to medium voltage at HV substations, and distributed through MV substations. At these MV substations it can then be transformed to low voltage electric power, which is distributed to the customers through the low voltage network.

2.1 Medium voltage distribution network

This thesis focuses on the medium voltage electricity network, because the failures in that part of the electricity network are the cause of the most outage time [1]. This network distributes medium voltage power, ranging between 2 and 35 kV, from HV substations via MV substations. Almost every MV substation has a transformer and is connected to a low voltage network, where the power is supplied to customers. This is shown in Figure 2.1, where a simplified MV network is given. This MV network is supplied by one HV substation which is connected to the HV network. When focusing only on the MV network, every MV substation is assigned a number of customers, which is the number of customers that are connected through only the LV network to that MV substation. This is

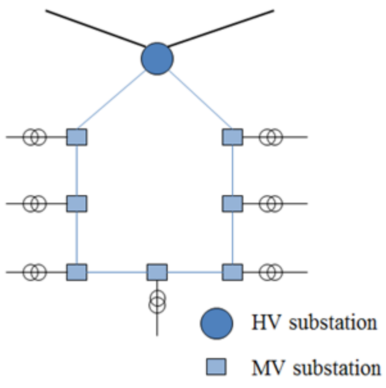


Figure 2.1: Simplified image of MV network

[1]Electric power is calculated as the current times the voltage, while the power loss grows quadratic with the current, so by transporting the same amount of power at high voltage requires a lower current which results in less power loss.

useful when considering failures in the MV network, because this is the number of customers that experience outage if that MV substation is no longer supplied with power.

2.1.1 Structure of the MV network

In Figure 2.1 a network is shown that has a closed circuit or cycle. This is the underlying structure of every MV network, a meshed structure, which is a network that contains these cycles. The underlying structure is the structure that is formed by the substations and distribution lines. That structure allows substations to be supplied with power via different paths. However, there are disadvantages to this structure, because more customers experience outage until the failure is isolated.

That is why so-called net openings can be placed on the MV network. A net opening is a disconnection between a substation and a distribution line, to create an opening in the cycles. An MV network containing net openings can form a network with no closed cycles. This has the benefit that if the underlying structure contains a cycle, there is still the possibility to supply power via another way by closing such a net opening, because the cables are already there. Figure 2.2 shows the same underlying structure, with a net opening to remove the cycle in the network. The net openings will be explained further in Section 2.3.

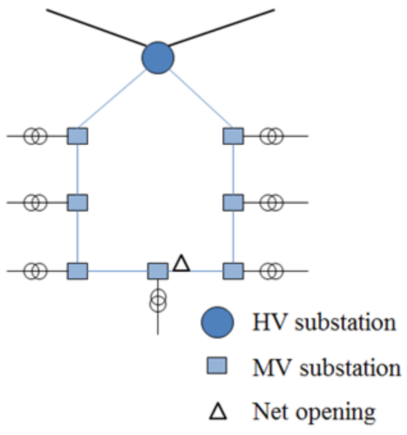


Figure 2.2: Simplified image of MV network with a net opening

Removing the cycles is one of the conditions on a **radial network**. The MV network together with net openings is a **radial network** if it satisfies three conditions:

- There are no cycles in the network.
- Every MV substation is connected to an HV substation, in order to be supplied with power.
- HV substations are not connected to each other through the MV network, because of the same reasons why there are no cycles in the MV network, but also because of physical properties of the electric current.

The net openings are used to create a radial network from the underlying structure. In this thesis we require the network to be radial.

In some special cases, where for instance an important customer is involved, these conditions can be violated. Those cases, where the network is not radial, are disregarded in this thesis.

The MV substations are commonly partitioned into the different branches from the HV substation, which are called **routes**. Every distribution line from an HV substation defines a route. Every MV substation that is supplied through this line is part of the same route. This is possible because of the radial structure of the network; every MV substation is supplied by exactly one HV substation and can only be supplied through one of the distribution lines from that substation.

2.1.2 Medium voltage network of Liander

Liander is the network operator of a large area of the Netherlands. This network consists of over 60.000 distribution lines and more than 45.000 substations. This network is supplied by 365 HV substations, which divide the whole area into smaller parts, although they can be connected in the underlying structure which is interrupted by net openings. The average number of routes from an HV substation is 11 to 12, with around 15 lines per route.

2.2 Expected outage time

The objective that we want to minimize in this thesis is the outage time for customers, the amount of time in which they have no power. Recall that the SAIDI gives the expected outage time of a network. The index depends on the number of customers that experience outage caused by failures over some period of time. There are several ways to reduce the value of this index. The average duration of failures can be reduced by resupplying certain customers with power faster, or the number of failures can be reduced. This thesis focuses on the third option, reducing the number of customers experiencing outage for certain failures, by protecting some customers from those failures.

The failures that are considered when calculating the SAIDI of a network are the failures that occur in distribution lines, because these cover almost all of the failures in the MV network. When a distribution line fails, like the one in Figure 2.3, a protective device is triggered to isolate the failure and prevent the outage to cover larger areas. The device cuts off the power to the part of the network containing the failure. This protective device is called a circuit breaker and is located between a substation and a distribution line. Circuit breakers are always present between the HV substations and all adjacent distribution lines. They can also occur at other locations in the MV network, this will be explained in Section 2.3. When a fault occurs in some line, then the circuit breaker is triggered. This circuit breaker closes off the part of the network behind it. All customers on the LV networks that are behind the red lines are now experiencing outage. The figure also clearly shows the usefulness of the net opening at the time of a failure, otherwise the entire network would be closed off from both sides.

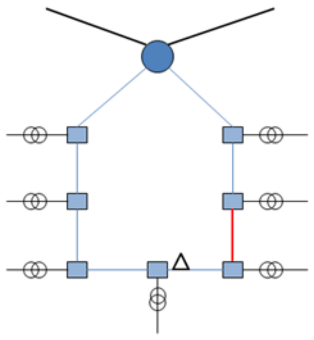


Figure 2.3: Simplified MV network with a failure in the line in red

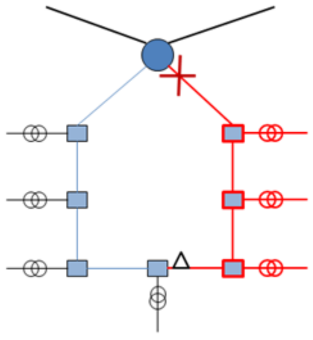


Figure 2.4: MV network with triggered circuit breaker, all lines, stations and LV networks without power indicated in red

System Average Interruption Duration Index

The outage time for all customers is indicated by the SAIDI. The index gives the cumulative expected outage time in minutes for every customer. This means that if two customers are expected to experience outage for six minutes caused by the same failure, that it adds twelve minutes to the expected outage time. The SAIDI is calculated by determining the number of customers that experience outage for every failure times the duration of that failure, which is multiplied with the expected frequency of that failure.

The value of the SAIDI for an MV network N is given by the following formula:

$$\mathrm{SAIDI}(N) = t \cdot \sum_{\text{distribution line } e} f_e \cdot I_e$$

where t is the average outage duration, f_e is the failure frequency of line e and I_e is the impact of line e. The impact of a line is the number of customers that experience outage if that line fails.

The duration of an outage depends on many different factors: the time it takes to observe a failure, to drive to the approximate location of the failure, to find the exact location and to take the steps to deal with the failure itself. Because the duration depends on so many variables, the average duration of a failure is taken as the duration for all failures.

2.2.1 Failure frequency

For every distribution line the expected failure frequency has been calculated. These data are part of the input of the optimization problem. The calculation of the expected failure frequency of a distribution line depends on many factors. A distribution line consists of multiple components, and the age, materials and quantity or length of these components are all taken into account to determine the expected failure frequency with as much precision as possible. The failure of a distribution line can be caused not only by failure of the components, but also by external factors. The failures caused by construction work in the ground where the distribution line could be damaged are also taken into account.

2.2.2 Impact

The impact of a failure is the number of customers experiencing outage caused by that failure. A failure in the MV network stops the supply of power to certain MV substations as is shown in Figure 2.4. The impact of the failure is calculated by taking the sum of the customer numbers of those MV substations. This impact indicates how many customers experience outage directly at the time of the failure.

2.2.3 Costs

The SAIDI gives an indication for the reliability of the network. A low SAIDI indicates a high reliability, because it means that the outage time at customers is low. Naturally, the network operator wants a high reliability. Every minute of outage for the customers therefore has a cost associated with it. This cost can be used to simultaneously minimize the cost of SAIDI and the cost of implementing protective measures.

2.3 Protective measures

The measures that can be used in this problem to reduce the expected SAIDI are explained in this section. These measures protect parts of the network when certain distribution lines fail. They are therefore called protective devices. They can reduce the expected SAIDI by protecting customers in parts of the network, and thereby changing the impact, for certain failures. This thesis covers two of them:

- Circuit breakers
- Net openings

Implementing these protective measures in the network has a cost per measure, for materials and for mechanics. The time it takes mechanics to implement the measure is also important, because even when there is enough money to pay for certain implementations, there is no infinite amount of time or mechanics available.

2.3.1 Circuit breakers

Circuit breakers are present at every HV substation as a protection. This was explained using Figure 2.4. Circuit breakers are automatically triggered after measuring a fault for a certain short amount of time. Triggering means that they disconnect the connection between the substation and the cable. This causes the power to be supplied to the substations before the circuit breaker, but not after that. This protection isolates the faulty distribution line from the network. When it is triggered the whole part of the MV network that was supplied through the distribution line with the triggered circuit breaker is no longer supplied with power. The number of customers experiencing outage at that time is therefore the sum of the customer numbers of those MV substations.

Circuit breakers can also be located at MV substations. They function in the same way. If they measure a faulty current for a certain amount of time, they are triggered and isolate part of the network. In Figure 2.6 an extra circuit breaker is placed and the same fault is simulated. This circuit breaker does not only protect the HV substation, but also an MV substation and therefore all customers in the LV network supplied by that substation.

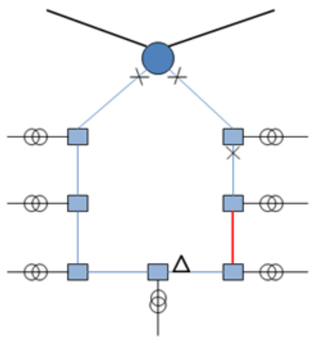


Figure 2.5: Simplified MV network with a failure in the line in red

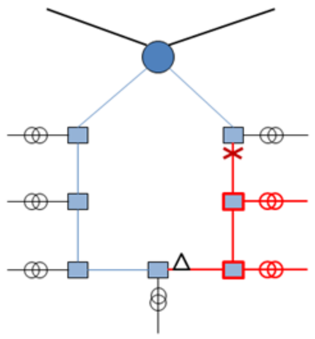


Figure 2.6: MV network with triggered circuit breaker, all lines, stations and LV networks without power indicated in red

This means that the impact of that failure is smaller, fewer customers experience outage if that line fails, which results in a lower expected SAIDI (assuming that the failure frequency of that line is not zero).

A circuit breaker is applied in a substation at one of the ends of a distribution line. In practice, a circuit breaker is always placed after an MV substation, at the beginning of a line. But it is also possible that a circuit breaker is located just before an MV substation.

Limit to the number of circuit breakers

When a line fails, multiple circuit breakers can measure the faulty current, because the supply of power to that line flows through multiple circuit breakers. This is the case in Figure 2.5, both breakers on the path on the right measure the fault. To ensure that the right circuit breaker is triggered, they need to measure the fault for a certain amount of time before triggering. Circuit breakers in series, on the same path to the HV substation, need an increasing amount of time. That ensures that the circuit breaker farthest from the HV substation is triggered first. If the next circuit breaker still measures the fault after that, it is also triggered, until the faulty distribution line is isolated from the network. This prevents unnecessary outage for customers above the triggered circuit breakers.

There is a limit to the amount of time for which a circuit breaker can wait. The waiting time of circuit breakers in series is increasing, therefore the limitation translates to a limit on the number of circuit breakers in series. The agreement is that there can be a maximum of three circuit breakers in series.

2.3.2 Net openings

The net opening was introduced in Figure 2.2. It is a way of creating a radial network from a meshed network. This is done by using net openings to remove all cycles and all paths between HV substations. Simplified, a net opening is the location where a distribution line and a substation are disconnected. Then the power does not go through the line to that substation but only until the net opening. This net opening can only be placed if the substation is also supplied via another path, to ensure that the customers behind that substation still have power.

Net openings can be used to supply power another way when a failure occurs, by disconnecting the faulty distribution line on both sides and then closing the net opening to resupply the customers until the faulty line. This all has to be done on-site. This is the main use of the net openings and the reason why the distribution lines are present even though they are not useful in the normal state of the network without failures.

In this thesis the impact of a failure is determined by the outage directly after the failure occurs, so the main use of net openings will be ignored (besides the effect it has on the average duration of an outage). The way net openings are used as a protective measure with this objective, is the standard locations of these net openings. This is the way they are located in the network when there is no failure. When a failure occurs, it depends on the standard locations of the net openings which MV substations are affected by that outage. The part of the network in which the failure occurs is disconnected, so we want to choose those different parts such that the outage time is minimal.

The locations of the net openings define the paths through which the power is supplied to substations. These locations have to be assigned such that the remaining structure is radial. There should be enough to make sure that there are no more cycles and HV substations are not connected to each other, but every substation still has to be connected to an HV substation. The assignment of locations for net openings and circuit breakers on the network together is called a **configuration** of the network.

2.4 Objective

All information necessary to understand this optimization problem is now given. The optimization problem that is investigated in this thesis is:

Minimize the SAIDI of a given MV network by placing circuit breakers and relocating net openings, given constraints on the structure, the number of circuit breakers in series and the cost and time needed for the implementation.
Alternative objective functions, which use the cost of outage, minimize the SAIDI together with the cost of the implementations. These functions are also investigated in this thesis.

The input of this problem is the network with its current configuration and the bounds on the cost and time needed for the implementation. The current network contains the information of the substations, the topology of the network, the failure frequencies and customer numbers. The network also has a current configuration, with net openings to make it radial and circuit breakers, which are at least located on every line from an HV substation. The configuration of the network can be improved by adding additional circuit breakers and changing the locations of net openings. The possible solutions consist of a list of locations for additional circuit breakers and the locations for all net openings in the network.

2.5 Test cases

The algorithms that are treated in this thesis have been tested on multiple test networks, which are different parts of the network of Liander. In this thesis the results for two of these test networks will be given to show the performance of the implementation on real MV networks.

2.5.1 Case 1: 1 HV substation

This network consists of all substations supplied by one HV substation. The current configuration consists of 416 substations and 65 circuit breakers.

2.5.2 Case 2: 55 HV substations

Network 1 is a subgraph of network 2. This network consists of all stations that are supplied by 55 HV substations, which includes 10954 substations and 2133 circuit breakers.

Chapter 3

Mathematical formulation

In this chapter the translation is made from the practical problem to a mathematical model. An optimization problem always consists of an objective function, the domain of this function and the feasible solutions in this domain. These elements are given in this chapter. The domain, which represents the set of radial configurations, is defined in Sections 3.1 and 3.2, the objective function in Section 3.4 and the feasible solutions in Section 3.5. Section 3.6 treats alternative objective functions, which add minimizing the cost of the solutions to the objective. The last section gives the argumentation for choosing the model as defined in this chapter.

The mathematical model that we use to represent the network in its underlying structure is an undirected graph $G = (V, E)$. The vertices of graph G represent high and medium voltage substations and distribution lines between them, where a set $A \subset V(G)$ indicates the HV substations. The edges of G represent the connections between substations and distribution lines. Configurations are represented by a tuple (G, A, B, O), where G represents the network, A the set of HV substations, B and O the locations of the circuit breakers and the net openings respectively. A configuration can be modeled as a directed graph, which will be called the configuration graph of that configuration. In Section 3.2 the definition of a configuration graph will be given.

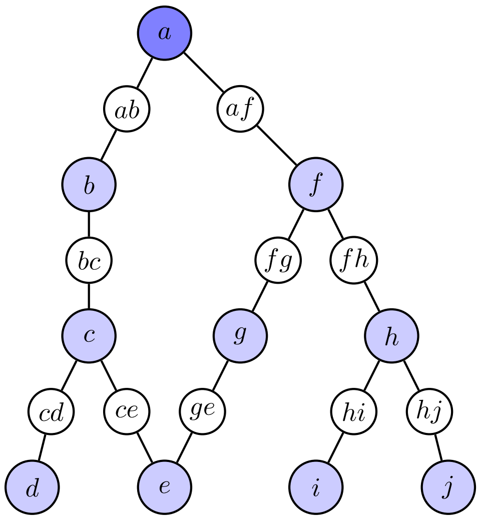


Figure 3.1: Graph G with $A = \{a\}$

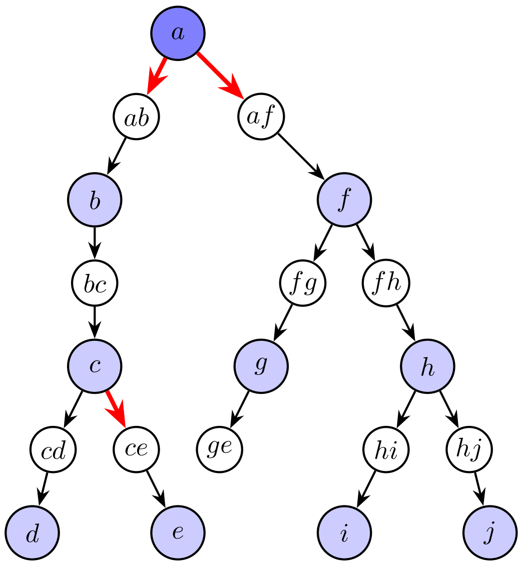


Figure 3.2: Configuration graph of G with $A = \{a\}$, $B = \{\{a, ab\}, \{c, ce\}, \{a, af\}\}$ and $O = \{\{e, ge\}\}$

Example 3.1. The undirected graph in Figure 3.1 is a representation of a network with one HV substation, represented by vertex a, and 9 MV substations. The blue vertices are representations of the substations, the white vertices represent the lines between them. The graph contains a cycle, so a configuration could be with a net opening between vertices e and ge, which means that the net opening is placed between substation e and the line to g. The configuration graph of such a configuration is given in Figure 6.7.

3.1 Graph of the network

In this section the model of all information of the underlying structure of the network is given. The undirected graph G represents the network in its underlying structure as it was explained in Section 2.1.1. The choice was made to model the distribution lines also as vertices of G, because both circuit breakers and net openings are placed between a substation and a distribution line. By modelling both the substations and lines as vertices, the locations of the circuit breakers and net openings are easily represented as edges of G. In Section 3.7 the more straightforward model is explained and the arguments for choosing this *'lines as vertices'* model are given.

The vertices of G can be partitioned into a set V_S representing substations and a set V_D representing distribution lines. In this particular model, the vertices are named in such a way that it is clear which vertices represent distribution lines, but also which two vertices are its neighbors. This is clear in Figure 3.1. It is also possible to add attributes to the vertices to store this information. Whatever method is chosen, it is necessary that the vertices representing lines can be identified and with which vertices these are connected.

A graph G representing the network has to satisfy certain properties. Connections in the practical network are always between one substation and one distribution line, so we know that G has the following properties:

- All vertices in V_D have degree 2.
- G is a bipartite graph where the two disjoint sets are V_S and V_D.

There are no further restrictions on the graph G, which is not necessarily simple, connected or a forest. The restrictions on the structure of the network are the radiality conditions on the configuration, not on the underlying structure. But we will see later that in order for G to allow radial configurations, it has to satisfy certain properties.

The network contains more information than only the topology, circuit breakers and net openings. The expected SAIDI of the network is calculated with the number of customers supplied by the MV substations and the failure frequency of the distribution lines. This information will be represented by weights and values on the vertices of graph G. We define them as functions on the vertices of G. We have the values of vertices, $v : V(G) \to \mathbb{N}$, which are equal to the number of customers supplied by the MV substations they represent. For all vertices that don't represent an MV substation, the value is equal to 0. Similarly the weight is defined, $w : V(G) \to \mathbb{R}_{\geq 0}$. For the vertices representing distribution lines this is equal to their failure frequency, for all other vertices it is 0.

G and A now contain the following information of the network:

- $V(G)$, representing substations and distribution lines
- $V_S \subset V(G)$, representing substations
- $V_D \subset V(G)$, representing distribution lines
- $E(G)$, representing connections between substations and distribution lines
- $A \subset V_S \subset V(G)$, representing HV substations, which are called **roots**
- $w : V(G) \to \mathbb{R}_{\geq 0}$, representing failure frequency of the lines

- $v : V(G) \to \mathbb{N}$, representing number of customers supplied by the substations

For the weights and values the following holds:

- $\forall i \in A : v(i) = 0 \land w(i) = 0$
- $\forall i \in V_S : w(i) = 0$
- $\forall i \in V_D : v(i) = 0$

In this thesis we will call functions on the vertices or the edges of the graph **attributes** and will also use the notation w_i or v_i instead of $w(i)$ or $v(i)$.

