INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Electrical Grid Maintenance and Scheduling Optimization

Eduardo Camões Fernandes

Dissertation for the Degree of Master in
Information Systems and Computer Engineering

Jury

President:	-
Supervisor:	Prof. Doctor Alexandre Paulo Lourenço Francisco
Co-supervisor:	Prof. Doctor Francisco de Moura e Castro Ascensão de Azevedo
Members:	-

October 2012

Resumo

A topologia de uma rede eléctrica é normalmente representada por um grafo parcialmente dirigido. Atribuindo valores de produção e consumo a cada nó, bem como limites de capacidade e probabilides de falha aos arcos, esta representação pode ser vista como uma rede de fluxos. Por sua vez, apesar de muitos dos problemas relacionados com fluxos terem algoritmos para os resolver, também podem ser adaptados e resolvidos através de problemas de optimização - exemplo do problema do fluxo máximo ou do custo mínimo.

Alguns dos desafios deste trabalho passam por trabalhar em diferentes abordagens ao problema, misturando algoritmos de problemas de fluxos em redes com optimização linear/não-linear. Este documento apresenta três abordagens diferentes, cujo propósito é o de obter resultados viáveis e melhorar os modelos existentes com a inclusão de novos elementos (como a probabilidade de falha dos nós).

O principal objectivo passa por encontrar a probabilidade de sucesso óptima de uma rede. Outro objectivo é responder a situações em que um ou mais arcos da rede são removidos, pretendendo-se uma redistribuição do fluxo. A comparação das vantagens e inconvenientes das abordagens e a análise dos resultados obtidos através de conjuntos reais de dados serão também focados na parte final do trabalho.

Abstract

The topology of an electrical power grid is usually represented by a partially directed graph. Assigning production and consumption necessities to each node as well as attributing capacity bounds and probabilities of failure to each edge, this framework representation turns into a flow network. On the other hand, although many of the network-flow related problems have algorithms to solve them, they can also be adapted to and solved as optimization problems - maximum-flow or minimum-cost, for example.

Some of the challenges of this work pass by working on different approaches to the problem and mixing network-flow related algorithms with linear/non-linear constrained optimization. This document presents three different approaches, whose purpose is to obtain viable results and improve the models by including new elements such as the probability of failure of the nodes.

The main goal focuses on finding a grid's optimal probability of success. Other goal is to respond to a situation where one or more edges are removed by redistributing the flow. The comparison of the advantages and inconveniences of the approaches presented and also the analysis of obtained results from real data sets will also be focused in the final part of the work.

Palavras Chave
Keywords

Palavras Chave

Grafo

Rede de fluxos

Optimização com restrições

Probabilidade de falha/sucesso

Função `fmincon`

Remoção de arcos

Keywords

Graph

Flow network

Constrained Optimization

Probability of failure/success

`fmincon` procedure

Edge removal

Acknowledgements

First of all, I would like to thank my thesis supervisor, Professor Alexandre Francisco, for the opportunity to work with him, for his wise presence during one year of hard work. For his guidance from the first to the last day; for his continuous availability that led to many fruitful and enlightening discussions. For making me think that somewhere in the future I may return to an area of research.

I would also like to say a word to my co-supervisor Professor Francisco Azevedo and to Engineer João Gomes-Mota from Albatroz Engineering for their availability to be ready to discuss the several approaches and for providing me with the data to work with.

The work of this thesis may resume to one year. But without the previous ones, I would not be where I am now. For that, I must thank to each and every one of my dearest friends.

I could not forget my family. I am deeply thankful for their support, for defining who I am. Their guidance and advice was/is indispensable: for the thesis and for everything else in life.

I must also address a special thanks to Paulo Gonçalves who was (and is) my great companion and friend since the beginning of this long five-year-old road that was the University. It is not possible to forget the endless nights of work that could not be surmounted without the humour and amusement that comes with him.

And finally, I must leave my kindest and dearest words to my Joana. For her enormous patience during this last year, mainly these last couple of months. Her support was enormous, from the first minute of every day to the last second of each night. Many times your words revealed to be a wise light and I will always be thankful for that. For that and for all the caring. For all the love.

Lisboa, October 2012

Eduardo Fernandes

To my grandmother Célia and my grandfather Bernardo

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

REN Rede Energética Nacional

RNT Rede Nacional de Transporte

POF Probability Of Failure

PLMI Power Line Maintenance Inspection

OHL Over-head Line

GIS Geographic Information System

BFS Breadth-first Search

SQP Sequential Quadratic Programming

QP Quadratic Program

POS Probability Of Success

MDN Maximal Distant Node

Chapter 1

Introduction

1.1 Motivation

Electrical power grids are constantly subject to problems related with the deployment of new electrical lines and with the maintenance of existing ones. As a way to meet the growing demand of electrical energy and the need of a higher quality of service, operators must continuously find new and more efficient techniques to solve the problems that can recurrently appear on the power grids.

The transportation of electrical energy in Portugal is made by Redes Energéticas Nacionais (REN). It is assured by a group of available production and consumption structures and electrical circuits connecting them. This group of elements constitutes what is called as Rede Nacional de Transporte (RNT) and can be topologically represented as a directed graph. This general graph is constituted by a set of edges that represent the electrical circuits connecting several nodes representing power plants, substations, sectioning posts or points of energy consumption.

The common problem faced by grid operators is that RNT electrical lines are continuously subject to failure, even when they are not transferring power from node to node. There are many reasons for line failures and they can be divided into two great groups: natural events, such as its age, vegetation proximity, forest fires, stork excrement or even lightning storms [14]; and artificial events, such as buildings proximity.

Each event can be seen as a point of interest (POI), which is defined by a set of several variables, and will be classified according to its criticality [13]. The difference seen in the criticality is directly associated to the "weight" of the effects each event has on the lines. Consequently, each electrical line has a probability of failure (POF) subject to the "weight" of the different factors.

Along with the POF there are other important features that define the electrical lines as the edges in the graph representation: capacity bounds and the cost of line's maintenance. To reduce the amount of failures and, consequently, the POF of a line in an

instant, the grid maintainers must perform a set of periodical inspections and maintenance actions. Also, lines have different length. Therefore, to a certain POF and length, there is always an associated cost of line maintenance.

Just like the lines, power stations (and each one of the other structural elements represented by the graph nodes) will have a set of defining features: initial conditions with respect to electrical energy production and consumption limits. With this kind of information the graph representation of the grid can be adapted and turned into a flow network.

Since most of the network-flow related problems can be solved in an optimization fashion, it will be built a bridge between the common rules of network-flow problem and constrained optimization. When optimizing and trying to find the optimal distribution of electrical energy throughout the grid lines (in an instant or through a period of time), the lines POF will play the most important role.

1.2 Goals

In short, the goals of this thesis are:

- A stable algorithm that computes the network minimum global POF, given a set of initial conditions of production and consumption, a topology and a set of each edge's POF. The result is a schematic of the distribution of the energy flow through the network edges and the value of the POF for each sink node's energy supply;
  - Incremental evolution of previous algorithm for a situation where one or more edges are removed from the network. That is, it should be possible to redistribute the flow through another set of edges and satisfy the restrictions of goal 1. Take into account different costs of energy production (by clients - source nodes), assumed to be absent in initial approaches;
- An algorithm that determines the edges from the network that should be more effective to maintain in terms of POF, given a fixed cost per kilometre and maintenance budget;
- The dual of the previous problem which consists in determining the minimum maintenance budget and the location of the actions to perform on the network, given an upper bound of the network's POF;
- According to the expansion of the POF's variance through time, create an algorithm for an optimal scheduling of the network's inspections that minimizes its expected global POF, admitting that its inspection is immediately followed by the maintenance.

1.3 Approach

The work of this thesis is based on the ideas of flows in network. Consequently, it started by understanding the basic definitions and rules that apply to network-flow problems. The first approach was based in the algorithmic method of Ford and Fulkerson [7] and its improvement by Edmonds and Karp [6], where it was considered that the POF of a line would be the element *cost* in the problem.

The subsequent two approaches have their foundations in the theory of constrained optimization. Initially the idea was to formulate the problem as a linear program somehow similar to the ones that solve the maximum-flow or minimum-cost. However, that turned out to be impossible and both approaches use the MATLAB®[1] procedure `fmincon` for non-linear constrained optimization problems, since the constraints that appear in the formulation are non-linear, as one will be able to understand in chapter 4.

1.4 Document Outline

The document is organized as follows:

- *Related Work:* chapter 2 presents the work done by Azevedo and Gomes-Mota on processing a framework for the maintenance of electrical grids, where the conclusions from this work will be integrated. It also presents theoretical definitions and rules to respect when dealing with network-flow problems. The maximum-flow and minimum-cost problems are introduced and explained in algorithmic and optimization fashion. The section ends with theoretical notions of non-linear constrained optimization, explaining the basis of the solver used in this problem;

- *The Problem:* chapter 3 addresses the problem and presents the different elements that constitute the electrical grid, by showing examples of distinct graphs;

- *Approach:* chapter 4 develops what was summarized in the previous section by explaining the different approaches to the problem, how they differ from each other and how the data was extracted and transformed to the graph format;

- *Evaluation:* chapter 5 provides the evaluation of results when applying the `fmincon` procedure to the optimization formulation for a certain instant of time in the grid. It also provides a comprehensive analysis of how the distribution of energy can change through a period of time, taking into account the changes in the function of probability associated to the several lines;

[1] http://www.mathworks.com/

- *Conclusions and Future Work:* chapter 6 summarizes the work reported in this thesis, the results achieved and its contributions. It also presents a set of open questions and future challenges related to this work that can still be answered.