Example 3.2. Suppose G is the graph depicted in Figure 3.3. The vertices represent substations and lines, the names of the vertices indicate whether it represents a distribution line and which substations that line connects. The following table gives the values and weights of some vertices.

vertex	**value**	**weight**
a	0	0
ab	0	.0025
b	10	0
bc	0	.0007
c	300	0
cd	0	.003
d	150	0
ce	0	.0018
e	20	0
$\cdots$	$\cdots$	$\cdots$

Table 3.1: Table of $V(G)$ with its values and weights

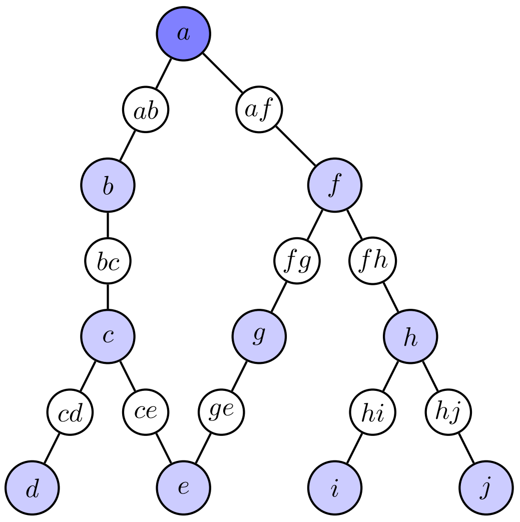


Figure 3.3: Graph G with $A = \{a\}$

This information is fixed for an instance of the optimization problem; $G = (V, E)$, A and the weights and values are given and not allowed to be changed. In the next section the mathematical translation of a configuration of the network will be explained. The feasible solutions of the optimization problem will be the different configurations of the given graph G with roots A.

3.2 Configuration of the graph

The domain of the objective function will be given in this section. In the practical problem, the SAIDI is calculated over the configuration of a network. In the model, the representation of configurations will be the domain of the objective function. A configuration of the network was defined in the previous chapter as the assignment of net openings and circuit breakers on the network. This configuration was only allowed if it satisfied certain radiality constraints. In the mathematical model, a configuration is defined by giving the sets B and O representing the locations of the circuit breakers and net openings. The radiality conditions are also translated for this model.

Definition 3.3. A **configuration** of the graph G with a given subset $A \subset V(G)$ is a tuple $C = (G, A, B, O)$, where $B, O \subset E(G)$. B is called the set of **breaker edges**, and O the set of **openings**.

Assumption 3.4. For every configuration $C = (G, A, B, O)$ we assume that all edges incident to vertices of A in G are elements of B.

3.2.1 Undirected configuration graph

This configuration can again be modeled by a graph G_C, which we will call the **configuration graph** of configuration $C = (G, A, B, O)$. This graph is a equal to graph G with the following adjustments:

- The attribute *root* is assigned to the vertices $V(G_C)$, which indicates whether or not the vertex is an element of A.
- The attribute *breaker* is assigned to the edges $E(G_C)$, indicating whether or not the edge is an element of B.
- Creating an opening for every edge in O, by removing that edge from the graph.

With these adjustments all necessary information of the configuration can be obtained from that graph G_C. The configuration graph G_C has an orientation which will be given in the next subsection, for now the undirected configuration graph is used.

By creating the net opening this way, it is possible to translate the configuration graph G_C back to the graph G. This is done by finding the vertices representing distribution lines and adding missing edges to the vertices that should be its neighbors in G, until all of these vertices have degree 2. This is possible because this information is in some way saved on these vertices. In this model it is clear from the names of the vertices.

Example 3.5. Let G be the graph from Figure 3.3 with $A = \{a\}$ and the weights and values given in Table 5.1. Let C be the configuration with breaker edges $\{a, ab\}, \{c, ce\}$ and $\{a, af\}$ and an opening between e and ge. Figure 3.4 shows the undirected configuration graph of C, with $A \subset V(G)$ darker blue, $B \subset E(G)$ in red and bold and the creation of an opening between e and ge.

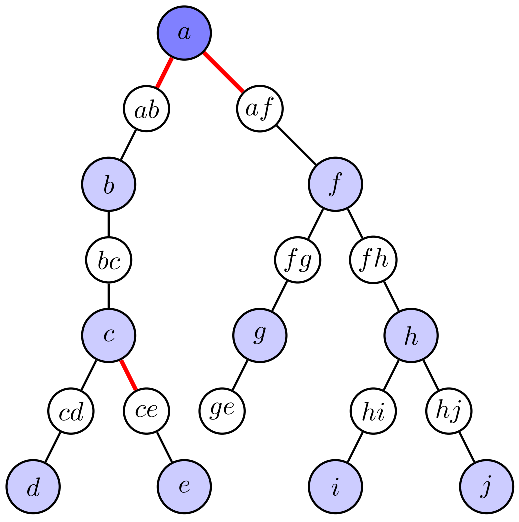


Figure 3.4: Undirected configuration graph of $C = (G, \{a\}, \{\{a, ab\}, \{c, ce\}, \{a, af\}\}, \{\{e, ge\}\})$

3.2.2 Radiality

We have assumed that the medium voltage network is configured in such a way that it is a radial network. In this subsection we will define radiality on the mathematical model. The background in graph theory can be found in the book by Bondy and Murty [5], in Chapters 1 to 4.

Let $A \subset V$ represent the high voltage substations. The constraints of a radial network configuration in the practical problem were given in the previous chapter; the network does not contain closed circuits and every substation is connected to exactly one high voltage substation. This is translated to the model via the following constraints:

- The graph is a forest[1], which means that the graph does not contain cycles.
- Every component[2] of the graph contains exactly one vertex from the set A.

Definition 3.6. If a graph satisfies these constraints for set $A \subset V$, then it is said to be ***A*-radial**.

We see that the radiality assumption from Chapter 2 translates to the condition that the configuration graph has to be A-radial for the configuration to be in the domain.

By adding openings to a graph G, it is always possible to create some configuration graph which is a forest. The second property is not always satisfiable by any configuration, because A cannot change and no edges are added. If G already contains a component without a vertex in A, then no configuration can add a vertex of A to this component. With the following assumption on G it is always be possible to create A-radial configurations.

Assumption 3.7. The graph G contains at least one vertex from A in every component.

An A-radial graph is a forest, a disjoint union of trees, with every tree containing exactly one vertex from A. This vertex represents the HV substation which supplies the other substations. This intuitively defines a orientation on the edges, namely all edges direct away from the vertex from A, which we can therefore call the root of the tree. A graph with such an orientation is said to be a rooted forest.

Definition 3.8. A **rooted forest** is a disjoint union of trees, where every tree is a rooted tree. A **rooted tree** is a tree with one vertex assigned as root and all edges implicitly directing away from the root.

Creating a rooted forest from an undirected forest in this way is possible because there is a unique path between every two vertices within a tree. This is because it is connected and does not contain cycles. So the orientation is created by following the unique paths from the root to all the leaves[3]. It is clear that the root has no incoming edges. The in-degree, which is the number of incoming edges, of all other vertices is 1, otherwise there would be a cycle in the graph[4]. The out-degree of vertices in V_D, the ones representing distribution lines, is 1. There is no limit on the out-degree of vertices in V_S. It is useful to work with a rooted forest because it makes it easier to look at paths between vertices with respect to the root. Therefore we will from now on consider the configuration graph as a directed graph with this orientation on the edges.

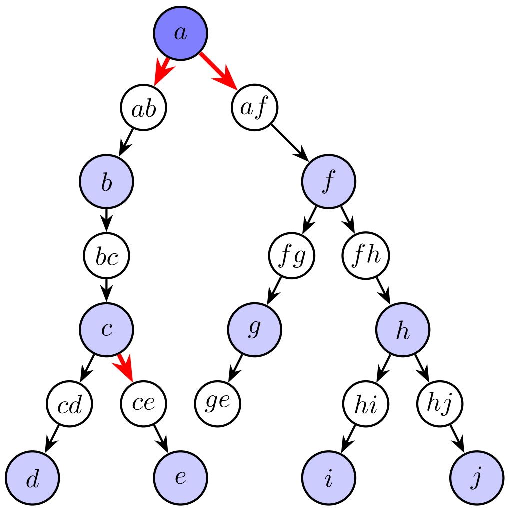


Figure 3.5: Configuration graph

[1]A forest is a disjoint union of trees, which are connected graphs without cycles.

[2]A component is a maximal connected subgraph, which means that all vertices in the subgraph have a path between them, and all other vertices from the original graph are not connected to any vertex in the subgraph.

[3]A leaf is a vertex with degree 1.

[4]If the vertex i has the incoming edge from a path from the root and one from j, then a cycle is formed because there is also a path from the root to j, which means there are two different paths to i.

Example 3.9. The configuration graph of Figure 3.4 is clearly A-radial (it is a tree with exactly one vertex from A), so we can give the described orientation to the edges. This is shown in Figure 3.5.

3.3 Summary of the model

In the previous sections we have defined the mathematical model of the medium voltage network. This section will give a summary of the translation of the practical network to the mathematical model. All essential characteristics of the practical network are listed below, with their representation in the model:

- High voltage and medium voltage substations with distribution lines between them.
  Represented by a graph G, substations and distribution lines as vertices $V(G) = V_S \cup V_D$, edges between them if they are connected. The high voltage substations are identified by $A \subset V(G)$.
- The distribution lines all have a failure frequency and the medium voltage substations a number of customers supplied by that substation.
  The vertices have a weight and a value assigned to them. These are equal to 0 for all vertices in A, the weights are 0 for elements of V_S and values are 0 for elements of V_D.
- A configuration of the network contains circuit breakers and net openings.
  A configuration of G is given by a tuple $C = (G, A, B, O)$, where $B \subset E(G)$ is the set of breaker edges and $O \subset E(G)$ the set of openings.
- All HV substations have a circuit breaker on all outgoing distribution lines.
  For every configuration $C = (G, A, B, O)$ we assume that B contains all outgoing edges from vertices in A.
- The network configuration is assumed to be radial and the medium voltage substations are supplied by a high voltage substation.
  The configuration graph G_C of C has to be A-radial and is oriented by directing the edges away from the roots $A \subset V(G_C)$.
- Every distribution line connected to an HV substation defines a route, which consists of all MV substations that are supplied through that line.
  Routes correspond to branches[5] of G_C.

All characteristics of the network essential to the problem are represented in the model. The remainder of this chapter gives the mathematical formulation of the optimization problem over this model.

3.4 Objective function

The objective function of this problem is the SAIDI (System Average Interruption Duration Index), which is the expected amount of time for which customers experience outage.

[5]A branch of a tree is a subgraph consisting of an outgoing edge from the root and all edges and vertices reachable from this edge

Suppose $C = (G, A, B, O)$ is an A-radial configuration of the graph G. Define I_j to be the **impact** of a vertex $j \in V_D$, this is equal to the number of customers experiencing outage if the line represented by vertex j fails. The SAIDI is then defined as the following formula:

$$\text{SAIDI}(C) = t \cdot \sum_{j \in V_D(G_C)} (w_j \cdot I_j)$$

The practical meaning of the result of this calculation is the expected total number of minutes of outage per period.

The result of this function depends on the configuration of the network. The aim is to find an optimal feasible configuration, with minimized expected SAIDI.

Impact of a vertex

In the previous chapter it is explained which MV substations are not supplied with power if a failure occurs in a certain distribution line. The number of customers without power is then equal to the sum of customers supplied by these substations. The first circuit breaker towards the HV substation is triggered when a failure occurs and every MV substation behind this circuit breaker is affected. In the model, this is the first breaker edge when tracing towards the root. So the impact of an edge is the sum of values of all vertices representing MV substations behind the first breaker edge towards a root.

The definition of I_j of a vertex j, given that e is the first breaker edge on the unique path from j to a root, is given by $I_j = \sum_{\substack{i \in V_S(G) \\ i \text{ after } e}} v_i$.

Example 3.10. We continue with the example that has G as the graph from Figure 3.5 with $A = \{a\}$, the weights and values given in Table 5.1. C is the configuration with breaker edges $B = \{\{a, ab\}, \{c, ce\}, \{a, af\}, \{f, fh\}\}$ and an opening between e and ge, $O = \{e, ge\}$.

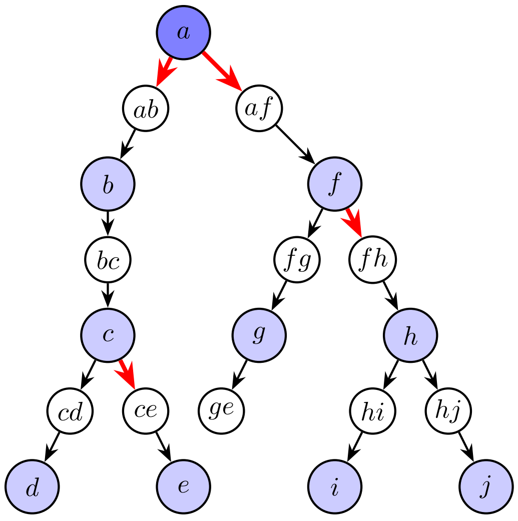


Figure 3.6: Configuration graph of configuration C

Suppose we want to know the impact of vertex hi. The impact represents the number of customers experiencing outage if a failure occurs in the line represented by that vertex. When a failure occurs, the closest circuit breaker on the path from the HV substation to the edge is triggered. In this case the closest breaker edge is edge $\{f, fh\}$. Every substation that is supplied through this edge will be affected. These substations are represented by h, i and j. The impact of hi is therefore equal to $175 + 50 + 310 = 535$. The expected number of customers experiencing outage over the time period caused by failure in hi is then equal to $w_{hi} \cdot I_{hi} = 0.0021 \cdot 535 = 0.7413$.

For every vertex i we know that it is an element of some component of G_C, which is a tree T. This tree has exactly one root a, and in this tree there is a unique path from i to a. That means that the first breaker edge on the path from i to an element of A is well-defined.

3.5 Solution space

We have a model of the practical problem and a formulation of the objective function which we want to minimize. For the optimization problem we now need to define the solution space. The solution space is the set of all feasible solutions. An optimal solution of the optimization problem is a solution from the solution space with the lowest result of the objective function.

3.5.1 Reconfigurations

Every solution is some adjustment of the original configuration $C = (G, A, B, O)$, because in the practical problem we want to find the best locations for additional circuit breakers and other net openings without changing the rest of the network. This means that G, with its weights and values, and A have to stay the same. The two possibilities to adjust the configuration define the solution space.

The first possibility is to place circuit breakers, which means that in the network the set of lines with circuit breakers will be extended. This translates to the model as the possibility to extend $B \subset E(G)$ to some B' with $B \subset B' \subset E(G)$.

The second possibility is to relocate net openings. This means that net openings can be closed or created. In the model this translates to the possibility to choose any $O' \subset E(G)$, as long as the configuration is A-radial.

Combining these two possibilities we see that the solution space, given the current A-radial configuration $C = (G, A, B, O)$, contains all configuration $C' = (G', A', B', O')$ which satisfy the following properties:

- $G' = G$
- $A' = A$
- $B \subset B' \subset E(G)$
- $G_{C'}$ is A-radial

Definition 3.11. A configuration $C' = (G', A', B', O')$ satisfying these properties is called a **reconfiguration** of C.

The feasible solutions are a subset of all possible reconfigurations of C, namely those that satisfy the constraints given in the following subsection.

3.5.2 Constraints

The solution space is limited by certain practical constraints. These were already explained in the previous chapter. The three constraints are on the circuit breakers in series, and on the cost and time needed for the implementation of the solution.

Limit to breaker edges

In the practical case, the number of possible circuit breakers in series is limited to 3. The reason for this is explained in Section 2.3. The constraint for the optimization problem is therefore that every possible path of the configuration graph cannot contain more than 3 edges in B. Another way to state this is the condition that there is no path in the configuration graph with 4 or more edges from B.

Capacity

All changes that are made in practice cost a certain amount of time. The capacity constraint is the maximum amount of time that is available for the reconfiguration of the network. The changes which can be made are the placement of circuit breakers and the relocation of a net opening. For both of these changes we know the amount of time that is needed for one change.

Let T be the maximum total time, T_b the time needed for the placement of a circuit breaker and T_o the time for the closing and opening of a net opening. The **capacity constraint** in the mathematical model is then

$$x \cdot T_b + y \cdot T_o \leq T$$

where x is the number of breaker edges added in the reconfiguration, and y the number of changes in the set of openings. These are calculated by $x = \#(B' \setminus B)$ and $y = \#(O' \setminus O)$.

Costs

Besides the time it takes to make changes to the configuration, it also has a certain cost. Let S be the total amount of cost allowed, S_b the cost of placing a circuit breaker and S_o the cost of closing and opening a net opening. The **cost constraint** is stated as

$$x \cdot S_b + y \cdot S_o \leq S$$

3.6 Alternative objective function

Reducing the SAIDI is the objective of this problem, but sometimes the reduction caused by one extra adjustment is so small that it is not worth the cost and the time that adjustment takes. So even when it is possible by the constraints, it may not be beneficial to add that adjustment. The optimization problem stated so far does not take this into account.

A choice can therefore be made between the reduction of SAIDI and the cost via different objective functions. This choice is subjective to the practical problem and will therefore not be made in this thesis. The alternative objective functions will be explained and considered in some of the following chapters.

Let $S(C')$ be the cost of the reconfiguration, $S(C') = x \cdot S_b + y \cdot S_o$. The options for reducing the SAIDI and the cost at the same time are the following:

a) $\min \mathrm{SAIDI}(C') \cdot p + S(C')$

b) $\min \frac{S(C')}{\mathrm{SAIDI}(C) - \mathrm{SAIDI}(C')}$

a) Assigning a cost to SAIDI

This objective function assigns a cost to the value of the SAIDI, which means that every unit of SAIDI has a cost of p. So if the SAIDI is reduced by n units, then the profit is $p \cdot n$. This is beneficial if the cost of the reconfiguration is less than $p \cdot n$. An optimal solution minimizes the cost of SAIDI added to the cost of the reconfiguration.

b) Minimizing the cost benefit function

With this objective function there is no given bound on the cost per reduction unit, but the minimal cost per reduction unit is found. This objective function therefore gives the best profit for the investment.

Conclusion

All objective functions can be used and have their advantages and disadvantages, and it depends on the practical question which objective should be used. This choice is between the three options:

- There is a given maximum total cost and we want the most reduction as possible.
- We want the most reduction as possible, but only if the benefits of the changes are large enough to be beneficial, given some value p which indicates the profit of reduction in SAIDI.
- We want to get the best reduction for the investments that have to be made.

3.7 Choice of the model

In this section the choice for this model will be explained. A graph shows the connections between certain objects and can be used to trace along these connections, which is for instance used to find the impact of a failing distribution line. Therefore a graph is used to model the network. The choice that had to be made was which objects are represented as vertices in the graph and which of those vertices should have edges between them. The most straightforward, intuitive way is to model the substations of the network as vertices and the distribution lines between them as edges, but this has several disadvantages. In this section the differences between that model and the model described above, with substations and distribution lines represented as vertices, are explained, and so are the reasons why the model described above was chosen.

3.7.1 Straightforward model

The intuitive way to model the underlying structure of the network, which consists of the HV substations, MV substations and distribution lines, is to represent the substations as vertices of the graph and the lines between them as edges. An example is given in Figure 3.7. In this subsection we explore the model starting with this graph G representing the underlying structure.

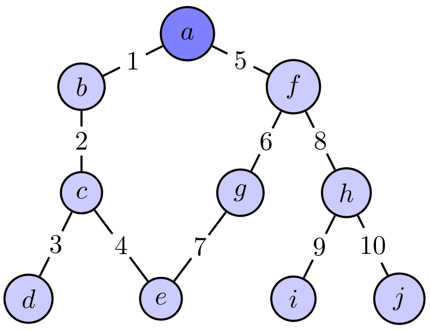


Figure 3.7: Intuitive model

Information of the underlying structure

There is information on the substations and lines that is needed for this objective function, namely the number of customers and the failure frequencies. These are represented by values on the vertices and weights on the edges respectively. The vertices that represent HV substations have value 0. These vertices are identified by a subset $A \subset V(G)$.

Configuration of the graph

A configuration of the network is made by adding circuit breakers and net openings. This is modeled by identifying a subset $B \subset E(G)$ representing circuit breakers, and creating openings in the graph. Figure 3.8 illustrates how a configuration is created from graph G with an opening between vertex e and edge 7. Creating an

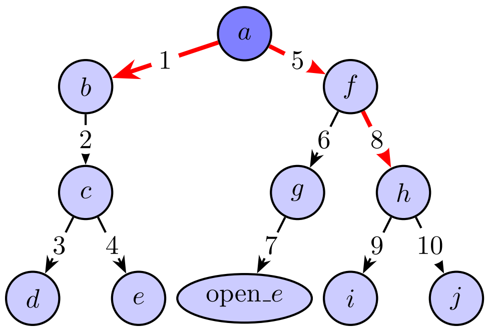


Figure 3.8: Configuration of the graph with $B = \{1, 5, 8\}$ and $O = \{(e, 7)\}$

opening consists of copying the vertex, assigning value 0 to it and a name associated to the original vertex, deleting the edge to the original vertex and adding a similar edge to the copy, with equal weight.

Disadvantages

Creating a configuration this way has several disadvantages:

1. For every opening we need to copy a vertex, delete an edge and add an edge
2. Dealing with multiple openings at the same vertex
3. Circuit breakers can also be placed at the head of an edge instead of the tail

The first disadvantage is that this model is from a graph theoretical perspective not a very nice solution. In the practical network a net opening is simply a disconnection between a line and a substation, while in the graph an opening alters the vertices of the graph and needs several operations. A graph of the configuration is therefore not a subgraph of the graph G, which means that changing the configuration takes more operations.

The second disadvantage deals with the way openings are created in this model. It needs to save in some way the original vertices of the copies, which is done by naming it to correspond to the original vertex. A problem then arises when there are multiple openings adjacent to the same vertex, because then multiple copies will be created that have equal names. This can be solved by adding a number in front of the name.

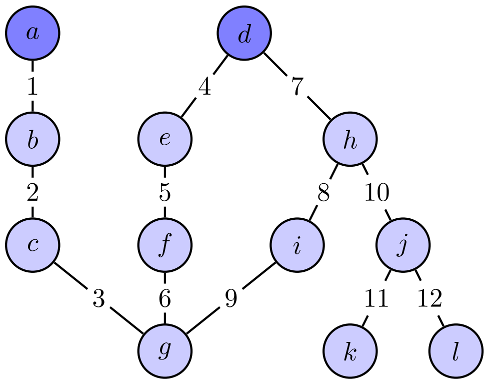


Figure 3.9: Graph G with $A = \{a, d\}$

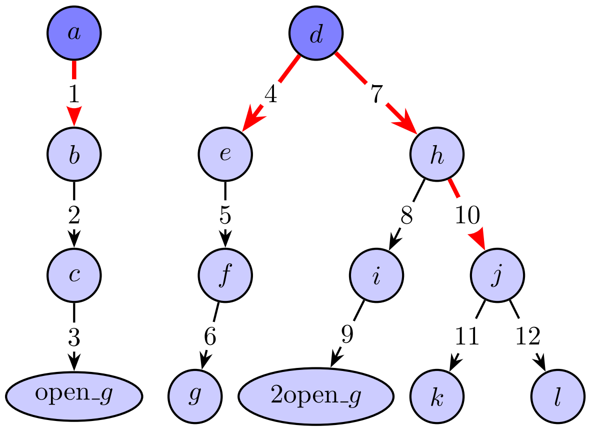


Figure 3.10: Configuration with $B = \{1, 4, 7, 10\}$ and $O = \{(g, 3), (g, 9)\}$)

The representation of circuit breakers so far is not a correct representation. The third disadvantage occurs when trying to solve this. Circuit breakers are physical objects placed between a substation and a distribution line. In practice these are placed after a substation, seen from the HV substation, but a circuit breaker can also be located before a substation. This is usually the case when a net opening has been moved, an example is given in Figures 3.11 and 3.12. The current description of the model does not consider which side of the edge contains the circuit breaker. In the figures it is not clear whether it is located at substation c or at e. This can be solved by saving for every breaker edge at which vertex it is located.

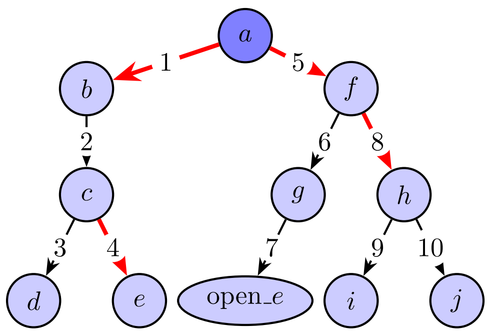


Figure 3.11: Configuration graph of configuration $(G, \{a\}, \{1, 4, 5, 8\}, \{(e, 7)\})$

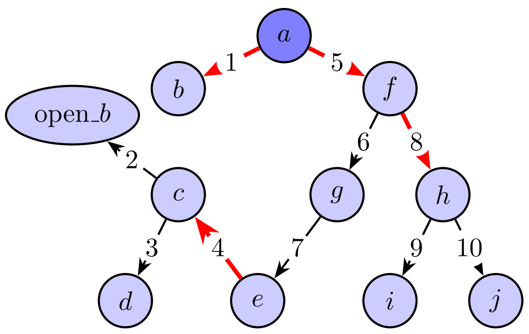


Figure 3.12: Configuration graph of configuration $(G, \{a\}, \{1, 4, 5, 8\}, \{(b, 2)\})$

It is clear that all of these disadvantages can be solved within this model, but there is a better solution available. The fact that circuit breakers and net openings are both located between a substation and a distribution line is a reason to model these connections as edges. In the model described in the other sections of this chapter, the substations and distribution lines are both modeled as vertices of the graph, with edges between them if they are connected in the network.

Of course, this model also has one clear disadvantage, namely that the size of the graph is almost doubled. It is however never more than a factor 2, which means that it doesn't worsen the complexity much. We can overcome this one disadvantage if it solves the ones mentioned above.

The *'lines as vertices'* model allows us to represent both the circuit breakers and the net openings as subsets of the edges. Moreover, creating an opening now only consists of deleting one edge from the graph. These are the reasons why we have chosen for that model.

Chapter 4

Calculating the objective function

In this chapter various methods are given for calculating the objective function. This can be done by simply using the definition, but then some of the same results are calculated multiple times. In an optimization problem, the objective function is calculated for multiple solutions. For practical purposes, it is therefore interesting to investigate if this function can be calculated more efficiently. The result of this investigation is given in this chapter.

The domain of the objective function is the set of radial configurations as was defined in the previous chapter. In this chapter we assume that every graph we consider is a configuration graph representing a configuration of the network. So this is a radial graph with weights and values on the vertices, a set of breaker edges, including all edges from the roots, and a set of net openings.

4.1 Objective function

The objective function that we want to minimize in this optimization problem is the SAIDI. In the model, the SAIDI of a graph G is defined by the following formula:

$$\text{SAIDI}(G) = t \cdot \sum_{j \in V_D(G)} (w_j \cdot I_j)$$

where t is a constant, w_j is the weight of vertex j and I_j is the impact of vertex j.

In this minimization problem, there are no negative values. Therefore we don't need to look at the whole formula, because t is a non-negative constant factor. The following lemma shows that we can simplify the objective function when searching for an optimal solution.

Lemma 4.1. *Let $c > 0$, $f : A \to \mathbb{N}$. If the minimum of $c \cdot f(a)$ is obtained at a', say $c \cdot f(a') = \min_{a \in A} c \cdot f(a)$, then the minimum of $f(a)$ is also obtained at a', $f(a') = \min_{a \in A} f(a)$.*

We are interested in the solution that has the smallest SAIDI, so we can find the solution where $\sum_{j \in V_D(G)} (w_j \cdot I_j)$ is minimized and use this lemma to conclude that this is the solution we are looking for. After this solution is found, it is easy to calculate, from this new objective function, the SAIDI of the solution. So from now on we will work with the objective function:

$$R(G) = \sum_{j \in V_D(G)} (w_j \cdot I_j)$$

The following lemma shows that the vertices in V_D do not necessarily need to be identified.

Lemma 4.2. $R(G) = \sum_{j \in V(G)} (w_j \cdot I_j)$.

Proof. We have $V(G) = V_D \cup V_S$, $V_S \cap V_D = \emptyset$ and $\forall j \in V_S : w_j = 0$, so we see that

$$R(G) = \sum_{j \in V_D(G)} (w_j \cdot I_j) = \sum_{j \in V_D(G)} (w_j \cdot I_j) + 0 = \sum_{j \in V_D(G)} (w_j \cdot I_j) + \sum_{j \in V_S(G)} (w_j \cdot I_j) = \sum_{j \in V(G)} (w_j \cdot I_j)$$

□

The weight w_j is known for every vertex j. I_j is defined as the sum of the values of certain vertices. So any difficulty in the calculation can only be in the calculation of the impact values. More specifically, the impact of j is determined by the first breaker edge e on the unique path from j towards a root. I_j is then the sum of the values of all vertices that lie after e, $I_j = \sum_{\substack{i \in V_S(G) \\ i \text{ after } e}} v_i$.

Of course, for the impact we have a similar lemma, because the values of vertices in V_D is 0.

Lemma 4.3. $I_j = \sum_{\substack{i \in V(G) \\ i \textit{ after } e}} v_i$.

So to find the impact of a vertex $j \in V(G)$ we have to trace back to the root and stop at the first breaker edge, then trace the other way from that breaker edge to all reachable leaves and add all the values of the vertices that we see.

The objective function can be implemented that way, by tracing for every edge, but in the following section we will show that this will execute some of the same calculations multiple times. This will therefore be one of the slowest methods to calculate $R(G)$. In the following sections more efficient methods will be given.

4.2 Dividing the graph in segments

Because the impact of every vertex is defined by what breaker edge lies closest to them on the path to a root, it is clear that the impact will be equal for several vertices, namely all those that have the same closest breaker edge towards the root. These are all the vertices from this breaker edge, but only until other breaker edges.

Example 4.4. Suppose we have the graph of Figure 4.1. In this figure it is clear that for instance vertex ab and bc have the same closest breaker edge towards the root. That means that the impact of these vertices is the same, the sum of values of all vertices after breaker edge $\{a, ab\}$. This is not the case for af and fh for example, because $\{f, fh\}$ is also a breaker edge. In this example we see that all vertices from $\{f, fh\}$, namely fh, hi and hj, have the same impact (the sum of the values of h, i and j) and that af, fg and ge also all have the same impact (the sum of the values of f, g, h, i and j).

This fact, that the graph can be divided into areas with the same impact, gives rise to the following definition. [6]

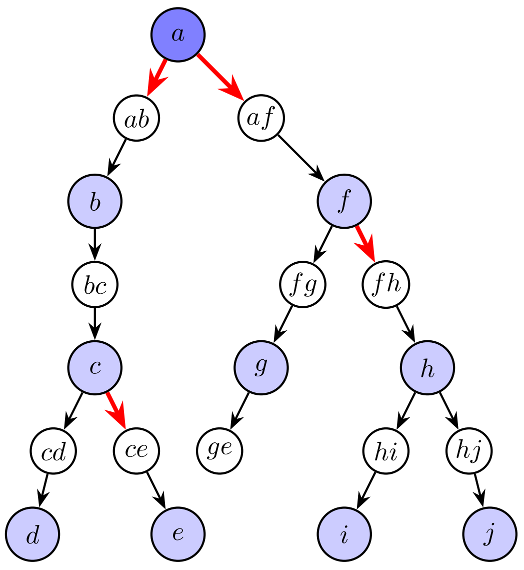


Figure 4.1: Configuration graph G

Definition 4.5. The collection of all edges and vertices from a breaker edge e, including e, until and excluding other breaker edges is called a **segment**.

So the graph can be divided into segments defined by the breaker edges. This division has no overlap and covers almost the whole graph. The roots are the only elements not contained in any segment, because of the assumption that all outgoing edges from roots are breaker edges.

Example 4.6. Figure 4.2 shows the division into segments in the tree from the previous example.

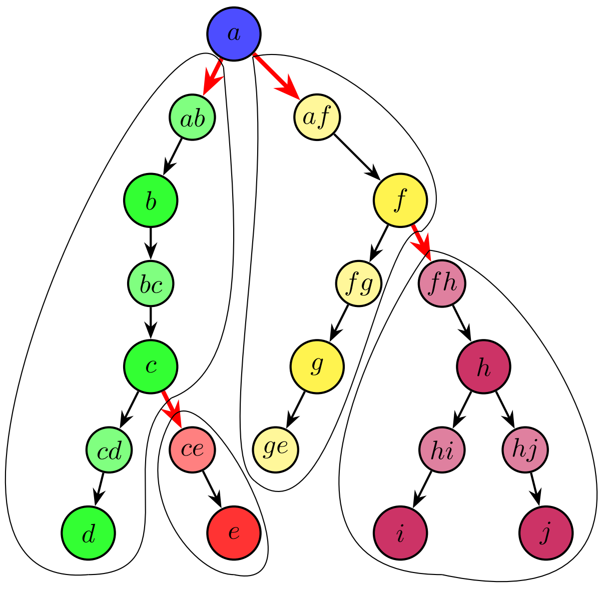


Figure 4.2: Configuration graph G with circled segments

We have seen that the impacts of the vertices in the same segment are equal. We will also call this the impact of the segment. This means we can calculate $\sum_{j \in V(G)} (w_j \cdot I_j)$ per segment, by taking the sum of the weights inside the segment times the impact. This holds because

$$\sum_{j \in \text{ segment } i} (w_j \cdot I_j) = \sum_j (w_j \cdot I_i) = (\sum_j w_j) \cdot I_i$$

where I_i is the impact of segment i.

Every element of $V(G) \setminus A$ is included in exactly one segment (because every vertex has a unique closest breaker edge towards the root, apart from the roots themselves). We can use this property, together with the fact that the weights of the roots are 0, to calculate $R(G)$ using segments. This is done by

$$R(G) = \sum_{j \in V(G)} (w_j \cdot I_j) = \sum_{j \in V(G) \setminus A} (w_j \cdot I_j) = \sum_{\text{segment } i} I_i \cdot (\sum_{j \in \text{segment } i} w_j).$$

In the next section this property will be used in the algorithms for calculating $R(G)$.

4.3 Algorithms

There are multiple methods to calculate the value of $R(G)$ for a radial graph G. The simplest way is to calculate the impact for every edge and multiplying this with its weight, but this will not be very fast. Other methods make use of the segments defined in the previous section, to avoid calculating the same impact values multiple times.

In this section we treat two different types of algorithms to calculate $R(G)$. The first is a way to calculate it directly using the definition of segments. The second consists of an initialization, where we define functions or attributes on the edges of the graphs, and an algorithm to use this initialization to quickly calculate the desired result.

4.3.1 Decomposing the graph

In this subsection we use the definition of segments by identifying the segments and calculating their impact. For the calculation of the impact, we want to identify the segments of the graph. In this part we look at a way to decompose the graph into segments. The segments are clearly defined by breaker edges, so we can choose between tracing from every breaker edge or deleting all breaker edges and identifying the connected components. Identifying every segment alone is not enough, because then we still need to calculate which impact value should be assigned to that segment. For this reason we choose to focus on decomposing the graph in which the edges in B have been deleted. Decomposing means dividing the graph into its components. It can be used to identify the segments and to calculate the impact.

The first thing we want to do in the algorithm is to delete the breaker edges and decompose the remaining graph into connected components. Then every component relates to a segment, except for the component consisting only of a root. For every component we determine which breaker edge corresponds to it. Because we have only deleted breaker edges, we know that every edge connecting these components in the original graph is a breaker edge. Besides that, we also know that there can only be one incoming breaker edge, which is exactly the breaker edge that corresponds to the component. So for every component we will find the only vertex with no incoming edges in the component and identify the incoming edge in the original graph.

Example 4.7. Suppose we have the same graph as before (Figure 4.1) and perform the deletion on it, as shown in Figure 4.3. In the graph we have 5 maximal connected components, including the root. The other 4 components correspond to the 4 segments that are circled in Figure 4.2. We see that these components all have exactly one incoming edge in G, which is the breaker edge corresponding to the segment.

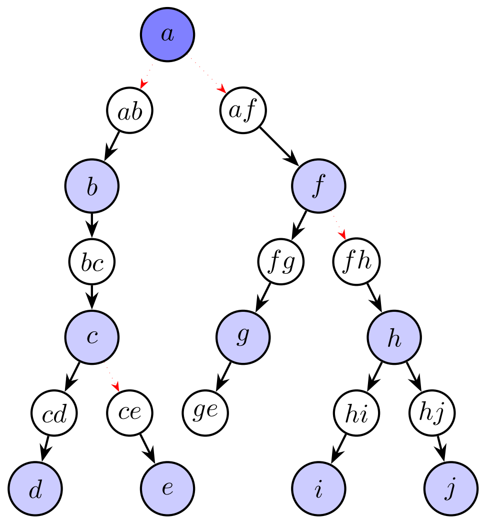


Figure 4.3: configuration graph G with breaker edges deleted/dotted

So for every component we can find the corresponding breaker edge e'. After that we want to calculate the impact of that segment. The impact is the sum of the values of all vertices after that breaker edge e'. This can contain vertices of other segments. We look at the original graph and identify the set of vertices after e'. This is the out-tree of the head[1] of e', the set of vertices that can be reached using only outgoing edges. The igraph package contains the function `subcomponent`, which does exactly this. It finds for a given graph and vertex with `mode = "out"` all vertices reachable from the given vertex through directed paths. It is easy to take the sum of the values of those vertices.

Now we have identified the segments, so we can sum over the weights of the vertices in the segment, and the impact for that segment. $R(G)$ is now equal to the sum of the multiplication of those values for every segment.

Algorithm 4.1 gives the pseudocode for the described method.

Algorithm 4.1: Calculate $R(G)$ directly by deleting breaker edges and decomposing

```
Data: Directed graph $G$ with values and weights on vertices and the attributes *root* and *breaker*
Result: $R(G)$
1  $H$ = delete breaker edges from $G$;
2  components = decompose $H$ into maximal connected subgraphs;
3  result = 0;
4  for $i$ in 1 : #components do
5      if components[$i$] does not contain a root then
6          $W$ = sum of weights of vertices in components[$i$];
7          topnode = node of components[$i$] with no incoming edges;
8          reachableSet = out-tree of $G$ starting at topnode;
9          $I$ = sum of values of reachableSet;
10         result = result + $W \cdot I$;
11     end
12 end
13 return result;
```

4.3.2 Summing from leaves to root

The other method to find the impact and the sum of weights of the segments is not to identify the segments first, but the other way around. We calculate some attributes of the edges of the graph, which are functions that assign a number to every edge. Then we use this to immediately get the impact and sum of weights of every segment. The attributes are calculated in such a way that eventually these values, impact and sum of weights, are assigned to the breaker edges.

We will call these new attributes *freq* and *imp*, *freq* is the sum of weights and *imp* the sum of values. They are the cumulative sum tracing from leaves to root. Summing from leaves to root means that the incoming edges of leaves have *freq* equal to the weight of the leaf and *imp* equal to the value of the leaf. All other edges have as *freq* the weight of all vertices below them, which is exactly the *weight* of its head added to the *freq* of the outgoing edges of its head. The *imp* is calculated the same way, but with values.

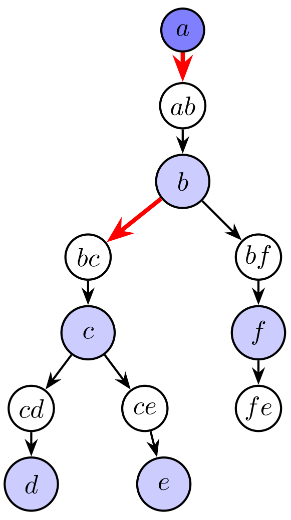


Figure 4.4: Simple configuration graph

[1]An edge with an orientation has a tail and a head, the vertex it is pointing to is the head.

Then for every edge it is clear that *freq* gives the sum of weights and *imp* the sum of the values of the leaves up to and including its head, which are all vertices after that edge.

Example 4.8. Suppose we have a graph with weights and values on the vertices. We will demonstrate the operation that is described. From the leaves up we sum all weights and values, the following two figures give the attributes *freq* and *imp* respectively.