Chapter 2

Related Work

In this chapter it is described the work that is being done in areas that affect directly this problem and its scope. It starts by explaining in section 2.1 the work done in Albatroz Engineering[1] to create a framework and a set of tools to perform regular inspections, extract and analyse data to subsequent maintenance scheduling. It is also presented the theoretical ideas of network-flow problems in section 2.2, the application of optimization ideas to these problems in section 2.3 and the theory of non-linear constrained optimization in section 2.4.

2.1 Albatroz Engineering Maintenance Cycle

In nowadays work with electric power grids the need to know the status of every line is the motivation that leads to regular inspections and resulting maintenance. After inspections reports are made from the data obtained from ground or airborne inspections systems like Power Line Maintenance Inspection (PLMI) [12] they are analysed by the maintenance systems and, in the end, fed into asset management systems. This summarized idea defines the work behind the twelve-task cycle of processes that Azevedo and Gomes-Mota [2, 3] came up with and where the scope of this work will enter (see Figure 2.1).

The framework proposed in [2, 3] to optimize the cycle of processes that keeps the electrical power grids working features an architecture consisting of a server with connections to Geographical Information Systems (GIS) and management systems with access to relational databases. These databases were built taking into account the topology of the network, the information associated with edges and nodes, the grid assets and the inspections made (identification, processed information from reports and historical data) [2, 3].

[1] http://www.albatroz-eng.com/

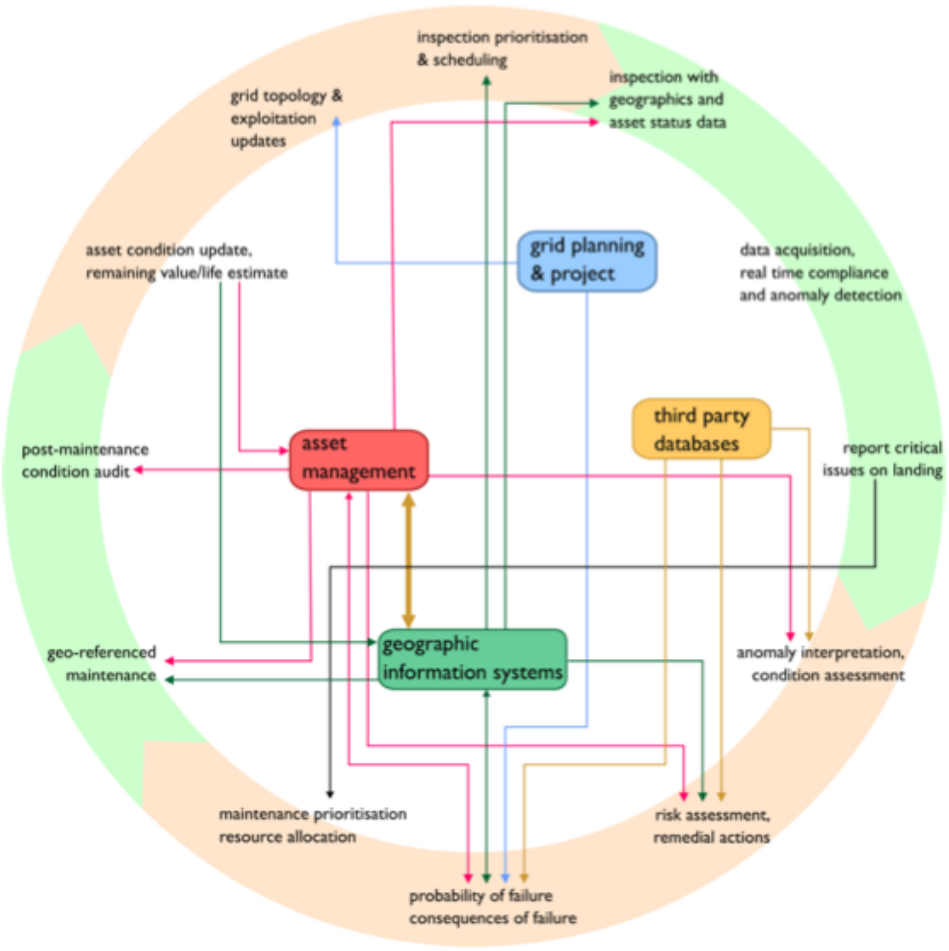

Figure 2.1: Albatroz Engineering Maintenance Cycle.

Every issue related to an element of the electrical grid must be addressed. So, after the deployment of the architecture, a set of tools was required to address those issues. These tools take into account an important common set of features in order to keep them consistent [2]:

- **Modularity:** follow the modular structure of the database so that every function or set of functions can be changed without modifying related data;
- **Quantitative, probabilistic outputs:** express all of the outputs in numerical forms, using stochastic variables when necessary and appropriate;
- **Independence:** functions and procedures of each tool kept autonomous from the data;
- **Traceability and benchmarking:** possibility of tracing back every result and operation to analyse them in the future.

The model presented in Figure 2.1, where green background represents the group of field tasks and sand background represents office tasks, is organized as a clock dial.

Entering it at one o'clock, there is the over-head line (OHL) inspection. One can see that it is directly pointed from the block of asset management, which is used to highlight previous issues and features of each element that are relevant for inspection, and from the block of GIS responsible for indicating the lines to inspect. GIS are also used for planning the optimal routes of inspection, since they have information about the location of every element in the grid.

The second set of tasks, at two o'clock, regards data acquisition/recording and anomaly real time detection. It will help the inspectors optimizing the procedures according to the condition of the line. The "real time" notion allows issues to be noticed as fast as possible and consequently reported immediately after inspection, which is the basis of the three o'clock set of tasks.

After the reports, there is a sequence of office tasks, where experts perform detailed analysis of the data (anomaly interpretation) - four o'clock - and make estimation of the condition of the OHL. The next step, which is still unimplemented in many utilities, involves the assessment of the risk associated to each issue and it tries to quantify the several risk inputs - vegetation, stork, pollution, fires, lightnings, buildings proximity - into a mathematical function. In the six o'clock tasks, the risk is computed into POF and the consequences of a certain failure are estimated locally in an OHL and globally in the entire grid by taking all potential risks into account. This set of tasks is the most recent and it aggregates data from every block as one can see in the model.

The scope of this work enters in the seven o'clock step which is done for maintenance prioritisation and resource allocation based on criticality reports, costs of maintenance and distances between neighbouring issues computed from GIS. The eighth step enters a new phase of field tasks and it is precisely destined for the field maintenance whereas the nine o'clock step is where the post-maintenance audits are made. However the audits can be performed simultaneously with the maintenance, depending on local practices and on the nature of the issues to be assessed.

The last phase of office tasks consists on updating on the information systems the grid condition after maintenance. This task is considered homologue to condition assessment at the four o'clock step. Azevedo and Gomes-Mota refer that half the cycle is dedicated to optimization remedy (maintenance) and the other half is dedicated to diagnostic optimization (inspection). At eleven o'clock the grid topology is updated (if new lines enter the grid service or others are cut out) and the cycle ends with the preparation for a new iteration with definition of an optimal route and scheduling of maintenance.

This cycle is explained in more detail by Azevedo and Gomes-Mota in [2, 3].

2.1.1 Inspection and POI

Every function of inspection returns a list of POIs [13], which is a descriptor of anything that may be worth noticing, because it can go beyond any faulty line or station and depict, for example, a building in the proximity of some power line that prevents the airborne inspection in the area.

Each POI is defined by localization, time reference, set of descriptors, qualitative, semi-quantitative and numerical variables and links to media objects. It is possible to compare POIs regarding the same object but created with different timestamps since they are all stored in the geo-referenced databases. From this comparison new POIs are generated in order to promote a new maintenance report and, consequently, a new set of response measures.

In order to respect every possible occurrence and because POIs have different incomparable descriptors, numerical fields and qualitative classification, it was created a way of classification to make different issues comparable in terms of criticality. The basis is the scale of classes presented in Figure 2.2.

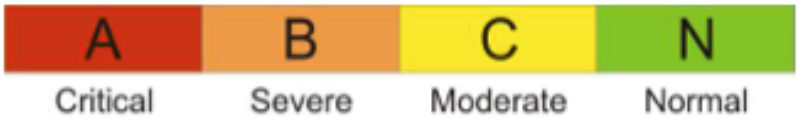

Figure 2.2: Classification for points of interest.

The classification of POIs (attribution to a class) varies according to the purpose they serve. One way is to classify POI with respect to the management practices (vegetation, for example) they will afford, so that it should be possible to associate a level of service or contract to a certain class. Each POI's set of descriptors is very important since they can include variables that serve as input to automated functions that calculate the severity of that POI.

After the classification and according to the severity of the event, each POI is associated to a POF or to the quantitative measure of "risk index" that relates several POIs. The computation of these values will be determinant for the ascertainment of where and when electrical lines should be maintained. This, in turn, leads to the length percentage of each line that must be verified in order to keep them and the grid under a certain risk of failure.