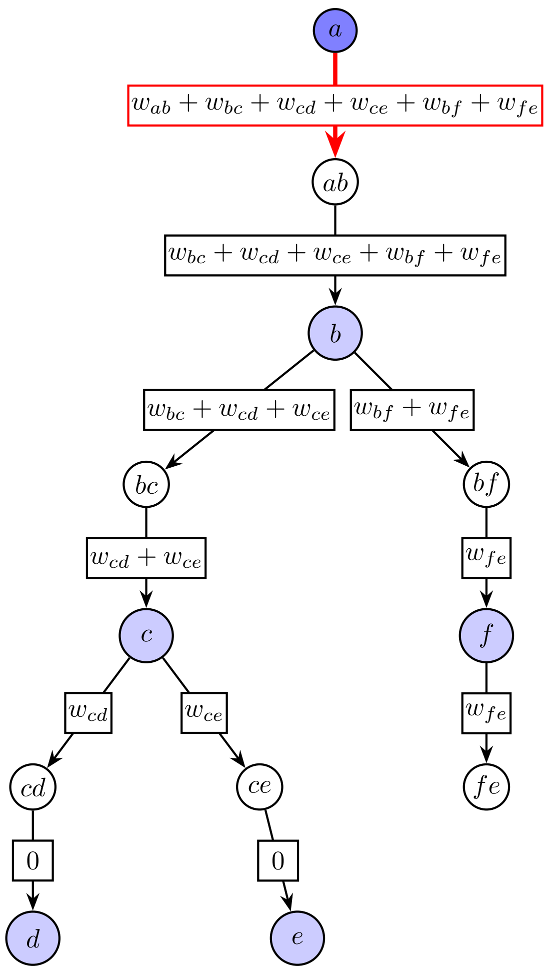


Figure 4.5: Configuration graph with *freq* given on every edge

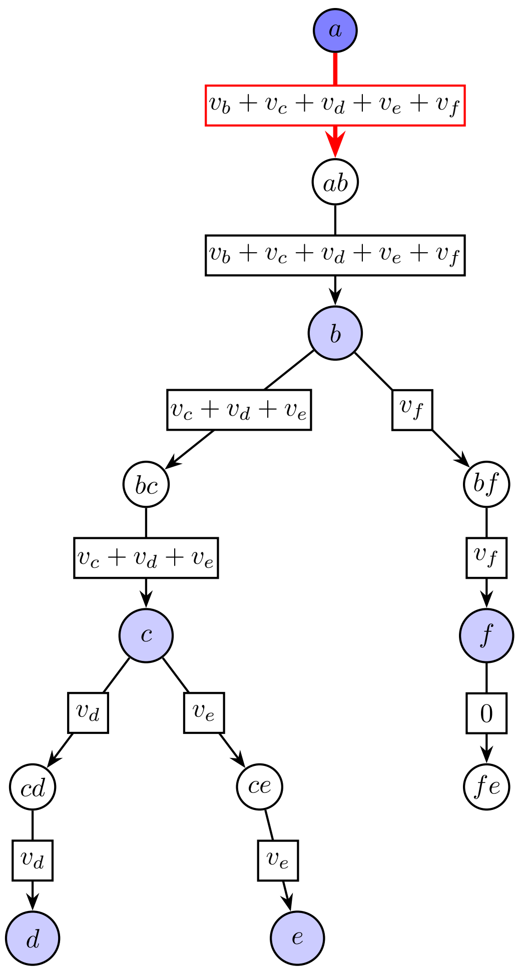


Figure 4.6: Configuration graph with *imp* given on every edge

In these figures we see that the *freq* and the *imp* of edge (a, ab) are equal to the sum of weights and sum of values of everything below a. Because (a, ab) is a breaker edge and its tail is a root, we know that every edge is part of the same segment and therefore has the same impact we see that $R(G)$ of this graph G is equal to $\mathit{freq}((a, ab)) \cdot \mathit{imp}((a, ab))$.

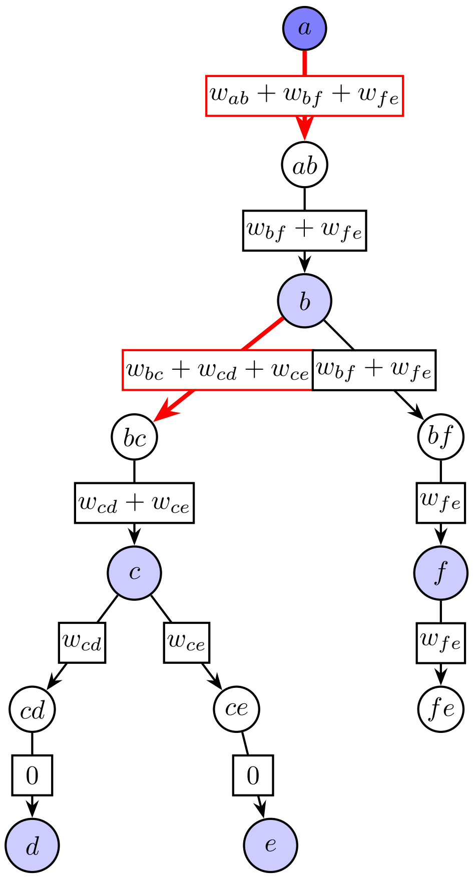


Figure 4.7: Configuration graph with *freq* given on every edge

So in the case where there are no other breaker edges than the one from the root, this method gives the right result. Now we have to look at the case where there are other breaker edges. In that case we have several segments for which we want to know the sum of the weights of all edges in the segment and the impact of the segment. The impact of the segment is defined as the sum of values of all vertices below the corresponding breaker edge. With this method *imp* gives exactly that sum. The sum of weights of the edges in a segment is however not yet given by *freq*. For *freq* to give the right result, we don't want to keep adding all weights from the leaves to the root, but stop every time we come across a breaker edge. That way the *freq* on every breaker edge is not the sum of weights of all edges below the breaker edge to the leaves but from the breaker edge to and not including other breaker edges. That is precisely the number we are looking for.

Algorithm 4.2: Assign attributes *freq* and *imp* to G by summing bottom-up

```
Data: Directed graph $G$ with values and weights on vertices, and booleans whether or
      not they are breaker edges
Result: graph $G$ with added attributes freq and imp
1  SortedVertices = topologically sorted list of vertices of $G$;
2  $freq(E(g)) = 0$;
3  $imp(E(g)) = 0$;
4  for $i$ in $1 : \#SortedVertices$ do
5      node = SortedVertices[$i$];
6      incoming = incoming edges of node in $G$;
7      if $incoming \neq \emptyset$ then
8          outgoing = outgoing edges of node in $G$;
9          $freq(\text{incoming}) = w_{\text{node}} + \sum_{\substack{e \in \text{outgoing} \\ e \notin B}} freq(e)$;
10         $imp(\text{incoming}) = v_{\text{node}} + \sum_{e \in \text{outgoing}} imp(e)$;
11     end
12 end
13 return $G$;
```

The topological sort used in this algorithm gives an ordered list of vertices. This ordering ensures that for every vertex, all vertices below that vertex come before it in the list.

For every edge we now know the sum of values of vertices below that edge and the sum of weights of vertices below that edge until other breaker edges. The impact and sum of weights of a segment are therefore exactly the *imp* and *freq* of the corresponding breaker edge. These attributes can be used to easily calculate $R(G)$.

Algorithm 4.3: Calculate $R(G)$ using attributes *imp* and *freq*

```
Data: graph $G$ with the attributes $w, v$ and root on the vertices and breaker, imp and freq
      on the edges
Result: $R(G)$
1 breakers = all edges of $G$ with breaker==true;
2 result = imp(breakers) · freq(breakers);
3 return result;
```

4.4 Results

The algorithms given in this chapter have been tested on the two networks given in Chapter 2.

Network 1

This network consists of all stations supplied by one HV substation and has 416 vertices. Let C be the current configuration, which is given. This contains 65 breaker edges.

The result of both algorithms is $R(G_C) = 3240.196$.
Algorithm 4.1 gives this result in 0.355 seconds.
Algorithm 4.2 takes 4.555 seconds. Algorithm 4.3 then gives the result in 0.002 seconds.

Network 2

Network 1 is a subgraph of network 2. This network consists of all stations supplied by 55 HV substations and contains 10954 vertices. Let C be the current configuration, which is given and contains 2133 breaker edges.

The result of both algorithms is $R(G_C) = 63158.71$.
Algorithm 4.1 gives this result in 16.261 seconds.
Algorithm 4.2 takes 2835.853 seconds. Algorithm 4.3 gives the result in 0.02 seconds.

Conclusion

It is clear that for a single calculation Algorithm 4.1 should be used because this is faster. This changes when the calculation has to be done a lot of times, which is the case in optimization problems. The objective function then needs to be calculated for a lot of possible solutions. In that case it may be faster to update the attributes of the graph initiated by Algorithm 4.2 and using Algorithm 4.3.

Remark 4.9. It is not clear why Algorithm 4.2 is so complex, that it takes a lot more time than Algorithm 4.1. Especially when we see that the topological sort is done very quickly. When investigating the algorithm after the topological sort we see that we want to deal with every vertex and edge only once or twice. The problem could lie in finding the outgoing and incoming edges of the vertices. It is likely that the implementation can be improved, but, seeing as this initialization needs to be executed only once if the attributes can be updated, improving the implementation of this algorithm has no priority.

Chapter 5

Optimal placement of circuit breakers

In this chapter one of the possibilities to change the configuration of the network will be considered, the possibility to add circuit breakers to the current set. The net openings will remain the same, which means that the topology of the configuration graph also stays the same. In the model, adding circuit breakers corresponds to expanding the set of breaker edges. We start with the configuration $C = (G, A, B, O)$ and will look at the reconfigurations $C' = (G, A, B', O)$ with $B \subset B'$ to find the solution which minimizes $R(C')$, while satisfying the constraints. In this chapter, a solution will also be identified by a subset of edges. The actual solution is then the reconfiguration with this subset added to B.

The configuration graph only changes in the sense that the attribute *breaker* will change for certain edges from false to true, all other characteristics stay the same. In this chapter we will therefore only consider configuration graphs, because these contain all necessary information.

5.1 Constraints

There are several constraints that have to be satisfied by feasible solutions. These were given in the practical sense in Chapter 2 and for the model in Section 3.5.2. These constraints are the following:

- Radiality constraint: the configuration graph is A-radial.
- Breaker constraint: the configuration graph does not contain a path with 4 or more breaker edges.
- Time constraint: the total amount of time a reconfiguration costs cannot exceed some maximum amount of time.
- Cost constraint: the total cost of a reconfiguration cannot exceed some maximum cost.

Radiality constraint

The first is always the radiality of a configuration. We assume that the current configuration always satisfies all constraints, so in particular the radiality constraint. The radiality constraint is on the set A and the topology of the configuration graph. Both of these stay the same with the reconfigurations considered in this chapter, so all of these solutions remain A-radial. This means that we don't have to consider this constraint in the current chapter.

Breaker constraint

The most important constraint when considering the breaker edges is that there can be no more than 3 breaker edges on any path in the configuration graph. We assume this is satisfied for the current configuration C, but by adding breaker edges this can be violated. This is a constraint that has to be checked, either during the placement (by removing certain edges or combinations from the possibilities) or afterwards (by checking if the solution that was found satisfies this constraint).

Time and cost constraints

The time and cost constraints are very similar and easier to satisfy when considering only a single kind of change. The cost and time needed for adding a breaker edge is the same for every edge, but it is different for changing an opening. The time constraint is formulated as $x \cdot T_b + y \cdot T_o \leq T$, where x is the number of breaker edges added and y is the number of changes to the openings. In this chapter we have $y = 0$, so the constraint is $x \cdot T_b \leq T$, which is the same as the constraint that the number of added breaker edges is less than or equal to $\frac{T}{T_b}$. This also holds for the cost constraint, $x \cdot S_b + y \cdot S_o \leq S$, which can be restated in this case as $x \leq \frac{S}{S_b}$.

Together these two constraints combine to the requirement that $x \leq Z$, where $x = \#(B' \setminus B)$ and the bound $Z = \min(\frac{T}{T_b}, \frac{S}{S_b})$. For an instance of the optimization problem, Z is a given constant.

Constraints

We see that there are two constraints to consider in this chapter. The feasible solutions for this type of reconfiguration have to satisfy the breaker constraint, which means that all paths in the configuration graph contain at most 3 breaker edges, and the constraint $x \leq Z$ for Z given as a part of the input of the optimization problem and $x = \#(B' \setminus B)$.

This chapter first focuses on several possible algorithms finding optimal solutions satisfying the time and cost constraints with Z as a variable. Then we will investigate different ways to check the breaker constraint and see which way should be used per algorithm and how much this affects their performance.

5.2 Placing one circuit breaker

In this section we will look at the special instance where $Z = 1$, which means that the maximal number of additions to B is 1. An optimal solution is a reconfiguration C' with 0 or 1 edges added to B with the smallest result for $R(G_{C'})$. In this section different methods to find an optimal solution are explained and tested on the test networks.

We will see that if adding a breaker edge is possible (there exists a non-trivial feasible solution[1]), then every non-trivial reconfiguration is not worse than the current configuration. Every non-trivial reconfiguration either has the same value for the objective function or a smaller one. This is clear from the practical sense, the only effect of a circuit breaker is that some customers will no longer be affected by certain failures. Placing a circuit breaker can therefore only reduce the SAIDI, not increase it.

[1]a reconfiguration is non-trivial if it is not equal to the current configuration, and feasible if it satisfies the constraints.

We will use heuristics to eliminate certain solutions because it is clear they will not minimize the objective function. Then it is investigated how the evaluation of the remaining possibilities can be done efficiently.

5.2.1 Brute force algorithm

The most obvious way to find an optimal solution is to try every solution. This means that for every $e \notin B$, the value $R(G_{C'})$ for $C' = (G, A, B \cup \{e\}, O)$ is calculated. A solution with the smallest result is an optimal solution (it is of course allowed that this is the current configuration).

Algorithm 5.1: Brute force $R(G_{C'})$ for all possibilities adding one breaker edge

```
Data: Configuration graph $G_C$
Result: Table of edges creating reconfigurations $C'$ together with $R(G_{C'})$
1 possibilities = $E(G_C) \setminus B$;
2 list = list of zeroes of same length as possibilities;
3 for i in 1:length(possibilities) do
4     tempgraph = $G_C$ with attribute breaker is changed to true for possibilities[$i$];
5     list[$i$] = $R$(tempgraph);
6 end
7 return table with column 1 = possibilities, column 2 = list;
```

This method calculates the objective function for every possible solution. It can also be stored for every solution, so that planners are able to choose a solution that works best in practice from the given list of results. The objective function is calculated using the fastest algorithm from the previous chapter, which is Algorithm 4.1.

Of course, it is also possible to forget every result but the best so far. The output is then only an optimal solution with its result for the objective function. This could be done to save space that is used if we store the result of all solutions.

5.2.2 Difference in outcome objective function

We want to see if there is a more efficient algorithm to find an optimal solution for this specific subproblem, instead of simply calculating the result of the objective function for every possibility. Two ways to find such an algorithm are to calculate the result of the objective function faster for every possibility, or to limit the number of calculations of the objective function by excluding certain solutions from the possibilities to be an optimal solution. In order to use either of those we need to know how the objective function changes with this one addition to the configuration.

Benefit of adding a breaker edge

The objective function is defined as $R(G) = \sum_{j \in V(G)} (w_j \cdot I_j)$, where $I_j = \sum_{\substack{i \in V(G) \\ i \text{ after } e}} v_i$ for e the closest breaker edge in the path from j to the root. We investigate what happens when there is one additional breaker edge. From now on the closest breaker edge will mean the closest on the path towards the root.

It is clear that adding a breaker edge does not change the vertices in G nor their weights w_j, it only changes the impact for some vertices. Let e be an edge in G that is added to the set of breaker edges. The edges for which e is now the closest breaker edge will have a different impact, while the impact of all other edges stays the same because their closest breaker edge does not change. The vertices that have e as the closest breaker edge are exactly the vertices below e, but only until the next breaker edges. The difference in the outcome of the objective function is therefore only caused by the new impact of these vertices.

Lemma 5.1. *Adding a breaker edge e cannot increase the value of the objective function.*

Proof. The difference is only caused by the new impact of the vertices below e until other breaker edges. The impact is the sum of values of vertices below the closest breaker edge, so if e would be the closest breaker edge for certain vertices, then we know that their current closest breaker edge is farther away than e, which means that more vertices are considered for the old impact. The new impact is therefore always smaller or equal to the old impact, and because the weights are non-negative, it's clear that the contribution of these vertices is smaller or equal to what it was, while it remains the same for all other edges. This shows the claim that adding a breaker edge will never increase the result of the objective function. □

This can also be stated in terms of segments. The vertices that have e as the closest breaker edge correspond to the segment that is now defined by e. The new breaker edge e was previously contained in a segment defined by some breaker edge e' and divides that segment into two. The segment defined by e' now consists of all edges and vertices of the old segment until e and the new segment is the one defined by e. The impact of a segment is defined as the sum of the values of all vertices below the breaker edge. Since e lies below e' (it was contained in its segment), all edges below e also lie below e', which means that the impact of the edges in this new segment is smaller or equal to the impact of the old segment, while the segment defined by e' has no change in impact.

Example 5.2. We will show these facts using the configuration graph on the right. Let's say we want to add edge $\{c, cf\}$ to the set of breaker edges. $\{c, cf\}$ was contained in the segment defined by edge $\{a, ab\}$, which includes vertices ab, bc, cf and fg. The impact of these vertices is the same for all of these vertices, namely the sum of values of all vertices below edge $\{a, ab\}$, which are vertices b to i. The contribution of this segment is therefore the sum of weights of vertices ab, bc, cf and fg times that impact. The other segments in the graph are defined by edge $\{c, cd\}$ and edge $\{f, fh\}$. We see that adding $\{c, cf\}$ to the breaker edges will not change anything in these segments, not the elements of the segments nor the impact, the only changes are in the segment defined by $\{a, ab\}$. When $\{c, cf\}$ is added, we see that it will be the closest breaker edge for cf and fg, and a new segment is created which is defined by edge $\{c, cf\}$ and will consist of vertices cf, f, fg and g. The vertices included in the segment of $\{a, ab\}$ are now only ab, b, bc and c. The impact of this segment remains the same, so we see that the contribution of this segment is now the sum of weights of ab and bc times the impact of this segment. The contribution of the segment defined by $\{c, cf\}$ is the sum of weights of cf and fg times the impact of this segment, which is the sum of the values of vertices f, g, h and i. We see that the reduction in outcome of $R(C)$ is the difference in impact for cf and

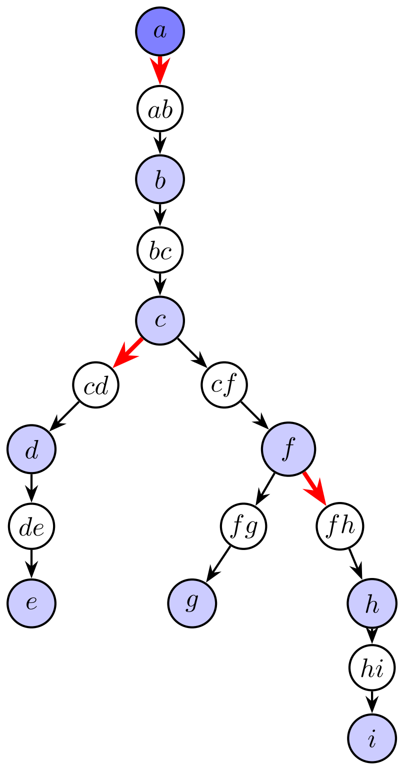


Figure 5.1: Tree with breaker edges highlighted in red

fg, so the difference is the sum of weights of cf and fg times the sum of values of vertices b to e.

Lemma 5.3. *Suppose e has current closest breaker edge e', and e is added to the set of breaker edges. Let W be the sum of weights of vertices in the segment defined by e, $I_{e'}$ be the impact of the segment defined by e' and I_e the impact of the segment defined by e. The reduction in the result of the objective function caused by adding e to the breaker edges is equal to $W \cdot (I_{e'} - I_e)$.*

Proof. Again, the only change in the result is caused by the new impact of the vertices in the segment defined by e. Therefore the old contribution of these vertices are subtracted from $R(G)$; $-W \cdot I_{e'}$, and the new contribution is added; $+W \cdot I_e$. The reduction that adding e to the set of breaker edges gives, which will be called the **benefit** of e, is exactly the sum of weights of the segment defined by e times the outcome of substracting the sum of values of all vertices below e, which would be the impact of the segment of e, from the impact of the current segment containing e; that is $W \cdot (I_{e'} - I_e)$. □

5.2.3 Improvement of the brute force algorithm

We see that there are two facts which we can directly use to reduce the time needed to calculate the objective function.

The first is that the difference in the results of the objective function only depends on the segment containing the edges. It is therefore not needed to calculate the objective function for the whole reconfiguration, but it is possible to save the result per segment and only recalculate that result for the segment where a change occurs.

The second method is not to recalculate the objective function, but to calculate the benefit for every edge, the reduction in outcome of objective function if that edge was added to the set of breaker edges.

Both of these methods will be faster than Algorithm 5.1, because the number of calculations per possibility is reduced greatly. The first method however requires that the contribution to $R(C)$ is saved for every segment, while it does not calculate less than the second method. Both need to determine the impact of the old segment, the impact of the new segment and the sum of weights in the new segment. The second method is therefore preferred over the first.

Algorithm 5.2: Improved brute force benefit of one added breaker edge

```
Data: Configuration graph $G_C$
Result: Table of edges creating reconfigurations $C'$ with their benefits $R(G_C) - R(G_{C'})$
1  possibilities = $E(G_C) \setminus B$;
2  list = list of zeroes of same length as possibilities;
3  for i in 1:length(possibilities) do
4      oldclosestbreaker = first breaker edge on incoming path from possibilities[i] to a root;
5      oldimpact = sum of values of vertices in out-tree $G_C$ after oldclosestbreaker;
6      newimpact = sum of values of vertices in out-tree $G_C$ after possibilities[i];
7      $H$ = delete breaker edges and possibilities[i] from $G_C$;
8      newsegment = component of $H$ containing head of possibilities[i];
9      newsegmentweight = sum of weights of vertices in newsegment;
10     benefit[i] = newsegmentweight · (oldimpact - newimpact);
11 end
12 return table with column 1 = possibilities, column 2 = benefit;
```

We could take these methods a step further by saving the impact for every segment, because this will be equal for every edge in the same segment and do not have to be calculated again for every edge. This idea of using the attributes on the graph is investigated in Section 5.2.5.

Algorithm 5.2 can also be improved by considering edges per segment. That way the closest breaker edge is already known so it saves a lot of the same work. Besides that improvement, the following algorithm also calculates $I_{e'} - I_e$ directly, by taking the out-tree from the head of the breaker edge, but in a graph where the edge under consideration is removed.

Algorithm 5.3: Calculate benefit directly for all possible solutions per segment

```
Data: Configuration graph $G_C$
Result: Table of edges creating reconfigurations $C'$ with their benefits $R(C) - R(C')$
1  $H$ = delete breaker edges from $G$;
2  for component in $H$ do
3      if component does not contain a root then
4          topnode = vertex with no incoming edges in component;
5          for $e$ in E(component) do
6              splitgraph = delete $e$ from $G$;
7              tempgraph = the out-tree of topnode in splitgraph;
8              $I$ = sum of values of tempgraph;
9              new$H$ = delete $e$ from $H$;
10             newsegment = component of new$H$ containing the head of $e$;
11             $W$ = sum of weights of vertices in newsegment;
12             append labs with label of $e$;
13             append benefit with $I \cdot W$;
14         end
15     end
16 end
17 return table with column 1 = labels, column 2 = benefit;
```

5.2.4 Heuristics

Excluding possible solutions

Besides calculating the benefit for every possible solution, a way to more efficiently find an optimal solution is by using the characteristics of the configuration to be able to tell beforehand which solutions will not be optimal. Four ideas are investigated in this part of the section.

a) Adding a breaker after a vertex of V_D is not beneficial.

b) While tracing along a path, determining when to stop because an optimal solution is already found.

c) While tracing a tree, determining which branch to find an optimal solution in.

d) Choose the edge with the greatest weight times impact, or the segment with the greatest sum of weights times the impact.

The first idea is used in the practical problem and will be proven mathematically. Ideas b) and c) are on tracing the forests and looking for an indication to stop tracing that branch or to choose a branch when coming at multiple outgoing edges. That way a lot of solutions could be excluded. The last idea is to get an indication on good solutions by using the values obtained when calculating $R(C)$.

a) Edges after V_D

In practice, when circuit breakers are placed on a distribution line, they are always placed directly behind the substation seen from the root. These locations correspond to the edges whose tails represent substations, which are the vertices in V_S, and heads represent distribution lines, the vertices in V_D. We will prove in this part that this edge is indeed always more beneficial then the other edge adjacent to the vertex representing the distribution line, which means that for every two edges adjacent to an element of V_D one of them is excluded from the best possibilities. This eliminates almost half of the edges of the graph from the possibilities. This fact is illustrated in Figure 5.2, where the benefits of the edges along a path have been calculated. The reason behind this heuristic is that the benefit of a circuit breaker is that the customers above it are protected from failures below it. Placing a circuit breaker at the other side of the distribution line will not protect more customers, but will protect from less failures.

Lemma 5.4. *For every $i \in V_D$ with degree 2 and no adjacent breaker edges the following holds:*
The reduction of the incoming edge is larger than or equal to the reduction of the outgoing edge.

Proof. Let $i \in V_D$ be a vertex with degree 2 and no adjacent breaker edges. Let e_1 be the incoming edge of i and e_2 the outgoing edge. Let $I \in \mathbb{N}$ be the impact of the current segment, I_{e_1} and W_{e_2} the impact and sum of weights of the segment that would be defined by e_1 and I_{e_1} and W_{e_2} the same for e_2.

The reduction of an edge is the impact of the current segment minus the sum of the values of all vertices below the edge, and that times the sum of weights below that edge until breaker edges. In this case the reduction for e_1 is $(I - I_{e_1}) \cdot W_{e_1}$ and for e_2 it is $(I - I_{e_2}) \cdot W_{e_2}$. We see that because i is of degree 2, the segment created by the incoming edge is the same as the one created by the outgoing edge together with vertex i. Therefore we have $W_{e_1} = W_{e_2} + w_i$. For the same reason the same holds for the impact: $I_{e_1} = I_{e_2} + v_i = I_{e_2}$, because $i \in V_D$. The facts that $I - I_{e_1} \geq 0$, $W_{e_2} \geq 0$ and $w_i \geq 0$ prove that the reduction of the incoming edge e_1: $(I - I_{e_1}) \cdot W_{e_1} = (I - I_{e_2}) \cdot (W_{e_2} + w_i)$ is greater than or equal to the reduction of the outgoing edge e_2: $(I - I_{e_2}) \cdot W_{e_2}$. □

b) Monotonicity on a path

The first question that arises when investigating the possibilities along a path is whether the benefits along the path are monotone. Figure 5.2 gives an example of a path with the values on elements of V_S given in the figure, $w_i = 0.1$ for all $i \in V_D$ the top node is the root and its outgoing edge is a breaker edge.

The benefit is calculated for every incoming edge of V_D by summing the weights of the vertices below the edge and multiplying that with the sum of values of the vertices above that edge. For example the second edge has seven elements of V_D below it and one vertex with value 1 above it, so the benefit of adding that edge is $0.7 \cdot 1 = 0.7$. It is clear from this figure that the benefit along a path is not monotone.

The second question is whether it is monotone after a certain point, so that it is certain that an optimal solution has been found when the benefit starts to get smaller. We see however in the same path that this is also not the case, because the next to last vertex is given a larger value. While it seems that the

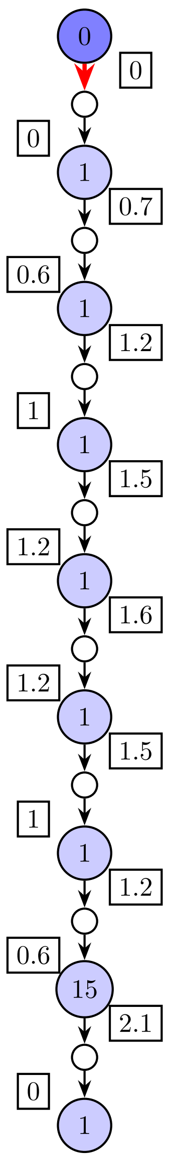


Figure 5.2: Path with benefit not monotone

benefits are only decreasing after edge 5, the benefit of the last edge is greater than the best so far.

It is clear that the values of the vertices can be distributed in such a way that there is no telling where on the path lies the best solution. Particularly because the weights of the vertices can also be distributed in many different ways. It may seem a good idea to try an edge after a vertex with the maximum value, because this gives a greater $I_{e'} - I_e$, but not if it is the case that the weights above it are so large that it would be better to incorporate these to get a large W. It is therefore also possible to look at the maximum weight. Besides the values and weight increasing the other way around, it is also difficult that multiple edges can have the maximum weight or multiple vertices with maximum value, or the maximum not differing enough from the other weights and values. All these reasons together create a lot of conditions to check before getting an indication where to find an optimal solution and even then it may not even be certain, it is therefore not efficient to use this method to solve the problem.

c) Branch of the tree

The third idea also raises difficulties. There is no easy way to indicate which branch will contain an optimal solution. In Figures 5.3 to 5.5, we again take equal weight for all elements of V_D, namely 0.1. These figures show that the benefit of the first edge is not an indication, nor the branch with the maximum value, nor the branch with the greatest total value. Seeing that this is already clear with equal weights, adding different weights will only increase the difficulty to give an indication. It is clear that these heuristics on the graph will not provide efficient algorithms.

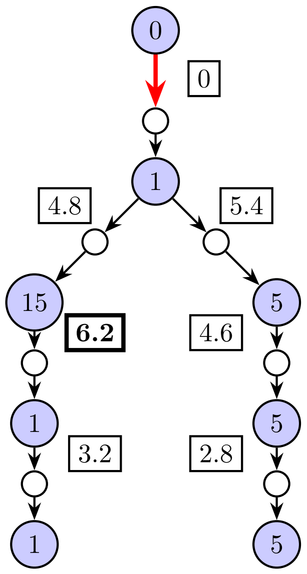


Figure 5.3: Tree with optimal solution in branch with the maximum value, the greatest sum of values, but not the best first edge

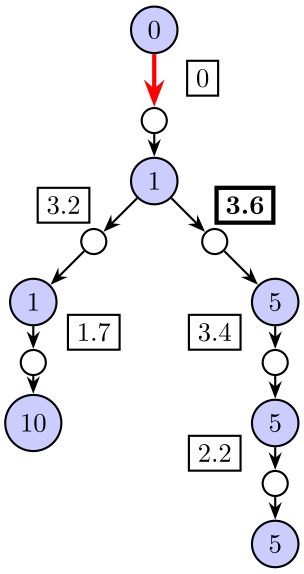


Figure 5.4: Tree with optimal solution in branch with the best first edge and the greatest sum of values, but not the maximum value

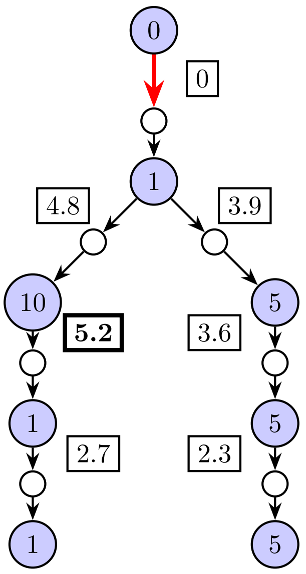


Figure 5.5: Tree with optimal solution in branch with the best first edge, but not the maximum value nor the greatest sum of values

d) Choose the worst edge

The last idea is to see where the greatest contributors to the objective function lie, which segment has the worst contribution to $R(G)$. It could be the place where the most reduction can be found. This could be used as an indication where to find an optimal solution. It is however not certain that an optimal solution is found at the segment with the greatest contribution, because adding a breaker in this segment will not give as much benefit as placing one in a segment that has lower contribution, but with the weights and values distributed in such a way that there is one edge that gives a greater reduction than all others.

Example 5.5. Figure 5.6 gives an example configuration graph with the values on the vertices and weights of 0.1 on the vertices in V_D. The benefit of adding an edge to the breaker edges is given next to that edge. This example shows that the best benefit is not found in the segment with the worst contribution, because the segment on the left contributes $(0.1+0.1+0.1)\cdot(1+5+1) = 2.1$ and the segment on the right $(0.1 + 0.1 + 0.1) \cdot (1 + 1 + 20) = 6.6$.

This idea is only useful if a good solution has been found, then the segments or edges that contribute less than the best reduction found can clearly not be optimal. But it needs to save the contributions for every segment or edge, while it probably will not give a much more efficient algorithm. In the next section we will give an algorithm that gives the exact benefit for every edge more efficiently.

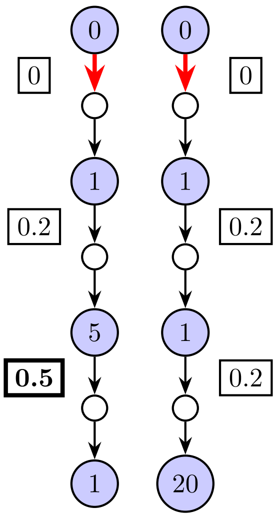


Figure 5.6: Configuration graph with values and benefits

5.2.5 Benefit calculation using attributes

In this section we will look at the benefit that was mentioned in the previous section. Calculating this benefit for all possibilities is just as useful as creating every reconfiguration and calculating the objective function. After calculating the benefit for all possibilities, the solution with the highest benefit will be an optimal solution.

The benefit of adding an edge e to the breaker edges is calculated with W, the sum of the weights of the vertices in the segment that would be defined by e if it was a breaker edge, I_e, the sum of the values of all vertices below e, and I, the impact of the segment e is an element of in the current configuration. The benefit of e is then calculated by $W \cdot (I - I_e)$.

It is possible to calculate this by tracing for every edge, which is faster then calculating the objective function for all edges. But another method shows the real advantage of Algorithm 4.2, where the weights and values are summed from leaves to root, because W and I_e are exactly the attributes *freq* and *imp* that are saved for every edge by this algorithm. *freq*(e) is the sum of weights of the vertices below e, but only until other breaker edges, and *imp*(e) is the sum of values of all vertices below e. So these values are easily obtained from a graph that is the result of Algorithm 4.2.

Example 5.6. We continue with the example given in Figure 5.1, Figures 5.8 and 5.7 give the result of Algorithm 4.2. If we want to calculate the benefit of adding edge (c, cf) to the breaker edges, we can use the attributes on the edges of this graph. If (c, cf) were a breaker edge, then the vertices cf and fg would have another impact. The current impact of these vertices is the same as that of breaker edge (a, ab), which is $v_b+v_c+v_d+v_e+v_f+v_g+v_h+v_i$, while the new impact is

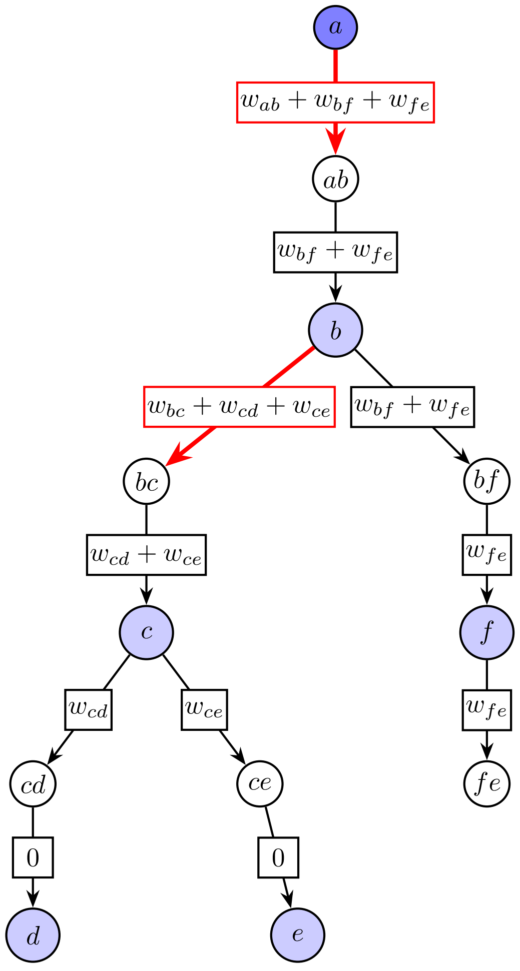


Figure 5.7: Configuration graph with *freq* given on every edge

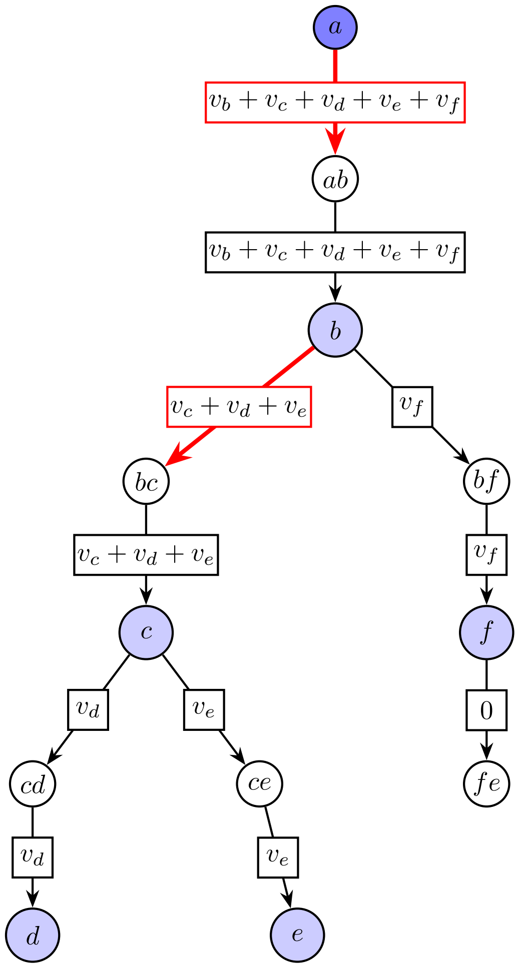


Figure 5.8: Configuration graph with *imp* given on every edge

all vertices below (c, cf), $v_f + v_g + v_h + v_i$, which is given as *imp* on edge (c, cf). The difference in impact is therefore $I - imp((c, cf))$, where I is the current impact of the segment. The difference of the contribution of the two vertices cf and fg to the objective function is therefore the sum of their weights times the difference in impact. The sum of their weights is exactly $freq((c, cf))$, so the benefit of edge (c, cf) is given by $freq((c, cf)) \cdot (I - imp((c,cf))) = (w_{cf} + w_{fg}) \cdot (v_b + v_c + v_d + v_e)$.

The impact of the current segment is *imp* of the breaker edge defining the segment. We want to know this for every edge within a segment. So we want to add another attribute to the graph *impsegment* to give the impact of the current segment of every edge. When this attribute is added, all information for the benefit of adding edges to the breaker edges is already known as attribute on these edges themselves. Calculating the benefit of every edge using these attributes is very fast.

Algorithm 5.4: Add attribute *impsegment* to the graph

```
Data: Configuration graph $G_C$ with attributes freq and imp
Result: Graph with attribute impsegment added
1  $impsegment = imp(E(G_C))$;
2  tempgraph = delete breaker edges from $G_C$;
3  comp = maximal components of tempgraph;
4  for $H$ in comp do
5      topnode = vertex without incoming edges in $H$;
6      if $topnode \notin A \subset V(G)$ then
7          bedge = incoming edge of topnode;
8          impact = $imp$(bedge);
9          $impsegment$ in $G$ of all edges that also occur in $H$ = impact;
10     end
11 end
12 return $G$;
```

Algorithm 5.5: Calculate benefits from attributes *freq*, *imp* and *impsegment*

```
Data: Configuration graph $G_C$ with attributes freq, imp and impsegment
Result: Table of edges creating reconfigurations $C'$ with $R(G_{C'})$
1  benefits = $freq(E(G)) \star (impsegment(E(G)) - imp(E(G)))$;
2  return table with column 1 = $E(G)$, column 2 = benefits;
```

5.2.6 Results

The algorithms have been tested on the two networks given in Chapter 2.

Network 1

This network consists of all stations supplied by one HV substation. Let C be the given current configuration. This configuration consists of 416 vertices and 415 edges between them and 65 breaker edges. $R(G_C) = 3236.483$, as was calculated in the Section 4.4.

The result of both algorithms is the edge with label 149 with $R(G_{C'}) = 3090.643$, where C' is the reconfiguration with that edge added to the set of breaker edges.

- Algorithm 5.1 gives this result in 317.185 seconds.
- Algorithm 5.2 gives the result in 40.939 seconds. The improved Algorithm 5.3 takes 5.36 seconds.
- Algorithm 5.4 takes 0.409 seconds, Algorithm 5.5 then gives the result in 0.003 seconds, its output is the result with largest benefit of 145.8401. Of course Algorithm 5.4 requires the execution of Algorithm 4.2 first, to obtain the attributes *freq* and *imp*, which takes 4.555 seconds.

Network 2

Network 1 is a subgraph of network 2. This network consists of all stations supplied by 58 HV substation. Let C be the current configuration, which consists of 10954 vertices and 10896 edges between them and 2133 breaker edges. $R(G_C) = 62196.06$ for this network. This was also calculated in the Section 4.4.

The result of both algorithms is the edge with label 8450 with $R(G_{C'}) =$, where C' is the reconfiguration with that edge added to the set of breaker edges.

- Algorithm 5.1 takes at least 100 hours, and was not executed completely.
- Algorithm 5.4 takes 124.833 seconds, Algorithm 5.5 then gives the result in 0.02 seconds, its output is the result with largest benefit of 342.4048. Of course Algorithm 5.4 requires the execution of Algorithm 4.2 first, to obtain the attributes *freq* and *imp*, which takes 2392.899 seconds.