2.2 Network-flow Algorithms

Network-flow related problems are widely studied in operations research, computer science and they are applied in many real world situations. Its study began with early

work in linear programming and it was prompted out of that scope by Ford and Fulkerson [7].

A flow network is an interpretation of a directed graph and it is used to answer questions about material flows like commodity transportation, packet dispatch in wireless communication or, in this case, electrical energy flow. These materials course through a system from one or more *sources*, where the material is produced, to one or more *sinks*, where it is consumed. The system itself is represented by the graph and, without loss of generality, each circuit connecting the physical elements - *nodes* - of the system is represented by each *edge* of the graph.

There are many problems related with this subject and two of the most studied are the *maximum-flow problem* and the *minimum-cost flow problem*. In the first, the goal is to compute the greatest rate at which is possible to send material from the source to the sink of the network without violating capacity constraints (usually associated to the edges); in the latter, the goal is to compute the cheapest possible way of sending a certain amount of flow through the network, where there is a *cost* associated to the use of each edge.

Before addressing each of the problems separately, some definitions used generally in flow networks are presented. From now on, in every definition presented it is assumed that $G = (V, E)$ is the given network, where V is the finite set of vertices (nodes) and E is the finite set of edges. G has a distinguished *source* vertex $s \in V$, a distinguished *sink* vertex $t \in V$ and non-negative capacities for each edge.

Definition 1 (Flow)**.** *A* flow *in a network $G = (V, E)$ is a real-valued function on vertex pairs $f : V \times V \to \mathbb{R}$ that satisfies the properties of* capacity constraint *and* flow conservation*. Its value, $|f|$, is given by:*

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) \tag{2.2.1}$$

The flow value as it is defined above can be intuitively seen as the amount of commodity that leaves the *source* minus the commodity that enters the *source* s. It can also be defined as the amount of flow that enters the *sink* t.

Definition 2 (Capacity Constraint)**.** *In a network $G = (V, E)$, for each pair of nodes $u, v \in V$ it is required $0 \leq f(u, v) \leq c(u, v)$, where $c(u, v)$ represents the edge's capacity.*

Definition 3 (Flow conservation)**.** *In a network $G = (V, E)$, for every node $u \in V \setminus \{s, t\}$, where s is the source and t is the sink, it is required that:*

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v) \tag{2.2.2}$$

Whenever edge $(u, v) \notin E$ there is no flow from u to v, thus $f(u, v) = 0$. In particular, the two members that form the equality are equal to 0 for every $v \in V \setminus \{s, t\}$.

2.2.1 Maximum-flow Problem

As it was aforementioned, the maximum-flow problem is one of the most studied in this area. It is also one of the simplest problems concerning the flows in networks, being a special case of the *minimum-cost problem*, and can be solved by many efficient algorithms with different theoretical approaches. Its purpose is to compute the *maximum flow* - a flow of maximum value - or alternatively a *minimum-cut* - a *cut* of minimum capacity - in a network.

Ford-Fulkerson Method

As it is referred in [4], this approach is considered a "method" because it can be implemented in several different ways and, consequently, with differing running times. It is important, presenting this method, to introduce some important definitions that are referred in [1, 4]: *residual network, residual capacity, augmenting path* and *cut*.

Definition 4 (Residual Network)**.** *Given a network $G = (V, E)$ and a flow f, the* residual network *G_f is defined with respect to f: each edge (i, j) in G is replaced by two edges, (i, j) and (j, i) and the* residual network *consists of this replacement of the edges, including only those with positive* residual capacity *(see* Definition 5*).*

Definition 5 (Residual Capacity)**.** *Given a network $G = (V, E)$, a flow f and considering two nodes $u, v \in V$, the* residual capacity *$c_f(u, v)$ is defined as follows:*

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{if } (u,\ v) \in E \\ f(v, u) & \text{if } (v,\ u) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.2.3}$$

Definition 6 (Augmenting Path)**.** *Given a network $G = (V, E)$ and a flow f an* augmenting path *$p = \langle v_1, \ldots, v_k \rangle$ is a simple path from source $s = v_1$ to sink $t = v_k$ in the* residual network *G_f. The maximum amount by which the flow can be increased on each edge in p (*residual capacity *of p) is given by:*

$$c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is on } p\} \tag{2.2.4}$$

The Ford-Fulkerson method repeatedly increases the flow through augmenting paths until it has found a maximum flow. When the algorithm terminates it must have reached the maximum possible flow. Using the *maximum-flow minimum-cut theorem* [7] this can be proved and in order to understand it the notion of *cut* must be addressed [4]:

Definition 7 (Cut). *A* cut (S, T) *of a network* $G = (V, E)$ *is a disjoint partition of* V *into* S *and* $T = V - S$ *such that the source* $s \in S$ *and the sink* $t \in T$*. The* capacity *of the* cut *is given by:*

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v) \tag{2.2.5}$$

Theorem 1 (Maximum-flow Minimum-cut). *If* f *is a flow in a network* $G = (V, E)$ *with source* s *and sink* t*, then the following conditions are equivalent:*

1. f *is a maximum flow in* G*;*
2. *The* residual network G_f *contains no* augmenting paths*;*
3. $|f| = c(S, T)$ *for some cut* (S, T) *of* G*.*

In other words, what the theorem states is that if the flow is the network maximum flow then its *residual network* does not contain any further *augmenting path* and the *capacity* of the *cut* (S, T) is the value of the flow.

The basic algorithm suggested by Ford and Fulkerson is less efficient than those developed afterwards, since it has no restrictions in the way it finds an augmenting path (it searches for one until there is none from source s to sink t). Regarding this fact, it has a running time[2] of $O(E\ |f_{max}|)$, where $|f_{max}|$ denotes the network's maximum-flow value. This bound can be very "harmful" in terms of spent time: one usual example is presented in Figure 2.3 when a network has a maximum-flow $f_{max} = 2 \times 10^6$ and each augmenting path found increases the flow by 1. One can conclude that it will take precisely 2×10^6 iterations of the algorithm to achieve a maximum-flow.

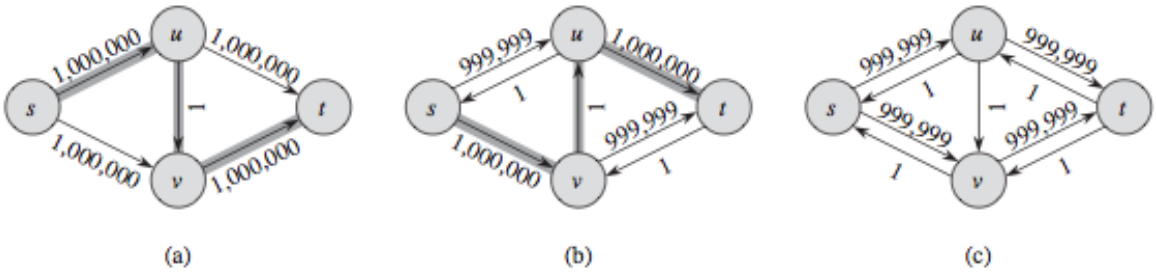

Figure 2.3: Flow network example.

Edmonds-Karp Algorithm

A simple modification to the basic algorithm of Ford and Fulkerson was proposed by Edmonds and Karp [6] to improve the algorithm's complexity bound. It used the same

[2]The complexity of network-flow algorithms is based on the size of V and E. It is assumed in the asymptotic notation that $V \equiv |V|$ and $E \equiv |E|$.

basic ideas but the difference was on a restriction for the search of an augmenting path: instead of choosing one arbitrarily, it would apply a breadth-first search (BFS) to find it. With the BFS the augmenting path is chosen as the shortest augmenting path (in terms of distance) between the source s and the sink t in the residual network, where each edge has unitary distance.

The Edmonds-Karp algorithm has an upper bound for the running time of $O(VE^2)$ and using the example of Figure 2.3 from [4] one can see that it only takes two iterations for choosing the augmenting paths and to achieve the maximum flow: paths $p_1 = \langle s, u, t \rangle$ with $|f| = 10^6$ and $p_2 = \langle s, v, t \rangle$ with also $|f| = 10^6$, producing the maximum flow $|f_{max}| = 2 \times 10^6$.

Algorithms Evolution

After the work of Ford and Fulkerson [7], Edmonds and Karp [6] found new improved bounds for the maximum-flow problem, as it was above-mentioned. Independently, Dinic [5] proposed a method to find all shortest augmenting paths in one phase with a better bound than Edmonds-Karp algorithm - $O(V^2E)$. Many different approaches were made afterwards based on the Dinic formulation of the problem. Goldberg *et al.* present a set of algorithms for the maximum-flow problem in [8], giving their attention to the *general push-relabel method* which yields a complexity bound of $O(V^2E)$. This algorithm is asymptotically better than the $O(VE^2)$ algorithm of Edmonds-Karp since $|E| = O(V^2)$. The *push-relabel* complexity depends on the used implementation and many improvements were made throughout the decades of 1970 and 1980: the vertex selection rule using FIFO (first-in, first-out) in lists which forms the basis of the *relabel-to-front algorithm* has an $O(V^3)$ complexity, the highest vertex selection rule allows a bound of $O(V^2\sqrt{E})$ and the implementation of Goldberg and Tarjan [9] using dynamic tree data structure, based on the concept of *preflow*, runs in $O(VE\ log(V^2/E))$, which is the best complexity known so far to solve the maximum-flow problem.