Conclusion

When trying to find an optimal solution for $Z = 1$, it is much faster to calculate the benefit for every solution instead of the objective function itself. The assignment of attributes to the graph takes some time, but the calculation of the benefits afterwards is very fast. We see that even with the algorithms assigning these attributes, the total amount of time needed for the execution is still faster than Algorithms 5.1, 5.2 or 5.3.

5.3 Optimal placement of multiple circuit breakers

In this section Z is any natural number. The solution set is all reconfigurations $C' = (G, A, B', O)$ with $B \subset B'$ and $\#(B' \setminus B) \leq Z$. All solutions can therefore be given by a set of edges not in B with at most Z elements. We have seen that adding an edge to the set of breaker edges can never increase the outcome of the objective function, so if we assume that $\#(E(G) \setminus B) \geq Z$ then there is always an optimal solution with Z elements.

If an optimal solution exists with fewer than Z elements, then this solution can also be found from an optimal solution with Z elements, by deleting those that have benefit 0. It is possible that there is an optimal solution with p elements and this method gives an optimal solution with $p+1$ elements and no element with benefit 0. But by using this method for all optimal solutions with Z elements, an optimal solution with smallest number of additions will be found. This is useful in the practical sense, because even though it may be allowed to use a certain amount of time or money, it is of course preferred to use as little as possible.

Some algorithms to find an optimal solution with exactly Z additions (if this is possible) and the results on the test networks are given in this section. Some algorithms always give an optimal solution, while others approximate the solution. The approximation algorithms are faster, but will not always give an optimal solution. It depends on the specific practical situation which is more important, speed or optimality. In this section we will use two standard algorithms, namely a greedy algorithm and a branch and bound algorithm.

5.3.1 Heuristics

In this subsection we will show that the heuristics that were proven in Section 5.2.4 can also be used when adding multiple breaker edges. This said that the reduction of the incoming edge of a vertex in V_D is always greater than or equal that of the outgoing edge. The following lemma shows that a similar fact holds when adding multiple breaker edges.

Lemma 5.7. *Let $C = (G, A, B, O)$ be the current configuration. Suppose we have the possible solution $C' = (G, A, B \cup B', O)$, where we add the edges in B' to the set of breaker edges. Suppose*

B' contains an outgoing edge e_1 from an vertex of V_D. Let e_2 be the incoming edge of this vertex. Then the solution which adds $(B' \setminus e_1) \cup e_2$ to the set of breaker edges is not a worse solution than C'.

Proof. Apply Lemma 5.4 to the configuration $(G, A, B \cup (B' \setminus e_1), O)$. This lemma says that the benefit of e_2 is greater than or equal the benefit of e_1. That means that, from the current configuration C, the reduction caused by adding $(B' \setminus e_1) \cup e_2$ to B is greater than or equal to adding B'. Adding the incoming edge instead of the outgoing edge is therefore not a worse solution. □

Corollary 5.8. There exists an optimal solution that adds only incoming edges of elements of V_D to the breaker edges.

Proof. The number of possible solutions is limited by the number of edges and Z, which makes it finite. So it is clear that there exists an optimal solution.

Suppose we have an optimal solution. We can use Lemma 5.7 to iteratively replace any outgoing edges of vertices in V_D with their incoming edges. We see that in every step the solution will not get worse, so the resulting solution will also be an optimal solution. This is an optimal solution that only adds incoming edges of V_D. □

5.3.2 Brute force

The first possible algorithm is to calculate the value of the objective function for every feasible solution. This algorithm finds all subsets of Z elements from the edges $E(G) \setminus B$ that are incoming edges of V_D. For all of these subsets a reconfiguration C' is created by adding the subset to B and the outcome of $R(C')$ is calculated.

Algorithm 5.6: Calculate $R(G_{C'})$ directly for all possible solutions with Z additions to B

```
Data: Configuration graph $G_C$, natural number $Z$
Result: Table of subsets of edges creating reconfigurations $C'$ with $R(G_{C'})$
1 possibilities = all possible subsets of $Z$ elements of $E(G_C) \setminus B$;
2 reduction = list of zeroes of same length as possibilities;
3 for i in 1:length(possibilities) do
4     tempgraph = $G_C$ with attribute breaker is TRUE for all edges in possibilities[$i$];
5     reduction[$i$] = $R$(tempgraph);
6 end
7 return table with column 1 = possibilities, column 2 = list;
```

5.3.3 Using benefit calculation

The number of subsets of Z elements grows exponentially with the size of the input. It is not efficient to calculate the objective function for every possible subset, so we want to make a selection which subsets to use. In the previous section a few examples are given which show that there are no good heuristics on which path or which branch to include or not. It was however possible to calculate the benefit of every edge. This can be used to find the subsets most likely to form an optimal solution.

Simply taking the Z edges with the most benefit will however not reduce the objective function with that sum of benefit. When adding one edge to the breaker edges, it is clear that adding for instance a neighboring edge after this will give a smaller benefit than was calculated before.

The attributes *freq* and *imp* are calculated in Algorithm 4.2, *impsegment* in Algortihm 5.4. The attributes *freq* and *impsegment* depend on the set of breaker edges, so adding an edge to this set changes the values of these attributes for certain edges. The way in which the attribute *imp* is calculated shows that this does not depend on B. These attributes are used to calculate the benefit. To use these attributes after adding edges to the set of breaker edges, there has to be a way to update these attributes. Then it is possible to calculate the benefit of every edge after expanding B.

Suppose we add edge e to B, we will see how the attributes change by adding this edge. *freq* is calculated by summing from leaves to root, but stopping at every breaker edge. The only change regarding this attribute after adding e is that we want to stop summing at e as well. So instead of adding *freq*(e) to the *freq* of the edge above e, we want to add 0 as contribution for edge e. Then the value of *freq* above e is *freq*(e) less, which means that the edge above that one also has *freq*(e) less. This continues until a breaker edge is encountered, and the summing stops again. We see that the attribute *freq* changes for all edges in the path from e to and including the next breaker edge with exactly -*freq*(e).

The attribute *impsegment* is equal for every edge in the same segment. Adding e to B defines a new segment below e. The segment that e was previously a part of, which is now above e, still has the same impact, so the *impsegment* remains the same for those edges. That makes it clear that *impsegment* only changes for the edges in the segment defined by e, and *impsegment* was exactly the *imp* of the breaker edge defining that segment. So all edges in the segment defined by e get *impsegment* equal to *imp*(e).

An important remark is that for some edges *freq* is smaller, and for some other edges *impsegment* is smaller. This means that the benefit $\textit{freq} \cdot (\textit{impsegment} - \textit{imp})$ can never increase by adding an edge to B.

These updates of the attributes after adding e to B are given in the pseudo-algorithm below.

Algorithm 5.7: Update the attributes freq, imp and impsegment after adding edge to B

```
Data: Configuration graph $G$ with attributes freq, imp and impsegment, edge
      $e \in E(G) \setminus B$
Result: Configuration graph $G$ with updated attributes
1 $p$ = shortest path from the tail of $e$ using incoming edges to and including one breaker
    edge;
2 $freq$(edges on $p$) = $freq$(edges on $p$) - $freq(e)$;
3 $breaker(e)$ = TRUE ;
4 $H$ = delete edges with $breaker$ from $G$;
5 comp = component of $H$ that contains the head of $e$;
6 $impsegment$(edges in comp) = $imp(e)$;
7 $impsegment(e) = imp(e)$;
8 return $G$;
```

We want to find the subset that together has the greatest benefit, so this method will help find edges with large benefits. We will see in the following subsections how this method can be used.

5.3.4 Greedy algorithm

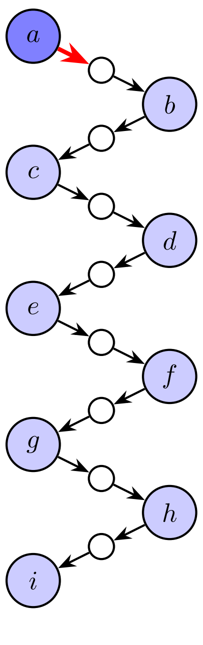
The easiest method to use the benefit calculation of all edges is by simply adding the edge with the greatest benefit, updating and repeating this until Z edges are added (the last update is of course not necessary). This algorithm, choosing at each stage a locally optimal addition, is called a greedy algorithm. This is a deterministic algorithm that is fast and gives a local optimum.

Algorithm 5.8: Greedy algorithm to find local optimum C' by giving subset of edges to add to B

```
Data: Configuration graph $G$ with attributes freq, imp and impsegment, $Z \in \mathbb{N}$
Result: Subset of edges that, added to $B$, give the local optimum $C'$
if $Z = 0$ then
    return empty list;
end
$l$ = result of Algorithm 5.5 with input $G$, the benefit for every edge;
best = row of $l$ with max benefit;
edges = list with edge of best;
result = benefit of best;
while $Z \neq 1$ do
    $G$ = output of Algorithm 5.7 with input $G$ and edge of best;
    $Z = Z - 1$;
    $l$ = result of Algorithm 5.5 with input $G$, the benefit for every edge;
    best = row of $l$ with max benefit;
    edges = append edges with edge of best;
    result = result + benefit of best;
end
return list of edges, result;
```

Every time a breaker edge e is added, the possible benefits of certain other edges can change. Section 5.3.3 shows that these edges are the ones on the path from e to its previous closest breaker edge and all edges in the segment defined by e. It is possible that an optimal solution contains two of these edges, but not e. In that case, the greedy algorithm does not give an optimal solution. The following example illustrates this.

Example 5.9. Suppose we have the following simple example: a path with 17 vertices, $A = \{a\}$, $B = \{\{a, ab\}\}$ and $O = \emptyset$. Let $v_i = 10$ for all $i \in V_S \setminus A$ and $w_i = 0.1$ for all $i \in V_D$. The table gives the benefit of all edges, and the benefits if (e, ef) or (d, de) is added to B.



edge	**benefit**	**benefit with** (e, ef)	**benefit with** (d, de)
(a, ab)	0	0	0
(ab, b)	0	0	0
(b, bc)	7	3	2
(bc, c)	6	2	1
(c, cd)	12	**4**	2
(cd, d)	10	2	0
(d, de)	15	3	0
(de, e)	12	0	0
(e, ef)	**16**	0	4
(ef, f)	12	0	3
(f, fg)	15	3	**6**
(fg, g)	10	2	4
(g, gh)	12	**4**	**6**
(gh, h)	6	2	3
(h, hi)	7	3	4
(hi, i)	0	0	0

Table 5.1: Table of edges with their benefits

Suppose we want to add two edge to B. We see that the greedy algorithm would first add edge (e, ef) with a benefit of 16 and then edge (c, cd) or (g, gh) with a benefit of 4. The reduction caused by the solution given by the greedy algorithm is therefore 20. However, if a second best edge, (d, de), is added first with a benefit of 15, then edge (f, fg) or (g, gh) can be added with a benefit of 6. This is a solution that causes a reduction of 21. This example shows that the greedy algorithm does not always give an optimal solution.

5.3.5 Branch and bound

Apart from the exclusion of outgoing edges from elements of V_D, there are no good heuristics on which paths or edges in paths to exclude from the possibilities. The branch and bound algorithm provides a method to exclude other edges or subsets, by calculating a bound on the benefits of the subsets. The edges of the configuration graph are ordered by their benefit, and the algorithm follows branches of choices between adding an edge to the solution or not, in the order given by their benefits.

By following all branches, eventually all subsets are considered. Starting with the best benefits ensures that the solutions with the greatest chance to be an optimal solution are considered first. The algorithm that is used for this problem uses a depth-first search, and is updated after every step because of the variable benefits. The first solution that is calculated is therefore the same as the result of the greedy algorithm. Using a recursive algorithm the option of including the first edge is calculated by adding this edge and recursively finding an optimal solution with $Z - 1$ edges added to this updated graph, then the first edge is excluded from the possible additions and the second best edge is included. This continues until it is certain that an optimal solution has already been found.

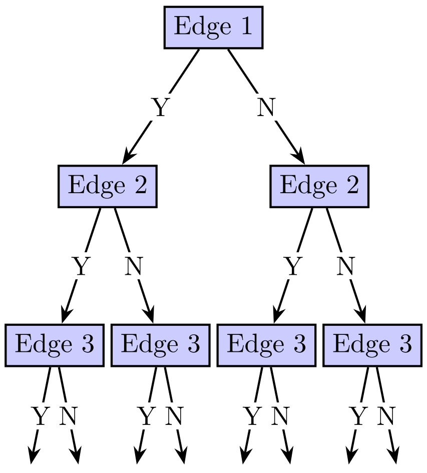


Figure 5.9: Tree of standard branch and bound algorithm

To be certain of this it is not necessary to calculate the objective function for all possible subsets, because of the fact that the benefits can never increase after adding edges to the set of breaker edges. This fact is seen from Algorithm 5.7, because the attributes *freq* and *impsegment* are both decreasing, while *imp* remains the same.

The bound part of this branch and bound variant is therefore given by the sum of the benefits of the Z best of the remaining edges, without updating. It is certain that the reduction of the objective function by adding these edges to the breaker edges is equal to or smaller than that sum, because after adding edges, the benefits of certain other edges decrease. The sum of the benefits without updating should therefore be greater than the actual best solution found so far, otherwise the real benefit of this set of edges can never be greater than the benefit of the solution already found. When this condition is violated, the algorithm can stop executing this branch, because even the best possible solution for these remaining edges has a smaller result than the solution found so far.

The algorithm uses a list of edges called processed, which is filled with the edges that are already processed. That makes sure that combination e_2 and e_1 is not examined when e_1 and then e_2 has already been processed. This list is initiated as all edges that are outgoing edges of the vertices in V_D, because we can find an optimal solution with the remainder of the edges according to Corollary 5.8. This eliminates a lot of possibilities.

This algorithm is deterministic and always gives the global optimum, because it processes every possible solution except for the ones for which it is certain that they will not give an optimal solution.

Algorithm 5.9: Branch and bound algorithm to find global optimum C' by giving subset of edges to add to B

```
Data: Configuration graph $G$ with attributes freq, imp and impsegment, $Z \in \mathbb{N}$,
      processed list of edges, bound $\in \mathbb{N}$
Result: Subset of edges that, added to $B$, give the global optimum $C'$
if $Z = 0$ then
    return edges = empty list, reduction = 0;
end
edges = empty list;
$l$ = result of Algorithm 5.5 with input $G$, the benefit for every edge;
$l$ = remove rows with edges in processed from $l$;
$l$ = $l$ ordered by benefit;
append processed with edge of $l[1]$;
if $Z = 1$ then
    append edges with edge of $l[1]$;
    result = benefit of $l[1]$;
end
else
    result = 0;
    limit = sum of benefit of $l[1]$ through $l[Z]$;
    while limit > bound do
        $H$ = add edge of $l[1]$ to $G$ and update attributes;
        temp = recursive call to function with $H, Z - 1$, processed, bound - benefit of $l[1]$;
        if result < result of temp + benefit of $l[1]$ then
            result = result of temp + benefit of $l[1]$;
            bound = max(bound,result);
            edges = append edges of temp with edge of $l[1]$;
        end
        $l$ = $l$ without rows with edges in processed;
        append processed with edge of $l[1]$;
        limit = sum of benefit of $l[1]$ through $l[Z]$;
    end
end
return edges,result;
```

5.3.6 Results

The greedy and branch and bound algorithms are tested for network 1, given in Chapter 2.

Network 1

This network consists of all stations supplied by one HV substation. Let C be the current configuration, which is given, this consists of 416 vertices and 415 edges between them and 65 breaker edges.

Executing Algorithm 5.7 takes 0.05 seconds. The following table gives the performance of Algorithm 5.8 and Algorithm 5.9.

Number	1	2	3	4	5	6	7	8	9	10
Greedy	0.006	0.07	0.134	0.199	0.234	0.343	0.416	0.458	0.502	0.608
Branch and bound	0.423	0.546	1.674	1.784	4.45	12.318	16.96	31.052	40.573	65.631

Table 5.2: Table with time in seconds taken by the different algorithms for different Z

Conclusion

In all of these cases the greedy and branch and bound algorithm both give the same result, but it was shown that the greedy algorithm does not always give an optimal solution. It is clear that the time the greedy algorithm takes grows linear with Z, while the execution time of the branch and bound algorithm grows exponentially with Z.

5.4 Breaker constraint

The algorithms that were treated in the previous sections find optimal solutions that satisfy the cost and capacity constraints, but by adding breaker edges they could still violate the breaker constraint. In this section this breaker constraint will be checked to find an optimal feasible solution.

By assumption the current configuration satisfies all constraints, so we know that in the current configuration every path has at most three breaker edges. We can use that to only check the breaker constraint at the parts of the graph where additions are made.

Approach

In the minimization problem we want to find an optimal feasible solution. This can be done by either first checking the breaker constraint to determine all feasible solutions and then finding an optimal solution among them, or by finding an optimal solution and then checking whether or not it satisfies the constraint. The first option requires too much space and may take a lot of time. In particular with the algorithms that we have defined, the greedy and branch and bound algorithms, it is not that useful because those algorithms also exclude many possibilities.

The second option is more efficient because that only checks the breaker constraint for that solution, which can be done by only investigating the added breaker edges and ignoring the rest of the graph. It becomes more difficult when an optimal solution does not satisfy this constraint, because then a different solution has to be calculated and it is likely that this does not differ so much from the previous solution. Therefore it is very likely that the following solutions will also violate the breaker constraint, which results in a lot of repetitions of this process.

Remark 5.10. In the previous section it became clear that the reduction caused by adding two breaker edges can be less than their added benefit if they lie on a path in the same segment. Therefore it is often the case that multiple breakers are not added in the same segment, which also lowers the chance of optimal solutions containing more than three breaker edges on the same path.

Because of this fact, the chance of optimal solutions violating the constraint depends on the ratio of number of segments and maximum number of additions. In practical applications, the maximum number of additions will probably be not so much larger than the number of segments, and most likely less than that.

The approach that we take in this section is therefore something in between these two options. The greedy and branch and bound algorithms both add the breakers one at the time, so we use that to check the solution at every addition, instead of checking the solution afterwards. That way it is easier to find the next best solution, and the breaker constraint only has to be checked in a very small part of the graph.

5.4.1 Checking a configuration graph

The assumption is made that the current configuration graph satisfies all constraints. But it is still useful to have an algorithm that can check the breaker constraint for a given configuration graph.

The algorithm checking the breaker constraint is given below. For all paths, the number of breaker edges cannot be larger than 3. The paths can be found from the leaves, but it is also possible to check paths from the breaker edges to the roots. From the leaves to the roots it is clear that in a forest all possible paths are contained in these paths. With the other possibility, checking paths from the breaker edges to the roots, it is also certain that if there exists a path with more than three breaker edges then this is found. When checking the path from the breaker edge furthest away from the root the other breaker edges are encountered. By checking the path from every breaker edge all paths with more three breaker edges are found, so it terminates with a negative result if such a path is encountered and with a positive result if the paths from all breaker edges are processed and a violating path is not found.

In this section the paths are found from the breaker edges, because these are known while the leaves first have to be identified.

Algorithm 5.10: Check the breaker constraint on a configuration graph

```
Data: Configuration graph $G_C$
Result: True if and only if $G_C$ satisfies the breaker constraint
1 for all breaker edges $e$ do
2     p = shortest path from tail of $e$ to a root in $G_C$;
3     if $p$ contains 3 breaker edges then
4         return false;
5     end
6 end
7 return true;
```

5.4.2 Adding a breaker edge

In this subsection we consider one possible addition, if this addition satisfies the breaker constraint then it is added, otherwise the next best edge will be investigated.

We assume that we have a configuration that satisfies the breaker constraint, and adding the new breaker edge e causes the violation of this constraint. It is clear that all paths violating this constraint have to contain e, otherwise they already contained four or more breaker edges. So what has to be checked is, for all of those paths, whether or not they already contain three breaker edges. If e is then added as a breaker edge, this creates a path containing four breaker edges.

Three methods are considered to check these paths:

1. Find all paths containing e and check if any of them already contains three breaker edges.

2. Initiate the algorithm by identifying all edges of the graph that are elements of paths with three breaker edges. All of those edges are excluded from the possibilities. When adding a new breaker edge creates new paths with three breakers, the edges of these paths are added to the excluded set.

3. Initiate the algorithm by assigning 0, 1 or 2 to the edges, indicating how many breakers can be added on the paths containing these edges. 2 is the maximum because the first edge of a path is always a breaker edge. These attributes are updated when a new breaker edge is added. All edges with a 0 assigned to them are excluded from the possibilities.

All of these methods need to find all paths containing edge e when examining the possibility of adding e to the breaker edges, either for checking the number of breakers or to do an update. The difference lies in the fact that method 2 and 3 only need to find them if the edge can be added to the breaker edges, while method 1 performs this search with the possibility that the edge cannot be added. The disadvantage of methods 2 and 3 is that they need an initialization of the entire graph. In this chapter we therefore choose to use method 1.

Algorithm 5.11: Check whether or not e is a feasible solution

```
Data: Configuration graph $G_C$, edge $e \in E(G_C)$
Result: Feasibility of $e$
1  pathabove = find shortest path from tail of $e$ to a root;
2  breakerabove = number of breaker edges on pathabove;
3  if breakerabove = 3 then
4      return False;
5  end
6  outtree = all edges from head of $e$ to leaves;
7  breakerbelow = number of breaker edges in outtree;
8  if breakerabove = 2 then
9      if breakerbelow $\geq 1$ then
10         return False;
11     end
12 else
13     for breaker edge $e'$ in outtree do
14         pathbetween = find shortest path from tail of $e'$ to head of $e$;
15         if pathbetween contains a breaker edge then
16             return False;
17         end
18     end
19 end
20 return True;
```

All paths containing e can contain at most two other breaker edges. All of these paths are the paths from leaves to root containing e. Because the path from e to a root is unique, all these paths overlap from e to the root. So the path p from the tail of e to a root is checked first. We know that this path p contains at least one breaker edge from the root, so we will get one, two or three breaker edges. If there are three breaker edges above e, then it is clear that e is not feasible. Otherwise it is necessary to look at the out-tree of the head of e. If there are two breaker edges in the path p, then e will be the third, so no other edge below e can be a breaker edge. This means that none of the edges between vertices in the out-tree can be a breaker edge, otherwise e is not a feasible solution. In the last case, where there is one breaker edge in the path p, all paths below e can have at most one breaker edge. This is checked by finding the paths from every breaker edge below e to the head of e and checking whether that breaker edge is the only one on the path. If that is the case then e is feasible, but if there exists a path to e with two breaker edges, than together adding e would create a path with four breaker edges.

5.4.3 Adding multiple breaker edges

It is possible to check the feasibility of a solution which adds multiple breaker edges, by checking all paths containing one of these edges. This can be done by executing Algorithm 5.11 for all added edges, while it outputs positive results.

However, if the result of the greedy or branch and bound algorithm gives a negative result, which means that it is not an optimal feasible solution, it is hard to calculate the next best solution. Therefore we don't check the feasibility of the entire set afterwards, but check the feasibility at every addition of one edge. Both of the algorithms add the edges one at the time, which provides the ability to check it before adding. The result of the algorithms will then always be a feasible solution.

The algorithms are modified to include Algorithm 5.11 before the addition of an edge. Nothing changes if it gives a positive result, otherwise this edge is skipped and the edge with the next best benefit is considered. The branch and bound algorithm will still process every feasible possibility, so with this adjustment, it gives an optimal feasible solution.

5.5 Alternative objective function

In Section 3.6 other objective functions were introduced. The algorithms that are given in this chapter so far are about minimizing $R(C)$, but in this section we will find algorithms minimizing the alternative objective functions. These functions were defined with the SAIDI function, but can also be stated with the new function R:

a) $\min R(C') \cdot p + S(C')$

b) $\min \frac{S(C')}{R(C)-R(C')}$

In this chapter only breaker edges can be added, which all have the same cost. So we assume that the addition of one breaker edge has cost K.

The alternative functions are minimized for the reconfigurations with only addition of breaker edges. Where possible, the algorithms given so far, the greedy and branch and bound algorithm, are adjusted so they minimize the alternative objective function.

5.5.1 Place only if beneficial

The first alternative objective function is: $\min R(C') \cdot p + S(C')$. This function assigns a cost of p to every unit of R. A breaker edge is then only added if it is beneficial, that is if the reduction of R is at least $\frac{K}{p}$.

We already know that the benefit of edges can only decrease by adding other breaker edges. This means that the more breakers are already placed, the less reduction can still be obtained. At some point this reduction could be less than $\frac{K}{p}$, which means that the objective will have a lower result if another edge is added to the breaker edges.

Example 5.11. The following table gives example results for the optimal result of $R(C)$ with corresponding cost, and the outcome of this alternative objective function for different values of

p. In this example reduction per cost unit decrease, which means that every next unit of cost reduces less than the last, the difference between two adjacent results of $R(C)$ is decreasing.

Costs	0	1	2	3	4	5	6	7	8
$R(C)$	30	25	21	18.5	16	14	12.25	11	10.5
$p = 2$	60	51	44	40	36	33	30.5	**29**	**29**
$p = 1.5$	45	38.5	33.5	30.75	28	26	24.375	**23.5**	23.75
$p = 1$	30	26	23	21.5	20	19	18.25	**18**	18.5
$p = 0.5$	15	13.5	12.5	12.25	**12**	**12**	12.125	12.5	13.25

Table 5.3: Example for cost, $R(C)$ and the result of the objective function with different p

It is clear from the table that if the reduction per cost unit decrease, then an optimal solution lies at the point where the difference in reduction is the same as p. The next additions cost more than they gain, so it is better to stop then, while before that point it is possible to gain more than it costs, so then you want to continue. We also see that when adding one more reduces exactly $\frac{1}{p}$ extra, then the result of the objective function is the same whether it is added or not.

We will use the fact that we want to keep adding breaker edges while it is beneficial to adjust the greedy and branch and bound algorithms to minimize this alternative objective function.

Greedy algorithm

The adjustment of the greedy algorithm is simple. The steps that are repeated in the algorithm are: calculate the benefit of every edge, add the edge with the greatest benefit. These steps are repeated until there is a solution containing Z additions to the breaker edges. The only adjustment that we need to make in this algorithm is to check whether the greatest benefit is still larger than $\frac{K}{p}$ and only continue adding if that is the case, otherwise the result is the list of additions so far.

Branch and bound algorithm

In the branch and bound algorithm there is one part that complicates this approach. Again, we only continue adding breaker edges if their benefit is larger than $\frac{K}{p}$, so we include that condition before adding an edge.

Suppose $Z = 10$, the algorithm is following a branch of possibilities and after 8 additions the benefits of all edges are not larger than $\frac{K}{p}$. Then this can be the best solution found so far, with a reduction of the 8 added breaker edges. The difficulty now lies in the bound created by this result. The upper bound on a branch is the sum of possible reduction for 10 additions, which is often larger than the reduction caused by 8 additions. Therefore this bound is not tight enough and causes the algorithm to process many more possibilities than necessary.

The solution lies in the function that we are considering. Previously, maximizing the reduction was the same as minimizing the objective, because it would always add Z breaker edges. Now we want to keep the possibility to add Z breaker edges, but also need a way to compare that possible reduction to solutions found that add less breaker edges. This is done by not maximizing the reduction, but by maximizing the reduction $+\frac{K}{p} \cdot (Z - x)$, where x is the number of additions. Maximizing that function will correspond with minimizing the objective.

Algorithm 5.12: Branch and bound algorithm to find global optimum C' of the alternative objective by giving subset of edges to add to B

```
Data: Configuration graph $G$ with attributes freq, imp and impsegment, $Z \in \mathbb{N}$, $p$, $K$,
      processed list of edges, bound $\in \mathbb{N}$
Result: Subset of edges that, added to $B$, give the global optimum $C'$ of the alternative
        objective function
edges = empty list;
$l$ = result of Algorithm 5.5 with input $G$, the benefit for every edge;
$l$ = remove rows with edges in processed from $l$ and rows with benefit $\leq p$;
if $l$ is empty then
    return edges = empty list, reduction = $\frac{K}{p} \cdot Z$;
end
$l = l$ ordered by benefit;
append processed with edge of $l[1]$;
if $Z = 1$ then
    append edges with edge of $l[1]$;
    result = benefit of $l[1]$;
end
else
    result = 0;
    if nrow($l$) $\geq Z$ then
        limit = sum of benefit of $l[1]$ through $l[Z]$;
    end
    else
        limit = sum of benefits of $l + \frac{K}{p} \cdot (Z - \text{nrow}(l))$;
    end
    while limit > bound do
        $H$ = add edge of $l[1]$ to $G$ and update attributes;
        temp = recursive call to function with $H, Z-1, p, K$, processed, bound - benefit of
          $l[1]$;
        if result < result of temp + benefit of $l[1]$ then
            result = result of temp + benefit of $l[1]$;
            bound = max(bound,result);
            edges = append edges of temp with edge of $l[1]$;
        end
        $l = l$ without rows with edges in processed;
        if nrow($l$) $\neq 0$ then
            append processed with edge of $l[1]$;
            if nrow($l$) $\geq Z$ then
                limit = sum of benefit of $l[1]$ through $l[Z]$;
            end
            else
                limit = sum of benefits of $l = \frac{K}{p} \cdot (Z - \text{nrow}(l))$;
            end
        end
        else
            limit = 0;
        end
    end
end
return edges,result;
```

5.5.2 Cost/benefit function

The other alternative objective function minimizes the cost needed per reduction: $\min \frac{S(C')}{R(C)-R(C')}$. In the case where we only add breaker edges, this objective will always have an optimal solution which adds only one breaker edge.

Lemma 5.12. *For any $Z \in \mathbb{N}$, the edge with the largest benefit is an optimal solution of this minimization problem.*

Proof. Suppose $e \notin B$ is the edge with the largest benefit q. The benefit of every other addition is then at most q. This means that for all x, every combination of x additions has a reduction of at most $q \cdot x$. The result for the addition of only e is $\frac{1}{q}$, while for any combination of x additions it is at least $\frac{x}{q \cdot x} = \frac{1}{q}$. We see that adding e gives an optimal solution. $\square$

The minimization of this objective function is solved by using an algorithm from Section 5.2 to find the edge with the best benefit.

Chapter 6

Optimal locations of openings

In this chapter the possibility of changing the set of openings is considered. We start with the set of openings in the current configuration and relocate certain openings. We will prove in this chapter that for a given graph and set of roots, the size of the set of openings is equal for all radial configurations. We have tried the same approaches as in the previous chapter. However, it turned out that the methods that worked well there, are less useful when considering openings. In this chapter we therefore give the ideas and show where the problems occur when trying to use them with openings. The proposed algorithms have not been implemented, because there was not enough time during this project. There are therefore no results on execution times of the algorithms.

First we will examine the constraints and show which constraints are important for this problem. We start again with only one adjustment, so where one opening can be changed. Then some definitions are given that can be used to find radial configurations. It is essential to be able to find radial configurations when solving any problem regarding the relocation of openings.

Changing the set of openings also changes the orientation of certain edges in the configuration graph. Therefore the orientation is no longer very important when finding paths in the graph. So when we consider paths in this chapter, these are undirected paths.

6.1 Constraints

In this chapter we only consider changing the set O. So, for current configuration $C = (G, A, B, O)$ the problem becomes: $\min R(C')$ for $C' = (G, A, B, O')$ with

1. C' radial
2. C' satisfies the breaker constraint
3. The reconfiguration satisfies the cost and capacity constraints

We are again focusing on only one type of change, so the cost and capacity constraint can be rewritten to $y \leq Z$, where $y = \#(O' \setminus O)$ and $Z = \min(\frac{T}{T_o}, \frac{S}{S_o})$.

We need to find sets O' such that the configuration is radial, which can be done by using the definitions of spanning trees or cycle basis, and we need to keep checking the breaker constraint for these radial configurations.

Radial reconfigurations

In this chapter we will not start by investigating all possible subsets of the edges, because a large part of these subsets will not create a radial configuration. This is shown in the example below. We will therefore focus on finding the radial reconfigurations first, and then trying to find an optimal solution among these.

Example 6.1. Suppose we have the following graph G. We want to assign edges to the set of openings to create a radial configuration. It has been calculated that there are 408 of such sets. All of these sets contain exactly three edges. We want to compare the number of radial configurations to the total possibilities for subsets of three edges. The graph has 22 edges, so the number of subsets containing 3 edges is equal to $\binom{22}{3} = 1540$. We see that the probability of the subset creating a radial configuration is not very large.

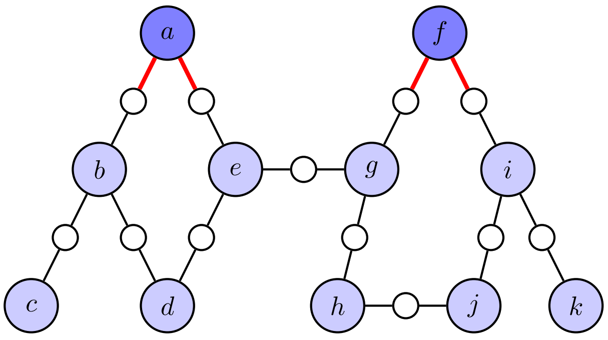


Figure 6.1: Example graph G with breaker edges

6.2 Changing 1 opening

We start again with the case where $Z = 1$, which means that we may only remove one element from O. So we start with O and for all elements of O, delete that element from O and see which reconfigurations can be made. In this section we will find these possible reconfigurations and try to find methods to determine an optimal solution.

6.2.1 Solution space

By deleting 1 element from O, the corresponding edge is no longer removed in the configuration graph. Suppose we delete edge $\{i, j\}$ from O. The current configuration is radial, which means that i is connected to some $a \in A$ and j is connected to some $b \in A$, where it is possible that $a = b$. Adding edge $\{i, j\}$ to the configuration graph creates a path between a and b and if $a \neq b$ this means that there are two elements of A in the same component, and if $a = b$ this means that the graph contains a cycle. In either of these cases the graph is no longer radial.

The possible radial configurations starting from this graph are given in the following lemma:

Lemma 6.2. *All possibilities to create a radial graph again are to remove one of the edges on the path from i to a and from j to b, without edges that occur in both.*

Proof. We distinguish between two cases:

- Case $a = b$: This means that i and j are in the same component. The other components of the configuration graph have not been changed, so they still satisfy the radiality conditions. In the component of i and j there is still exactly one element of A, so it has to remain connected. There exists a path P_1 from i to a and a path P_2 from j to a. If they do not overlap, then adding $\{i, j\}$ creates a cycle containing i, j and a. Suppose these paths do overlap, and h is the first vertex on both paths. There is a unique path from h to a so this path is contained in P_1 and P_2. There is a cycle formed by the path from i to h, j to h and $\{i, j\}$. These are the edges on P_1 and P_2 without their overlapping edges. Any one of the edges on the cycle can be removed to create a tree, while all other edges have to remain to ensure that the component is still connected.

- Case $a \neq b$: In this case i and j lie in different components, which means that the configuration graph is still a forest. The component that now contains both a and b should be cut such that two trees with one element of A in each is created. Removing one of the edges not on the path from a to b will create two trees, with a and b in one, and no element of A in the other, so these edges have to remain in the configuration graph. For all edges on the path from a to b it holds that removing one creates two trees where there is no path between a and b, so both of these trees contain exactly one element of A.

In both cases all possibilities to create a radial graph are removing any one of the edges on the path from i to a and from j to b, without edges that occur in both. □

Example 6.3. We have the graph G and configuration graph G_C for $C = (G, A, B, O)$ with $A = \{a\}, B = \{\{a, ab\}, \{c, ce\}, \{a, af\}\}$ and $O = \{\{e, ge\}\}$) introduced in chapter 3.

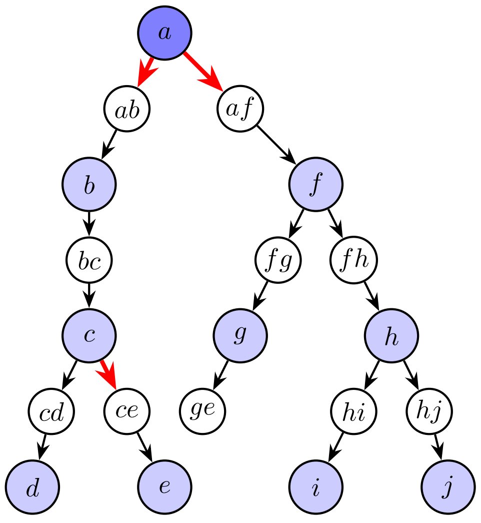


Figure 6.2: Configuration graph G_C

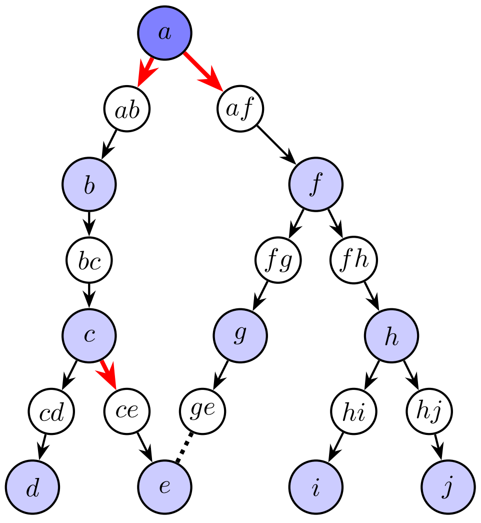


Figure 6.3: Deleting an opening

Suppose we delete $\{e, ge\}$ from O, which in this small case leaves the empty set. Adding this edge to this configuration graph creates a cycle, which is this edge together with the path from e to a and the path from ge to a. Creating an opening in any one of the edges on this cycle creates again a radial configuration graph. The configurations with one of these edges as opening are the possible solutions.

So all possible reconfigurations can be made by removing an edge from the path from i to a or from j to b without edges that occur in both. These reconfigurations are radial and satisfy $\#(O' \setminus O) \leq 1$. The solution space consists of these reconfigurations that also satisfy the breaker constraint. This constraint can be checked using the algorithms defined in section 5.4.

The solution space then consists of, for all elements of O, the combination of that element, to be removed from O, together with the possible feasible additions of an edge.

6.2.2 Strategies

Now that we know which solutions create radial configurations, we can design algorithms to find an optimal solution among them. Again we investigate the brute force method, the possibility of calculating the benefit, using the attributes *imp* and *freq*, the accumulated sums of the values and weights respectively, and heuristics.

Brute force

The brute force method simply calculates the result of the objective function for every element of the solution space. The cardinality of the solution space is $O(|E(G)|^2)$, so this is acceptable, but will blow up when considering more than a single change.

Brute force; Adjusting reconfigurations

The creation of the configuration graph of the reconfiguration can be done by starting with G and A and incorporating B and O'. It is however more efficient to start with the configuration graph G_C, add the edges in $O \setminus O'$, delete the edges in $O' \setminus O$ and adjust the orientation of the arrows accordingly. We can also adjust the configuration graph of a reconfiguration to get the configuration graph of another reconfiguration, which is useful because that makes it possible to 'walk' along the paths to a and b. The configuration graphs of the possible solutions are therefore created in this way.

Reduction

Instead of calculating the result of the objective function for every possible radial solution, we investigate the possibility of calculating the reduction in this result. If that is possible, then it is not necessary to create the configuration graph for every feasible solution. In this subsection we will use the attributes *freq* and *imp* that were defined in Chapter 4.

We will examine the case where the only breaker edges are the edges incident to a root. If we have the case where the two paths to the root overlap, then they are part of the same segment. The sum of weights and the impact of the segment will not change when relocating that opening. The benefits are then all 0. We will therefore now treat the case where the opening connects two different segments. By changing an opening, certain vertices and edges are located on a different branch than before. That means that they are removed from one segment and added to another. These vertices and edges are the ones that are behind the new net opening, because they are no longer supplied through that edge.

Lemma 6.4. *Suppose we have a current configuration. In this configuration we want to close opening e_{old} and add e_{new} to the openings. Let W be the sum of weights of the vertices after e_{new} and I the sum of their values. Suppose e_1 defines the segment containing the tail of e_{old} in the current configuration, and e_2 defines the segment containing the head of e_{old}. Then the reduction of adding e_{new} to the openings is equal to*

$$(\mathit{freq}(e_2) \cdot I) + (\mathit{imp}(e_2) \cdot W) - (\mathit{freq}(e_1) \cdot I) - (\mathit{imp}(e_1) \cdot W) - (2 \cdot W \cdot I).$$

Proof. The current contribution to the objective function of e_1 is $\mathit{freq}(e_1) \cdot \mathit{imp}(e_1)$, and similar for e_2. The current closest breaker edge of the vertices after e_{new} is e_2. When e_{old} is closed and e_{new} is opened, then these vertices are supplied through e_{old} with closest breaker edge e_1. The rest of the configuration graph will remain the same. In that new configuration the contribution of e_1 is equal to $(\mathit{freq}(e_1)+W)\cdot(\mathit{imp}(e_1)+I) = (\mathit{freq}(e_1)\cdot\mathit{imp}(e_1)) + (\mathit{freq}(e_1)\cdot I) + (\mathit{imp}(e_1)\cdot W) + (W\cdot I)$. We see that this adds $(\mathit{freq}(e_1)\cdot I) + (\mathit{imp}(e_1)\cdot W) + (W\cdot I)$ to the current contribution of e_1. Similarly for e_2, this adds $-(\mathit{freq}(e_2)\cdot I) - (\mathit{imp}(e_2)\cdot W) + (W\cdot I)$. Putting these together we get the reduction of

$$-((\mathit{freq}(e_1)\cdot I) + (\mathit{imp}(e_1)\cdot W) + (W\cdot I) - (\mathit{freq}(e_2)\cdot I) - (\mathit{imp}(e_2)\cdot W) + (W\cdot I)).$$

□

We see that in the case where we only have to consider breaker edges incident to the root, we can use the attributes *freq* and *imp* to calculate the benefit of opening an edge, given the edge that will be closed. Especially since W is exactly $\mathit{freq}(e_{\text{new}})$ and I is $\mathit{imp}(e_{\text{new}})$. For a given opening that will be closed, $\mathit{freq}(e_1), \mathit{imp}(e_1), \mathit{freq}(e_2)$ and $\mathit{imp}(e_2)$ are constant. The benefits of an edge can therefore be calculated using only the attributes on that edge.