2.2.2 Minimum-cost Problem

It was aforesaid that one of the goals of this work was directly related to the computation of the cheapest possible way of sending a certain amount of flow through the network. In order to do that and with the topology of this network, each edge would be associated to a certain cost of traversal. However, this cannot be seen as an usual linear cost: each edge will have a POS associated and it will be this element that will be considered the *cost* of traversing the edge. The approach with this consideration will be detailed in chapter 4.

The most common approach for this problem uses linear programming (see subsection 2.3.2) but there are some other ways to address the problem that comprehend different

techniques like *capacities-scaling*, introduced by Edmonds and Karp [6], *cost-scaling* proposed by Röck [17] and the *generalized cost-scaling* proposed by Goldberg and Tarjan [10, 11]. Explanations on the details of these approaches can also be found in [1, 8].

2.3 Optimization and Flows in Networks

When the term "optimization" is applied the idea of choosing the best solution, configuration or set of parameters for a problem comes to mind. The complexity of a problem like this varies depending on the function to optimize and on those that restrict the set of possible solutions - the constraints. According to [16], an optimization problem can be classified as *general non-linear*, *convex* or *linear*, with decreasing complexity from first to latter. The general non-linear programming problem can be represented as follows[3]:

$$\begin{aligned} \underset{\mathbf{x}}{\textbf{minimize}} \quad & f(x) \\ \textbf{subject to} \quad & g_i(x) \geq 0,\ i = 1, \ldots, m \\ & h_j(x) = 0,\ j = 1, \ldots, n \end{aligned} \tag{2.3.1}$$

where f, g_i and h_j are general functions of $x \in \mathbb{R}^n$.

If all functions are restricted to be linear then a linear programming problem is to be faced.

2.3.1 Maximum-flow Problem

For the formalization of the maximum-flow problem as a linear program it is necessary to introduce an important concept of a commonly used idea in the conception of network flow problems in optimization form, which is the *node-edge incidence matrix*:

Definition 8 (Node-edge Incidence Matrix)**.** *Given a network $G = (V, E)$, the* node-edge incidence matrix *A is a $|V| \times |E|$ matrix, where each row represents a node and each column represents an edge; for the column correspondent to each edge (u, v), row u has value 1 (the tail of the edge), row v has value -1 (the head of the edge) and all other entries have value 0.*

Also important for the definition of the *maximum-flow problem* as a linear program is the fact that the network has two nodes with different properties than the others - source s and sink t -, each edge (u, v) has a non-negative capacity $c(u, v)$ and the flow

[3] An optimization problem can be a minimization or a maximization. On the other hand, a maximization expression may be easily transformed to the minimization of its negative. Without loss of generality, it is used the *minimize* statement. The representation is intended to be general: any kind of inequalities can appear in the constraints of the problem ($<$, $>$ or $\leq$).

$f(u, v) : V \times V \to \mathbb{R}$ satisfies the capacities constraints and the flow conservation. In this sense, Papadimitriou and Steiglitz [16] propose the following formalization of the problem:

$$\begin{aligned} \textbf{maximize} \quad & v \\ \textbf{subject to} \quad & Af + dv \leq 0 \\ & f \leq c \\ & f \geq 0 \end{aligned} \tag{2.3.2}$$

where v is an $s - t$ flow, A is the node-edge incidence matrix of the network, f, $c \in \mathbb{R}^n$ are, respectively, the flow and capacity vectors and vector $d \in \mathbb{R}^n$ is defined by:

$$d_i = \begin{cases} -1 & i = s \\ 1 & i = t \\ 0 & \text{otherwise} \end{cases} \tag{2.3.3}$$

for all edges $(u, v) \in E$ and where $n = |E|$.

2.3.2 Minimum-cost Problem

The minimum-cost flow problem is a generalization of the maximum-flow and therefore their formalization as an optimization problem is quite similar. The difference comes with the fact that in the previous problem there isn't any *weight* or *cost* associated to the edges traversal and the minimum-cost problem asks for flow of fixed value that is cheapest among all such flows. Considering this, it is possible to define the problem of the minimum-cost flow as the following linear program:

$$\begin{aligned} \textbf{minimize} \quad & w^\top f \\ \textbf{subject to} \quad & Af = -v_0 d \\ & f \leq c \\ & f \geq 0 \end{aligned} \tag{2.3.4}$$

where $w \in \mathbb{R}^n$ is the cost (weight) vector and v_0 is the value of an initial flow for the program, which can be easily computed using the maximum-flow. It is important to refer that each one of the inequalities of the formalization are satisfied with equalities since, according to [16], a deficit in the flow balancing at any node implies a surplus at some other, respecting the idea of flow conservation.

2.4 Non-linear Constrained Optimization

Constrained optimization consists in seeking for a solution that minimizes a certain objective function, subject to constraints on the variables. If any of this functions is non-

linear, the optimization problem is non-linear. The following formulation is the general form for these type of problems:

$$\begin{aligned} \underset{\mathbf{x}}{\textbf{minimize}} \quad & f(x) \\ \textbf{subject to} \quad & c_i(x) = 0, \quad i \in \mathcal{E} \\ & c_i(x) \geq 0, \quad i \in \mathcal{I} \end{aligned} \tag{2.4.1}$$

where f and the functions c_i are all smooth, real-valued functions on a subset of $\mathbb{R}^n$ (possibly non-linear) and $\mathcal{E}$ and $\mathcal{I}$ are two finite set of the indices with respect to the equality or inequality constraints.

2.4.1 Karush-Kuhn-Tucker (KKT) Conditions

Methods for solving constrained optimization problems have been evolving and the most effective ones find their focus on the solution of the KKT conditions. They define the *first-order necessary conditions* as they are concerned with properties of the gradients (first-derivative vectors) of the objective and constraint functions.

Definition 9 (Gradient)**.** *The gradient of a scalar function $f(x_1, x_2, \ldots, x_n)$ is denoted $\nabla f(x)$ and is written component-wise as:*

$$\nabla f(x) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} \\ \vdots \\ \dfrac{\partial f}{\partial x_n} \end{bmatrix}. \tag{2.4.2}$$

Theorem 2 (KKT Conditions)**.** *If x^* is a local minimum in Equation 2.4.1, the functions f and c_i also in Equation 2.4.1 are continuously differentiable (C^1) and the LICQ[4] holds at x^*, then there is a Lagrange multiplier vector λ^*, with components $\lambda_i, i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions are satisfied at (x^*, λ^*):*

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \tag{2.4.3}$$

$$c_i(x^*) = 0, \quad \textit{for all} \quad i \in \mathcal{E}, \tag{2.4.4}$$

$$c_i(x^*) \geq 0, \quad \textit{for all} \quad i \in \mathcal{I}, \tag{2.4.5}$$

$$\lambda_i(x^*) \geq 0, \quad \textit{for all} \quad i \in \mathcal{I}, \tag{2.4.6}$$

$$\lambda_i c_i(x^*) = 0, \quad \textit{for all} \quad i \in \mathcal{E} \cup \mathcal{I} \tag{2.4.7}$$

[4]Linear independence constraint qualification (LICQ) holds if the set of active constraints $\{\nabla c_i(x), i \in \mathcal{A}(x)\}$ is linearly independent, where x is a given point and the *active set* $\mathcal{A}(x)$ is defined as $\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} : c_i(x) = 0\}$

The expression $\mathcal{L}(x^*, \lambda^*)$ is known as the *Lagrangian function* and is defined as

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x) \tag{2.4.8}$$

and one can compute its gradient (with respect to x) as

$$\nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i \nabla c_i(x). \tag{2.4.9}$$

This set of conditions relates the first derivatives of the objective function f to each of the constraint functions c_i at a possible solution x^*. When the KKT conditions are satisfied, a move along a certain vector w either increases the first-order approximation to the objective function - $w^T \nabla f(x^*) > 0$ - or keeps this value at the same place - $w^T \nabla f(x^*) = 0$. Whenever this second case happens the second-derivative vectors of f and c_i assume a "tie-breaking" role, since the KKT conditions are unable to decide whether a move along a certain vector w will increase or decrease the objective function.

It is also important to refer that whenever there is an absence of inequality constraints, the KKT conditions resume to the conditions of Equation 2.4.3 and Equation 2.4.4.

2.4.2 Sequential Quadratic Programming (SQP)

The MATLAB® procedure `fmincon` solves nonlinear constrained optimization problems. It also allows the existence of linear constraints as well as variable bounds, respecting the following notation:

$$\begin{aligned}
\underset{\mathbf{x} \in \mathbb{R}^n}{\textbf{minimize}} \quad & f(x) \\
\textbf{subject to} \quad & c(x) \leq 0 && \text{(function of nonlinear inequality constraints)} \\
& c_{eq}(x) = 0 && \text{(function of nonlinear equality constraints)} \\
& Ax \leq b && \text{(linear inequality constraints)} \\
& A_{eq}x = b_{eq} && \text{(linear inequality constraints)} \\
& lb \leq x \leq ub && \text{(variable bounds),}
\end{aligned} \tag{2.4.10}$$

where x, b, b_{eq}, lb, ub are vectors, A and A_{eq} are matrices and f, c and c_{eq} are functions that can be nonlinear.

There are different algorithms for solving nonlinear constrained optimization. The `fmincon` procedure enables the choice between four of them and the one used in this process is based in SQP.

SQP methods represent the state of the art in the nonlinear programming area. They are one of the most effective as they generate steps by solving quadratic subproblems and they show their strength precisely when the constraints present significant nonlinearities.

The general quadratic program (QP) can be stated as:

$$\begin{aligned} \underset{\mathbf{x}}{\textbf{minimize}} \quad & q(x) = \frac{1}{2}x^T G x + x^T c \\ \textbf{subject to} \quad & a_i^T x = b_i, \quad i \in \mathcal{E} \\ & a_i^T x \geq b_i, \quad i \in \mathcal{I}, \end{aligned} \tag{2.4.11}$$

where G is a symmetric matrix.

The basic local SQP method will be explained in this section, as it is presented in the 18$^{\text{th}}$ chapter of [15], since it is the base of the SQP algorithm in the `fmincon` procedure.

It starts by considering the following equality-constrained problem:

$$\begin{aligned} \underset{\mathbf{x}}{\textbf{minimize}} \quad & f(x) \\ \textbf{subject to} \quad & c(x) = 0, \end{aligned} \tag{2.4.12}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are smooth functions.

Newton's Method

The simplest SQP methods can be seen as an application of the Newton's method from [15] to the KKT optimality conditions.

Definition 10 (Jacobian matrix)**.** *Matrix of all first-order partial derivatives of a vector- or scaled-valued function with respect to another vector. If $F : \mathbb{R}^n \to \mathbb{R}^m$ is a set of functions composed by $F_1, \ldots, F_m$, then, the Jacobian matrix $\mathcal{J}$ of F is written as follows:*

$$\mathcal{J}_F = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \cdots & \frac{\partial F_m}{\partial x_n} \end{bmatrix}. \tag{2.4.13}$$

From Equation 2.4.8, it is possible to infer that the Lagrangian function for Equation 2.4.12 is $\mathcal{L}(x, \lambda) = f(x) - \lambda^T c(x)$, where λ^T is the Lagrangian multipliers vector. Let $\mathcal{J}(x)$ be the $n \times n$ Jacobian matrix of the constraints. Its transpose matrix will then be

$$\mathcal{J}(x)^T = \begin{bmatrix} \nabla c_1(x), \nabla c_2(x), \ldots, \nabla c_n(x) \end{bmatrix}, \tag{2.4.14}$$

with $c_i, i \in \{1, \ldots, m\}$ being the i^{th} component of vector c.

The KKT optimality conditions of Equation 2.4.12 form a system of $n + m$ equations with respect to x and λ. The corresponding $n \times m$ matrix $F(x, \lambda)$ is written taking into account the Jacobian matrix $\mathcal{J}(x)$,

$$F(x, \lambda) = \begin{bmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ c(x) \end{bmatrix} = \begin{bmatrix} \nabla f(x) - \mathcal{J}(x)^T \lambda \\ c(x) \end{bmatrix} = 0, \tag{2.4.15}$$

and it is satisfied by any solution (x^*, λ^*) of Equation 2.4.12 for which $\mathcal{J}(x^*)$ has full rank - $rank(\mathcal{J}(x^*)) = n$.

One of the suggested approaches to solve the nonlinear equations of Equation 2.4.15 is the Newton's method. To use it, one starts by computing the Jacobian matrix of Equation 2.4.15 with respect to x and λ,

$$\mathcal{J}_F(x, \lambda) = \begin{bmatrix} \dfrac{\partial F_1}{\partial x} & \dfrac{\partial F_1}{\partial \lambda} \\ \\ \dfrac{\partial F_2}{\partial x} & \dfrac{\partial F_2}{\partial \lambda} \end{bmatrix}, \tag{2.4.16}$$

where the components of matrix $F(x, \lambda)$ are defined as

$$\begin{aligned} F_1 &= \nabla f(x) - \mathcal{J}(x)^T \lambda \\ F_2 &= c(x). \end{aligned} \tag{2.4.17}$$

Consequently, $\mathcal{J}_F(x, \lambda)$ is written as:

$$\mathcal{J}_F(x, \lambda) = \begin{bmatrix} \nabla^2_{xx} \mathcal{L}(x, \lambda) & -\mathcal{J}(x)^T \\ \mathcal{J}(x) & 0 \end{bmatrix}. \tag{2.4.18}$$

The Newton step from iteration (x_k, λ_k) will be defined by:

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix}, \tag{2.4.19}$$

where k indicates the k^{th} iteration of the problem and p_k and p_λ solve the Newton-KKT system:

$$\underbrace{\begin{bmatrix} \nabla^2_{xx} \mathcal{L}_k & -\mathcal{J}_k^T \\ \mathcal{J}_k & 0 \end{bmatrix}}_{\text{KKT Matrix}} \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix} = \begin{bmatrix} -\nabla f_k + \mathcal{J}_k^T \lambda_k \\ -c_k \end{bmatrix}, \tag{2.4.20}$$

which is well-defined when the KKT matrix is nonsingular. This will only happen if at $(x, \lambda) = (x^*, \lambda^*)$ the following assumptions are respected:

Assumption 1. *The matrix $\mathcal{J}(x)$ has full row rank - it only happens when the LICQ holds;*

Assumption 2. *The matrix $\nabla^2_{xx} \mathcal{L}(x, \lambda)$ is positive definite on the tangent space of constraints:*

$$d^T \nabla^2_{xx} \mathcal{L}(x, \lambda) d > 0, \forall d \neq 0, \tag{2.4.21}$$

such that $\mathcal{J}(x)d = 0$, which holds when (x, λ) is close to the optimum (x^*, λ^*).

The Newton method is proved to be a very good point of view that eases the analysis. It also constitutes an excellent algorithm for solving equality-constrained problems like Equation 2.4.12, provided that the starting point x_0 is close enough to the optimum solution x^*. Whereas the Newton approach can be quite simple, the SQP framework presented by [15] and briefly explained in the next section is the drive to the practical algorithms used nowadays. It also enables the extent of the technique used in equality-constrained problems to the inequality-constrained case.

SQP Framework

As it was aforementioned, SQP methods are effective by generating (and solving) quadratic subproblems that model Equation 2.4.12 at a certain iteration k. Consequently the minimizer x_k regarding the k^{th} subproblem will define the new iteration $k+1$, x_{k+1} will define iteration x_{k+2} and so forth.

So, at the iteration k, with the pair of vectors (x_k, λ_k) named as iterate, the problem represented by Equation 2.4.12 can be modelled by the following quadratic program:

$$\begin{aligned} \underset{\mathbf{p}}{\textbf{minimize}} \quad & q_k = \frac{1}{2} p^T \nabla^2_{xx} \mathcal{L}_k p + \nabla f_k^T p + f_k \\ \textbf{subject to} \quad & \mathcal{J}_k p_k + c_k = 0 \end{aligned} \tag{2.4.22}$$

Assuming that Assumptions 1 and 2 hold, this problem will have a unique solution (p_k, l_k), satisfying the first KKT optimality condition $\nabla_x \mathcal{L}_k = 0$.

Definition 11 (Derivative Rule)**.** *Let $q(x) = ax^T Gx + x^T b + c$, where a is a scalar, x, b and c are vectors and G is a matrix. The rule for the gradient shows that*

$$\nabla q(x) = a \times (G + G^T)x.$$

Consequently, if G is symmetric, $G = G^T$ and

$$\nabla q(x) = 2a \times Gx.$$

Regarding the fact that the matrix $\nabla^2_{xx} \mathcal{L}_k$ is symmetric, by the Definition 11 the gradient of q_k is given by the following expression:

$$\nabla q_k = \nabla^2_{xx} \mathcal{L}_k p_k + \nabla f_k \tag{2.4.23}$$

Since the model of Equation 2.4.22 is subject to a constraint, (p_k, l_k) must satisfy the following system:

$$\begin{aligned} \nabla_x \mathcal{L}_k = \nabla^2_{xx} \mathcal{L}_k p_k + \nabla f_k - \mathcal{J}_k^T l_k = 0 \\ \mathcal{J}_k p_k + c_k = 0, \end{aligned} \tag{2.4.24}$$

The iterate (p_k, l_k) is related with the solution (p_k, p_λ) of Equation 2.4.20 and by eliminating the term $\mathcal{J}_k^T \lambda_k$ from both sides of the equation, the following Newton-KKT system is obtained:

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}_k & -\mathcal{J}_k^T \\ \mathcal{J}_k & 0 \end{bmatrix} \begin{bmatrix} p_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f_k \\ -c_k \end{bmatrix}, \tag{2.4.25}$$

where $\lambda_{k+1} = l_k$ and p_k solves Equation 2.4.20 and Equation 2.4.22. The new pair (x_{k+1}, λ_{k+1}) can be seen as the solution of Equation 2.4.22 or as the Newton step from Equation 2.4.19. This is the consequence of the pair generated by Newton's method in Equation 2.4.20 applied to the KKT optimality conditions of the initial problem in Equation 2.4.12.