A problem arises when there is an additional breaker edge on the path from a to b. Even if one of the ends of the old opening lies in the same segment as the new opening the lemma does not hold. When the new opening lies in another segment, which is the case if there is a breaker edge between the old opening and the new opening even more values have to be known. It is still possible to calculate the benefit of this new opening, but it needs to know where the breaker edges are and the attributes of all involved breaker edges. The advantage of not having to trace the graph no longer holds in this case. This means that this method does not have a clear advantage over calculating the reductions brute force.

Heuristics

We see that for every edge in O, all possible relocations lie on a path. The first idea for a useful heuristic is to see if the benefit along this path is monotone. The following example gives the benefits of all edges along a path and shows this is not the case.

Example 6.5. Suppose we have the configuration graph depicted in Figure 6.4. We see that the benefit along the path is not monotone everywhere. When we have seen the best benefit so far at $\{ij, j\}$, the benefits on the path after it is still not monotone.

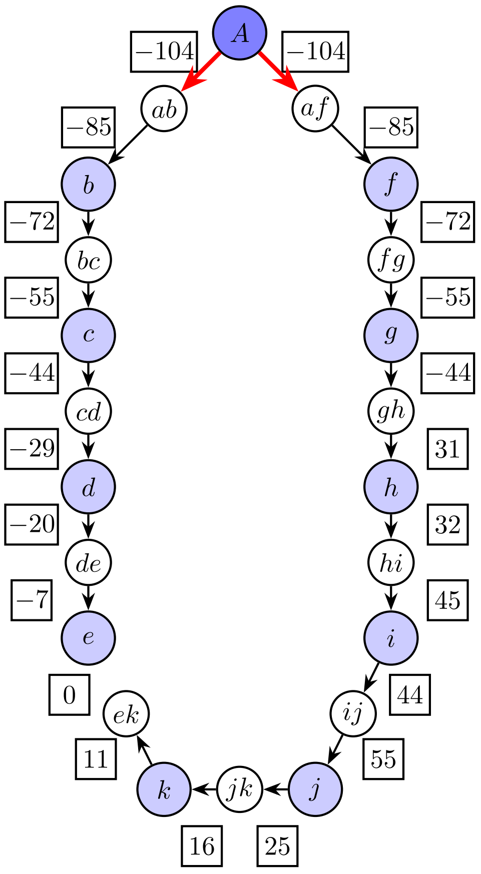


Figure 6.4: Configuration graph with one opening together with the benefit if that opening is relocated

This example also shows that if the current opening is at $\{hi, i\}$, then both directions will have a negative benefit. So choosing the direction of positive benefit is also not a good heuristic.

This is in the case with only breaker edges incident to the root. Additional breaker edges will only cause more variation in the benefits.

6.3 Radial solutions

In this section we will give important definitions regarding radial solutions. These can be used to find radial solutions after removing edges from the set of openings. Recall the definition of a radial configuration:

Definition 6.6. A graph is **A-radial** if it satisfies the following:

- The graph is a forest.
- Every component of the graph contains exactly one vertex from the set A.

The theses of H. Fritschy [7] and of M. van der Meulen [8] investigate the problem of finding radial solutions among other things. The definitions and strategies given in those theses are summarized in this section. They use a different model than in this thesis, but the creation of an opening is similar. It is simply the deletion of an edge. The difference is that they do not use information of the edges, which means they don't need the extra vertices representing distribution lines.

6.3.1 Spanning forest

We will use the definition of a spanning forest in this section. More information on spanning trees can be found in [5], Chapter 4.

Definition 6.7. A **spanning tree** of a graph G is a subgraph which is a tree and contains all vertices of G. A **spanning forest** is a subgraph of a graph G which consists of a spanning tree for every component of G.

In this model, the creation of a configuration graph G_C from a graph G is done by adding attributes and deleting edges. To create an A-radial configuration graph, all cycles have to be removed. This relates to spanning trees. Indeed, if A contains only one vertex, then by the assumption that every component of G contains at least one element of A we see that G consists of one component. All A-radial configuration graphs correspond to all spanning trees of G.

If A contains more vertices, then the second property also has to be considered. G_C is radial if it does not contain cycles and every vertex is included in a component with exactly one vertex from A. This means that it is a spanning forest, but not every spanning forest is A-radial because it may not satisfy the second property.

If all paths between vertices from A are interrupted first, then the second property is satisfied and all maximal spanning forests in this graph are A-radial configurations. By finding all different combinations to create graphs satisfying the second property, it is possible to find all A-radial configuration graphs by considering all maximal spanning forests of all those combinations.

There is also a nicer solution available, which is given in the following subsection.

6.3.2 Merging the roots

In this subsection we will merge set A into one vertex. This idea is introduced in [7]. Every edge incident to a root is then incident to this vertex. We see that all branches remain the same, the only difference is that the vertices in A are removed and one vertex is added. This does not change anything in the result of the objective function, because there is no information lost of the vertices of A and all edges incident to a root are breaker edges. In this graph with merged roots, we see that any path between roots is a cycle.

All radial configurations of the graph now correspond to all spanning trees of the graph with merged roots.

Example 6.8. The following figures show a configuration graph and the configuration graph with merged roots.

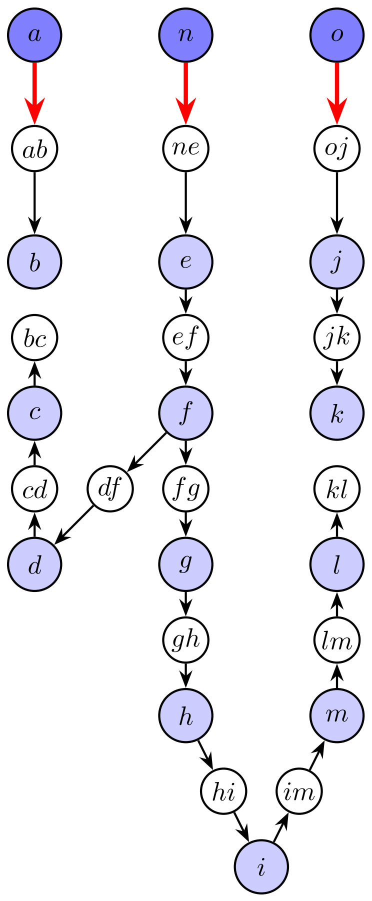


Figure 6.5: Configuration graph G_C

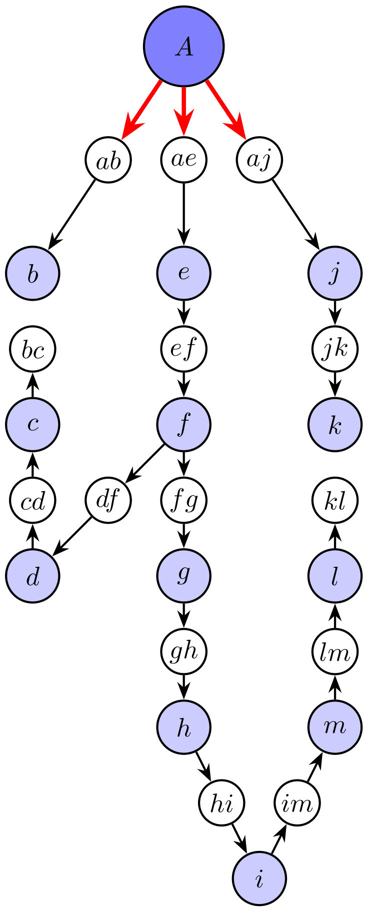


Figure 6.6: Configuration graph G_C with merged roots

Because there is almost no information on the vertices of A and all outgoing edges are breaker edges, everything that was done so far in this thesis behaves equally on a configuration graph with merged roots. The only real difference is that all radial configurations are now spanning trees of the underlying graph G with merged roots. From now on we will therefore work only with graphs with merged roots.

Using this fact it is easy to see that the size of the set of openings is always equal. To show this we use the following well-known lemma.

Lemma 6.9. *A tree of n vertices has $n - 1$ edges.*

Lemma 6.10. *Let $C = (G, A, B, O)$ be the current configuration. Suppose we remove x edges from O, which leaves O_1. Let $C' = (G, A, B, O_1 \cup O')$ be a reconfiguration of C. Then we have $\#(O') = x$.*

Proof. The current configuration C is radial, which means that G_C is a spanning tree of G with merged roots. Suppose G has n vertices, then G_C is a tree with n vertices. According to the previous lemma, the configuration graph then has $n-1$ edges. If we remove x edges from O, then that means that we add those edges to the configuration graph, which than contains $n - 1 + x$ edges. We know that all radial configuration graphs are spanning trees, so the configuration graph of a reconfiguration contains $n - 1$ edges. We see that we need to remove exactly x edges, so we add x edges to the set of openings. □

6.3.3 Cycle basis

A cycle basis can be used to find spanning trees. This is done in [8]. In this subsection we will summarize the important notions given in that thesis.

Definition 6.11. A **cycle basis** of a connected graph G is a set of cycles in G which are linearly independent and all other cycles can be constructed from these cycles. A set of cycles is linearly independent if no cycle can be constructed using the other cycles in this set.

In [8] it is shown that all spanning trees of a graph G remove one edge from every cycle in a cycle basis. This therefore gives a method to find the radial configurations. It is however not true that every combination of edges that takes one edge from every cycle in a cycle basis corresponds to a spanning tree. It is for instance possible that an edge is contained in multiple cycles. If we choose that edge for all those cycles, the number of edges that is removed is not large enough to create a tree.

However, it does give a lot less combinations to be checked than simply every combination of x edges, where x is the number of removed openings.

A cycle basis can be found using the following definition:

Definition 6.12. Let T be a spanning tree of G and let $e \in E(G) \setminus E(T)$. The **fundamental cycle** of e is the cycle in G that is formed by the path in T between the two endpoints of e and e itself. The collection of the fundamental cycles for every edge in $E(G) \setminus E(T)$ is a **fundamental cycle basis** of G.

Starting from a current configuration, we know every edge in G that is not in the configuration graph. We can use that to create a fundamental cycle basis of G.

6.4 Changing multiple openings

In this section we will show what has been tried to solve this problem for multiple openings. We have not found a very efficient way to solve this problem in the case where 1 opening is changed, so we want to find a method in which the number of times that the benefits have to be calculated is small. The idea for that method is to find the optimal location per opening. That way, the benefits are calculated once for every opening.

The problem with that idea is that the locations of other openings influence the optimal locations for one greatly. The following example shows that it depends on the order in which the openings are considered what the result is.

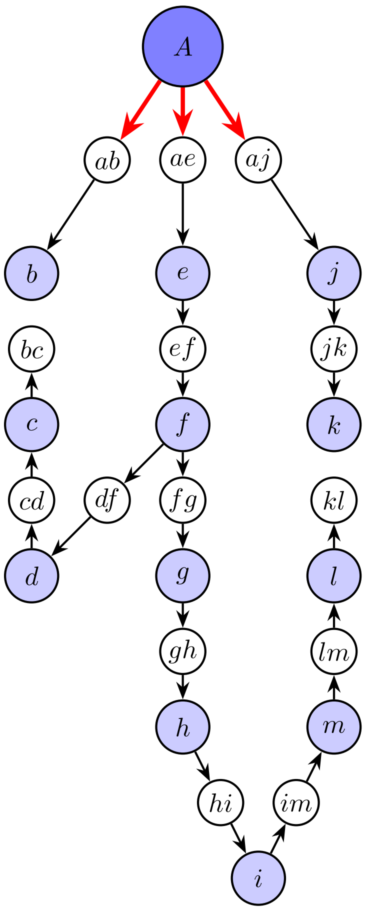


Figure 6.7: Configuration graph G_C

Example 6.13. Suppose we have this configuration graph G_C, with an opening between b and bc and one between k and kl. For all $i \in V_S \setminus A$ we have $v_i = 10$ and for all $i \in V_D$ we have $w_i = 0.1$. We will give the best result for changing the first and then the second, and the best result for the other way around. We see that $R(G_C) = 10 \cdot 0.1 + 90 \cdot 1.1 + 20 \cdot 0.2 = 104$.

- Close the opening between b and bc, this creates a cycle. The greatest benefit is at edge (f, df), with a benefit of 32. We open this edge and close the other opening. On the created cycle the greatest benefit is of 15, at edge (h, hi).

- Now we work the other way around: first close the opening between k and kl. On the cycle that is created the edge with the greatest benefit is (gh, h), with a benefit of 32. We open this edge and close the other opening. The greatest benefit is now 12 at edge (f, df).

We see that it depends on the other in which the openings are treated what the result will be. It is of course possible to keep going back and forth, if we only consider two openings. But when considering more openings, the number of operations will blow up if they all influence each other.

6.5 Further research

The ideas that have been tried in this thesis have not yielded positive results. It has been shown how the radial configurations can be found. We have however not found a way to give the optimal solution from these radial configurations. The suggestion for further research is to try approximation algorithms for this problem, because finding the exact global optimum will very likely need an unacceptable amount of time in practice.

The thesis of M. van der Meulen [8] proposes several approximation algorithms to find optimal radial configurations. An option that can be tried is adjusting these algorithms to minimize the objective function of this optimization problem.

Chapter 7

Conclusion

This chapter summarizes the important conclusions of this thesis and answers the research question. Afterwards, suggestions are made for further research.

7.1 Answering the research question

Recall the research question:

How do you minimize the SAIDI of a given MV network by placing circuit breakers and relocating net openings, given constraints on the structure, the number of circuit breakers in series and the cost and time needed for the implementation?

Alternatively, minimization of the SAIDI together with the cost was investigated.

This question has been solved partly in this project, the remainder is left to future research. The answer given by this thesis consists of several parts that correspond with the chapters:

1. Model the practical problem as a mathematical optimization problem.
2. Calculate the result of the objective function.
3. Find the optimal locations for additional breaker edges.
4. Find the optimal locations for openings.
5. Find the optimal locations for additional breaker edges and openings.

1. Mathematical model

The network is modeled as a graph. The most important conclusion in this part is that there are a lot of advantages to modeling the distribution lines of the network as vertices of the graph. Graphs of the configurations, which include the breaker edges and openings, are then subgraphs of the original graph.

Another observation is that the radiality conditions on the configuration graph give the possibility to create a rooted forest, by defining an orientation on the graph. This orientation makes it easier to find paths to or away from a root.

2. Objective function

The definition of a segment was given in this thesis. It is very inefficient to determine the impact of a single vertex, but it was shown that the impact of vertices in the same segment is equal. It is clear that every algorithm should calculate this impact only once per segment, instead of for every vertex.

This fact is used in two types of algorithms. The first is to calculate the result of the objective function directly for every segment, this is the most efficient when only executing it on one graph. The other is to initiate attributes on the edges of the graph which are the accumulated sums of the weights and values of the vertices from leaves to root. The initiation is not very efficient, but only needs to be executed once. The attributes prove to be very useful for the calculation of the objective function, but also for finding the optimal additional breaker edges.

3. Additional breaker edges

The attributes can be used to produce a list with the benefit of every edge very quickly, so this solves the case where we want to find the optimal location for one additional breaker edge.

It is also used for several types of algorithms that solve the more general problem where at most Z edges can be added. The branch and bound algorithm has a complexity that is exponential in Z. In practical cases it can still be doable in the time available to execute the algorithm. The advantage is that the branch and bound algorithm gives a global optimal solution. The greedy algorithm is more efficient, because it has a complexity linear in Z. However, it finds a local optimum, which is not necessarily the global optimum. The choice between these two algorithms is therefore balancing between time and accuracy.

An optimal feasible solution has to satisfy the breaker constraint. An algorithm is defined that checks if a given edge lies on a path with three breaker edges. This algorithm is executed every time an addition will be made to the set of breaker edges to check whether or not it violates the breaker constraint. This ensures that the resulting solution is feasible.

The branch and bound and greedy algorithm can both be adjusted to minimize the alternative objective functions. With the original objective function, there is always an optimal solution with Z additional breaker edges. When considering a alternative objective function the iteration of adding a breaker edge can end earlier. This means that the number of considered possibilities can only decrease, because more possible solutions are clearly not optimal. These algorithms are therefore useful in this case as well.

4. Relocating openings

When considering relocating openings, the radial conditions are very important. Of all subsets of the edges of the graph, only a relatively small number will create a radial configuration graph. It is shown that the set of roots can be merged into one vertex. All possible radial configurations are then exactly all possible spanning trees. The notion of a cycle basis can be used to create these spanning trees. The approaches used in this thesis do not solve this part of the problem, because of the great number of possibilities. The suggestion is made to try approximation algorithms like the evolutionary algorithms.

5. Combination of breaker edges and openings

The placement of breaker edges and openings influence each other. The optimal combination of locations has not been solved in this project. In the next section some suggestion will be made for further research, also on this particular problem.

7.2 Future research

The problem that is the subject of this project is so big that we were forced to leave certain questions unanswered. The remaining part of the research question is treated first, then other question are discussed.

7.2.1 Combination

An optimal solution of the optimization problem defined in this thesis consists of additional breaker edges and a new set of openings. There are two ways to obtain such a solution. One way is to do the assignment to the set of breaker edges or the set of openings simultaneously. However, we saw that for the set of openings, the best approach is to find the radial configurations first. This suggests that it might be better to start with radial configurations and then determining the locations for breaker edges on those configurations. It is clear that the solution set is so large that no matter which approach is taken, the global optimum will most likely not be found in a reasonable amount of time.

A heuristic that might be acceptable is to start with the set of openings, by using a search algorithm to find a the set that divides either the values or the weights of the vertices almost equally among the branches. Then with that set of openings, the optimal locations for additional breaker edges is calculated.

7.2.2 Other suggestions

Several other questions came up during this project, they are discussed in this subsection.

- The configuration of the network has to satisfy another constraint, namely the load flow constraint, which deals with the physics aspect of the electricity network. How can we solve this problem with this additional constraint?
- In this project, the outage duration for one failure is taken as a constant. However, this duration can be estimated more precisely, which means that it can no longer be omitted from the optimization problem. This will likely increase the difficulty of the optimization problem, but it will also be more precise in the expected outage time. Therefore it is an interesting adjustment to this problem.
- This is also the case with the time needed and cost of a reconfiguration. In this thesis, this is a constant for the circuit breakers and a constant for the net openings, but this does not have to be the case. When a location is for instance harder to reach, the time needed to make an adjustment there can be higher.
- What other protective devices are available for this objective?
  This thesis focuses on the net openings and circuit breakers, but these are not the only options available. It is for instance possible to build substations that can remotely open and close the connections with distribution lines. A similar research question can be stated using other protective measures like that one.

Bibliography

[1] H. Wolse, T. Bogaert en L. Derksen, *Betrouwbaarheid van elektriciteitsnetten in Nederland, resultaten 2016*, Netbeheer Nederland, Movares Nederland B.V., 2017.
`https://www.netbeheernederland.nl/_upload/Files/Betrouwbaarheid_van_elektriciteitsnetten_2016_76.pdf`

[2] R Core Team (2018). *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria.
`https://www.R-project.org/`

[3] G. Csardi, T. Nepusz, *The igraph software package for complex network research*, InterJournal, Complex Systems 1695. 2006.
`http://igraph.org`

[4] A. Middag, *Elektriciteitsvoorziening*, Liander Netmanagement, 2013.

[5] J.A. Bondy, U.S.R. Murty, *Graph theory*, Springer, 2008.

[6] J. Nieland, *Outage in the network, tackling the hitch*, Alliander, University of Twente, 2016.

[7] H. Fritschy, *Checking the $m-1$ principle on large electricity distribution networks*, master thesis, Radboud University Nijmegen, Alliander N.V., 2018.

[8] M. van der Meulen, *Reconfiguring Electricity Distribution Networks to Minimize Power Loss*, master thesis, Radboud University Nijmegen, Liander N.V., 2015.