For the algorithm to continue one or more conditions must be satisfied. The optimality convergence tests that form the stopping condition of the SQP basic method can be associated to the feasibility of x^* or to the change on the step size. The algorithm cycle ends if one of the *tolerance values*[5] is crossed. In the SQP method[6] of `fmincon` procedure the following stopping criteria tests are performed:

- *First-order optimality measure:* it is based on the KKT conditions, analogous to the fact that the gradient of a point must be zero at a minimum, also taking constraints into account:

$$\texttt{TolFun} = \|\nabla_x \mathcal{L}_k\| = \|\nabla f_k - \lambda_k^T \nabla c_k\| < \mathbf{10^{-6}}; \tag{2.4.26}$$

- *Change in the objective function*[7]*:* a lower bound on the change of the objective function value during a step (iteration) of the algorithm. The algorithm ends if:

$$\texttt{TolFun} = |f_k - f_{k+1}| < \mathbf{10^{-6}}; \tag{2.4.27}$$

- *Step size:* a lower bound on the step size of the algorithm; the algorithm ends if

$$\texttt{TolX} = \|x_k - x_{k+1}\| < \mathbf{10^{-6}}; \tag{2.4.28}$$

The following picture[8] is an example of the evolution of the SQP algorithm:

[5] Threshold values associated to each one of the criterion for stopping the algorithm.
[6] The **bold** values represent the default values of SQP in MATLAB® `fmincon` procedure.
[7] `TolFun` is the tolerance value used for both *first-order optimality* and *change in the objective function* criteria.
[8] This graphic was obtained from MATLAB® documentation to explain the thresholds `TolFun` and `TolCon`.

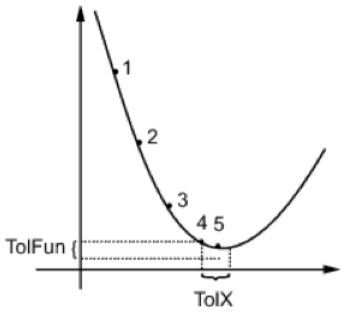

Figure 2.4: `TolFun` and `TolX` example.

From iteration 4 to 5 both `TolFun` and `TolX` thresholds would be crossed. At least one of the convergence tests would not be satisfied and therefore the algorithm would end.

It is important to refer that `fmincon` will only output a point that is a local minimum if the *first-order optimality measure* criterion is satisfied and if the *maximum constraint violation* is less than the value `TolCon`:

$$\texttt{TolCon} = |c_k| < \mathbf{10^{-6}}. \tag{2.4.29}$$

This last criterion is not included in the previous list since it is not a stopping criterion. That is, even if `TolCon` is lower than the tolerance value, the algorithm tries to continue until it halts because one of the previous explained conditions was satisfied.

The basic algorithm for the SQP method is shown in the following pseudo-code:

Algorithm 1 `Basic SQP for Equation 2.4.12`

Input: Non-linear constrained optimization problem of Equation 2.4.12.
Output: (x^*, λ^*) - optimal solution.

```
1  Choose initial iterate - (x_0, λ_0);
2  k ← 0;
3  while convergence test is not satisfied do
4     Model Equation 2.4.12 using Equation 2.4.22:
5        Compute f_k, ∇f_k, ∇²_xx L_k, c_k and J_k;
6     Solve Equation 2.4.22 and obtain (p_k, l_k);
7     x_{k+1} ← x_k + p_k;
8     λ_{k+1} ← l_k;
9     k ← k + 1;
10 end while
```

Chapter 3

The Problem

Following the ideas presented in the previous chapter, this one presents the problem itself in its topological form, describing the elements that take part in the electrical grid that serves as center of the work. The last section presents the problem statements given the description of the structural elements.

3.1 The Electrical Grid

As it was mentioned in chapter 1, there is a group of structural elements that constitute an electrical grid. Each one of this elements has a set of defining characteristics. For example, the nodes can produce or consume limited amounts of electrical energy that courses through the edges that connect the nodes.

There are five different groups of elements represented by the nodes that exist in the electrical grid of the problem:

- *Power plants*: nodes that are exclusively responsible for feeding the grid with energy, *i.e.*, they only produce energy - they can be seen as *source nodes*;
- *Substations:* nodes that can simultaneously produce and receive energy from other nodes; most of them are responsible for feeding the outer nodes of the grid (what is called the *distribution network*);
- *Sectioning/Cut posts:* intermediate nodes that only have the purpose of redirecting the energy to other nodes using the edges connected to them.
- *Derivations:* virtual nodes that are placed in the middle of the electrical lines; they redirect the flow coursing through a line to another line;
- *Sinks:* as the name itself says, these nodes are the ones that only consume energy.

Every node has a limit of production or consumption and that has to be taken into account, when trying to optimize the distribution of the electrical energy.

To connect pairs of nodes there is always, at least, one electrical line - an edge. For each edge there is an upper bound of the amount of energy that can course through it and most important of all, there is a POF associated to it. It is this element that will command the actions of the optimization of flow distribution, since its purpose is to find the distribution of flow that minimizes the grid's POF.

3.1.1 Inner and Outer Grid

One important aspect of this grid is that it can be divided into two great sub-grids: the grid of transmission and the grid of distribution of electrical energy. From now on, the first will be named as *inner grid* whereas the latter will be named as *outer grid*.

The outer grid is constituted my almost every sink. This makes all the sense, since the sinks are the only nodes exclusively responsible for consumption of energy. However, there are some sinks that make part of the inner grid. Every other structural element - power plants, substations, sectioning/cut posts and derivations - are exclusive part of the inner grid.

3.2 Topology

The grid can easily be topologically represented by a directed graph. As a matter of fact, it can be seen as a flow network, where the flow conservation or the capacity constraint rules play a very important role. From now on, let G be the graph that represents the electrical grid.

G is generally defined as $G = (V, E)$, where V is the set of nodes and E is the set of edges of the graph, $m = |V|$ and $n = |E|$. On the other hand each edge $(u, v) \in E$ has an associated POF. Consequently, it will embrace a complementary probability of success (POS), which will be used from now on, for an easier manipulation of the mathematical elements of the optimization.

Beside the function of probability, each edge has an amount of flow associated to it (the one used as argument of the function). Since it has to respect the rule of capacity constraint,

$$0 \leq f(u, v) \leq c(u, v), \tag{3.2.1}$$

where $f(u, v) : V \times V \to \mathbb{R}_0^+$ is the flow that courses through the edge (u, v) and $c(u, v) : V \times V \to \mathbb{R}_0^+$ is the capacity of the edge, *i.e.*, the upper bound of $f(u, v)$.

Each node $v \in V$ is also subject to some constraints, because many of the power stations represented in the graph nodes may have a limit of energy which can course through it. The following graph example has the purpose of showing the type of node restrictions that may appear.

Example 1 (Node capacities constraint)**.** *The schematic of Figure 3.1 is based on the example network from [2, 3, 13]. It is assumed that there is a production of 480 units of flow in the source node A.*

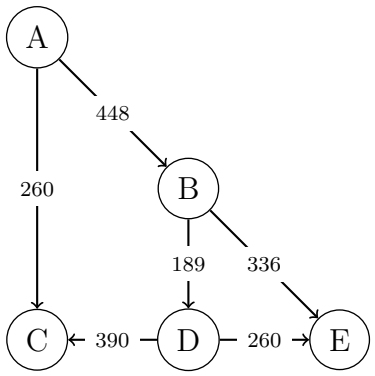

Figure 3.1: Example network G with multiple sinks - C and E.

The numbers over the edges (and from now on in all figures below) represent the edge capacity. It is possible to verify that, for example, node C can receive a flow of 260 units from source node A through edge (A, C)*. However, if there is a constraint in the node itself saying that the maximum flow that can go through it is 240 units, that must be taken into account. In this sense, the total sum of flow through edges* (A, C) *and* (D, C) *must not be over the limit of 240. Consequently, the initial hypothesis of coursing the flow of 260 units through edge* (A, C) *would not be feasible as* $260 > 240$*.*

Every theoretical definition related to network-flow problems that was explained in the previous chapter assumes a single-source, single-sink network. However, the real network has multiple sources and sinks and every detail on the maximum-flow problem is easily adapted to these circumstances. The previous graph is a perfect example of a multiple-sink (nodes C and E) network.

3.2.1 Power Plants

The power plants are definitely considered the exclusive *source nodes* of the graph as they consist on the nodes that only produce energy.

Let S be a subset of V, $S \subset V$. If $s \in V$ is a source node, then $s \in S$. Also, if s is connected to other nodes by any edge $(v_x, v_y) \in E$, then $v_x \equiv s$. This happens because these nodes do not receive energy and consequently the edges always depart from them towards others (substations, derivations or sinks). In other words, they are *always* the starting points of edges that connect them to other nodes.

Example 2 (Power Plant)**.** *This example illustrates the difference between a power plant and another nodes. The graph shows that A (in blue) is a source node feeding nodes B, C. On the other hand, by applying the rule of flow conservation, node B will transfer energy*

towards node D, whereas node C will redistribute the flow received from node A towards either node B or node D (or both).

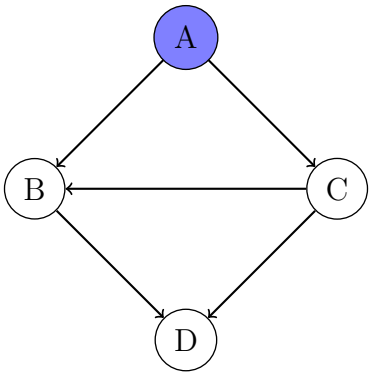

Figure 3.2: Power plant example.

As one can see, the node A is the only one which has "leaving" edges. That is why it is the only source node of this example graph.

3.2.2 Substations

Just like the power plants, substations are responsible for feeding the grid with energy. The great difference to the nodes described in the previous section is that they are, in the majority of cases, endpoints of the edges.

Another characteristic of this set of nodes is that almost all of them are associated to a number of sinks (there are few exceptions that do not have any). So, they are both responsible for feeding the inner grid (other nodes) and the outer grid (sink nodes).

Example 3 (Substation)**.** *Making little modification of last example, it is shown that nodes B and C (in green) are substations. This example also allows to show the difference between these two substations: while B only distributes energy to a sink node of the outer grid (node E), node C makes the distribution of electrical energy to both inner (towards substation node B) and outer grid (towards node D).*

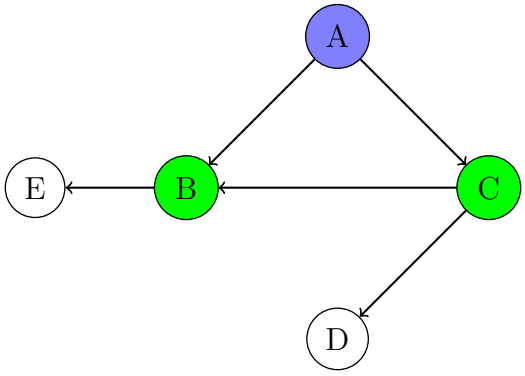

Figure 3.3: Substations example.

One can also conclude from the graph that nodes D and E are exclusive sink nodes of substation C and B, respectively. Node A is represented in blue to be coherent with the previous example: in this graph it is also considered a source node, by the reasons above explained.

3.2.3 Sectioning/Cut Posts

A sectioning post may be seen as a node that has no production or consumption. It only serves the purpose of redirecting the energy that flows through a certain edge to other nodes. As it was said before, a sectioning post is an intermediate node and therefore it is neither a source or a sink node.

Example 4 (Sectioning/Cut Posts)**.** *With an example graph similar to the one in Example 2, one can see how a sectioning post (node C in red) is different from a power plant (node A in blue) or from a substation (node B in green).*

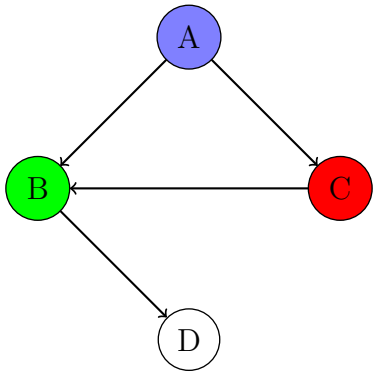

Figure 3.4: Sectioning/cut post example.

What one can denote is that node C creates another possibility for transferring the electrical energy produced by node A and that is received by node B. Instead of having only edge (A, B), this sectioning post creates the possibility of using edge (C, B).

3.2.4 Derivations

When presenting the structural elements of the electrical grid, there is one group of virtual nodes named *derivations*. Actually, this name should be addressed to a electrical line that is divided and ends in another node.

The fact that the element is considered a virtual node comes from the fact that physically, it does not exist. This set of nodes is created to ease the manipulation of data into the graph structure. It is important to refer that these nodes are neutral, meaning that they only serve the purpose of structuring the network. In terms of functionality they are very similar to the sectioning posts referred immediately above. The following example shows how a derivation works.

Example 5 (Derivation)**.** *Let (A, B) be an edge. There is a derivation on the edge (A, B) that ends in node C. The following figure shows the example physically in the grid:*

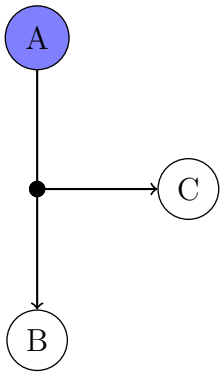

Figure 3.5: Derivation example - physically.

To represent it in the graph structure there is a little modification: a new node D (in light red) will be created between nodes A and B to represent the derivation, dividing the edge (A, B). This edge disappears and two new edges leaving node D are created:

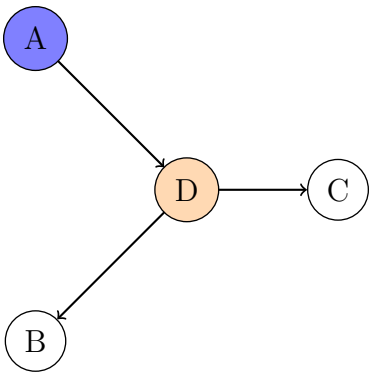

Figure 3.6: Derivation example - graph.

The rule of flow conservation is also applied and the flow that leaves A is divided through edges (D, B) and (D, C) always respecting the capacity constraints and the amount of flow that is needed in nodes B and C.

3.2.5 Sinks

The last set of structural elements of the grid are the sink nodes. These nodes can be seen as the inverse of the power plants, when speaking in terms of functionality. The power plants only produce electrical energy whereas these sink nodes are the responsible for the consumption. Consequently, as the opposite of the power plants, sink nodes are *always* the endpoints of the are connected to them.

It is important to note that the generality of the sink nodes makes part of the outer grid. However, there are some exceptions, since there exist some sink nodes in the inner grid.

Example 6 (Sinks)**.** *The graph shown below is the same as the one used in Example 4, changing the color of node D to yellow, in order to identify it as a sink node.*

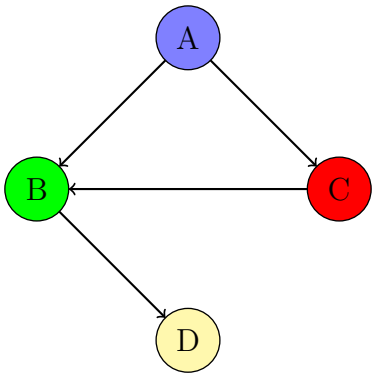

Figure 3.7: Sink example.

One can easily see that node D does not have any edge "leaving" it, which is characteristic of this type of nodes.

3.3 Problem Statement

Given the electrical grid that is the central element of this work and every structural constituent that forms it, the problem that drives this work is directly related to the flow distribution from each source (power plant) to every sink node.

Having the POS associated to each electrical line that connects a pair of nodes from the network that represents the grid, the goal is to find one or more suitable approaches that are stable enough to be prepared to:

- Compute a global POS of the network: at the same time, be able to extract the POS of each node that is included in the network, namely the sink nodes;
- Accept single- or multi-edge removal and be able to redistribute the flow, while maintaining the capability of computing every POS needed;
- Be able to compute the POS in distinct moments of time: incremental evolution of the probabilities in each moment of time by applying the algorithm iteratively;
- Relate budgetary maintenance terms, such as the cost of maintenance per kilometre, to the importance of the POS of each element (nodes and edges).

Each one of the items above presented is important to the idea of scheduling inspection and consequent maintenance. It is from the results obtained from the first three items that will be possible to understand, for example, how the POS of a certain sink node evolves throughout the time and where and when it is of greatest importance to effectively schedule an inspection

Chapter 4

Approach

This chapter describes the three different approaches to try to solve the problem in hands: section 4.1 presents the first approach using an optimization formulation; in subsection 4.1.1 it is presented the algorithmic approach based on the method of Ford and Fulkerson [7]; section 4.2 addresses the final approach that uses a non-linear constrained optimization formulation with a quadratic function of POS; the last sections are dedicated to the method of data extraction and consequent transformation to the MATLAB® environment with the help of MATLAB® engine.

Before addressing the approaches, it is important to refer that the first two do not treat exactly the real network. Instead, they perform a little modification regarding the sinks by creating a super-sink. Having the network $G = (V, E)$, the new network will be defined as $G_T = (V_T, E_T)$, where $V_T = V \cup \{S, T\}$, S is the new super-source node, T is the new super-sink node and $E_T = E \cup \{(S, s_1), \ldots, (S, s_m), (t_1, T), \ldots, (t_n, T)\}$, considering m as the number of existing source nodes - $\{s_1, \ldots, s_m\}$ - and n as the number of existing sink nodes - $\{t_1, \ldots, t_n\}$:

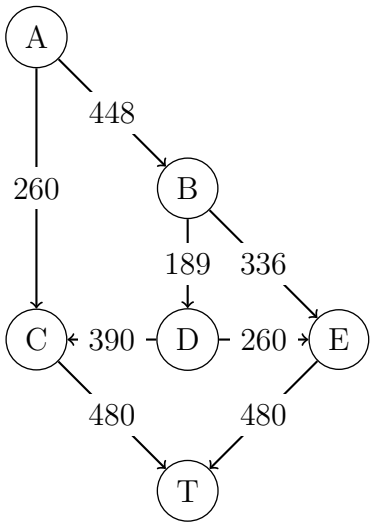

Figure 4.1: Updated network G_T with super-sink T.

To every new edge belonging to E_T will be assigned the value of 1 as probability of success because each one of them will be virtual and must *always* be traversed. Also, the capacity bound will be assumed as the energy produced by the network's source node. With this information, the network G from [2] will be updated to network G_T in Figure 4.1.

4.1 Optimization I

When talking about network-flow related problems solved in an optimization fashion, the formulations presented in subsection 2.3.1 and subsection 2.3.2 come to mind. Obviously, if one is facing a problem like this must make the necessary modifications to the optimization formulations.

If the POS is considered to be the element *cost* in this approach to the problem, it is possible to pick the formulation of subsection 2.3.2 referring to the minimum-cost problem and partially use it here. By making some changes, one can obtain a more concrete formulation for this situation.

When facing the real grid (and real data), the POS associated to each edge is computed by a quadratic function that takes into account several external factors (see section 4.2). In this first approach, however, the use of the real function is replaced by a simpler one: it is considered that the POS of a traversed edge is constant. That is, the POS of an edge traversed by one unit of flow is the same as the traversal of n units. Still, if the edge is not traversed at all (zero units of flow) the POS is automatically zero.

The assumption that the *cost* factor is considered as explained above changes the idea of the original minimum-cost problem formulation. In the original formulation there is a linear proportion between the flow traversing in each edge and the cost it takes - different quantities of flow infer different costs.

For this formulation, one must also take into account the idea of *using* an edge (which comes in favour of what was described immediately above). It is assumed that if one edge has a positive quantity of flow $0 < I_{(u,v)} \leq c_{(u,v)}$ coursing through it, it is being *used*. In order to compose the formulation, each edge will have another variable associated, x_{uv}, defined as follows:

$$x_{uv} = sign(I_{(u,v)}) \equiv \begin{cases} 1 & \text{if } (u,v) \in E \text{ has a positive flow} \\ 0 & \text{otherwise} \end{cases} \tag{4.1.1}$$

It is assumed, for this problem, that the global POS of the network, p_{global}, will depend on the used edges. That is:

$$p_{global} = \prod_{u,v \in E} p_{uv} \tag{4.1.2}$$

One way to look at the expression is to think that the global POS is equal to the product of the success rate of each used edge - if one fails, the initial flow won't get totally to the sink (consumption) nodes.

As aforementioned, the goal is to maximize the global POS (or minimize the failure),

$$\textbf{maximize} \quad p_{global} = \prod_{u,v \in E} p_{uv}, \tag{4.1.3}$$

but in optimization problems it is easier to work with summations than with products. So, the maximization of the product of Equation 4.1.2 can be transformed into the minimization of the symmetric of a summation of logarithms, which is done with respect to the symmetric because the logarithm of values in $]0, 1]$ is non-positive.[1]:

$$\begin{aligned} \textbf{minimize} & -\log(p_{global}) \\ & = -\log\left(\prod_{u,v \in E} p_{uv}\right) \\ & = -\sum_{u,v \in E} \log(p_{uv}) \end{aligned} \tag{4.1.4}$$

The POS logarithm of an edge will only be taken into account if that edge is being used, where $x_{uv} = 1$. On the other hand, it does not depend on the quantity of flow that courses through the edges. Therefore it is possible to formulate the following objective function:

$$\textbf{minimize} \quad F(x_{uv}) = -\sum_{u,v \in E} x_{uv} \log(p_{uv}) \tag{4.1.5}$$

Since the use of the edges must be introduced to the objective function it wouldn't be viable to have a variable $x_{uv} = 0$ multiplying in a product. This would lead to a global value of 0 every time an edge of the network is not used, and it is the reason why the variable x_{uv} defined in Equation 4.1.1 will only appear when formulating the objective function above. Also, if it appeared in the product, it would also be inside the logarithm, as:

$$\prod_{u,v \in E} x_{uv} \cdot p_{uv} \Rightarrow \sum_{u,v \in E} \log(x_{uv} \cdot p_{uv}) \tag{4.1.6}$$

It is trivial to notice that this would lead to cases outside of the domain of the logarithm function, when $x_{uv} = 0$, and the optimization would automatically be compromised.

Whichever is the computed combination of edges to traverse in order to minimize the function F they all must respect the rules of capacities constraints and flow conservation presented in section 2.2. Using the formulation of the minimum-cost problem, the

[1] To understand the relation between the summation and the product, one of the logarithms rules is announced: $\log(a \times b) = \log a + \log b$.

optimization formulation of this approach is stated as follows:

$$\begin{aligned} \textbf{minimize} \quad & F(x_{uv}) = -\sum_{u,v \in E} x_{uv} \log(p_{uv}) \\ \textbf{subject to} \quad & Af = -v_0 d \\ & f(u,v) \leq c(u,v) \qquad \forall_{(u,v)\in E} \\ & f(u,v) \geq 0 \qquad \forall_{(u,v)\in E} \end{aligned} \tag{4.1.7}$$

where d is defined in Equation 2.3.3, A is the node-edge incidence matrix of the network G and v_0 is defined as the expected flow to be produced by the source nodes and consumed by the sinks: following the network of Figure 3.1, the production of node A is assumed to be of 480 units. Thus, for that network, $v_0 = 480$.

This formulation must still take into account the node capacities constraints showed in Example 1, in order to avoid a possible loss of flow that enters into a node with a lower limit of production/consumption.

4.1.1 Algorithmic

One other way to address the problem is to apply the ideas that form the Ford-Fulkerson method and refining the search for an augmenting path. While Ford and Fulkerson don't have any pattern of searching for an augmenting path, Edmonds and Karp refined that part of the method by applying a BFS.

The BFS of the Edmonds-Karp algorithm is done according to the length of the path. In this approach the search for the path is made taking into account its *cost*. The *cost of the path* is considered to be the total POF of the path itself in the logarithm form.

The approach is then divided into three steps:

1. Sort available paths by *cost* and choose the best.
2. (a) Update the residual network by increasing the flow through the augmenting path chosen until there is none;
   (b) Repeat step 1.
3. Apply the Edmonds-Karp algorithm to the set of edges obtained in step 2.

After the second step of the approach, there is a possibility that the updated network has a non-optimal POS, when comparing the result with the Optimization I approach. To optimize the result the BFS of Edmonds-Karp is then performed, so that there is a certainty that the shortest paths (and cheapest, in terms of probability) are traversed. Below, there is a detailed explanation on each of the steps performed.

Sorting Paths

Each path is constituted by a set of edges. Each one of these edges $(u, v) \in E$ has associated a POF, p_{uv}. On the other hand, for purposes of coherence with the optimization approach, it is possible to calculate the negative logarithm of each p_{uv}, which will always be a non-negative value. Consequently, the total *cost* of a path, w_{path}, will be obtained by an expression similar to the summation in Equation 4.1.2:

$$w_{path} = -\sum_{u,v \in path} \log(p_{uv}) \tag{4.1.8}$$

and it will also be non-negative.

By computing all augmenting paths (and its weights) from source to sink it is possible, after that, to sort them by cost and choose the best one.

Updating the Network

After choosing the preferred augmenting path, the residual network is updated with the flow with respect to that path. Then, the sorting phase is again applied to the newly updated network: since new paths can be formed in the residual network after the update of a flow, it is necessary to perform the calculation of new paths' cost. It is important to notice that if the original network has a certain edge (u, v) and one respective residual network has both (u, v) and (v, u) edges, the POF is the same if the flow traverses in a way or another: $p_{uv} = p_{vu}$. However, even having the same POF in both ways, if there is a flow cancelling from one of the directions, that POF must automatically be discarded. This cycle between updating the residual network with new flows and choosing another augmenting path occurs until there is none, just like it happens in Ford-Fulkerson method.

Application of Edmonds-Karp Algorithm

Assuming this is the third step or phase of the approach, what is obtained from the previous ones is a new network that will only have the edges traversed by the chosen augmenting paths of step 2. Even obtaining a maximum-flow, it is possible to compute a final network with a non-optimal POS. In order to solve this problem, it is applied the BFS of Edmonds-Karp algorithm over the new network, which contains only the necessary edges. Within the so called necessary edges, there will be with certainty the shortest paths as well as the cheapest in terms of the logarithms of the POS.

After the application of the BFS, the flow is expected to be maximal and the POS is also optimal.

In Appendix A, a detailed example is shown, so that each step of the algorithmic approach can be understood and seen when being applied directly to a network.

The Algorihm

A summarized idea of the algorithmic approach is presented as follows:

Algorithm 2 `Algorithmic approach`

Input: Network $G_T = (V_T, E_T)$
Output: Network G''_T updated with optimal flow - $|f_{max}|$ - and global POS - p_{global}

```
1  while no maximum-flow do
2    while exists any augmenting path do
3      Compute G_T augmenting paths and its cost.
4    end while
5    Sort augmenting paths by its cost;
6    Choose best augmenting path - p;
7    E ← {(u, v) : (u, v) is on p};
8    Update the residual network of G_T - G'_T - with the flow c_f(p);
9  end while
10 Remove unused edges from G_T in the while-cycle - G''_T = (V_T, E_T − E);
11 Apply Edmonds-Karp algorithm to G''_T;
12 return (G''_T, |f_max|, p_global).
```

4.2 Optimization II - Final

This second optimization approach takes into account two new elements of true importance: the real function of probability and the redundancy of edges that connect the same pair of nodes.

In the previous optimization approach, the POS of a certain edge would be a simple constant. It would appear on the problem formulation depending whether the edge was traversed by a non-negative amount of flow.

This approach uses the real function. To be more precise, the POS of an edge $(i, j) \in E$ will be computed by a quadratic function[2] $p : \mathbb{R}_0^+ \to [0, 1]$, as follows:

$$p(f(i,j)_k) = 1 - \underbrace{\left(k_0 + k_1 f(i,j)_k + k_2 f(i,j)_k^2\right)}_{\text{POF}} \tag{4.2.1}$$

where $f(i, j)_k$ represents the flow in the edge $(i, j)_k$. It is this subscript k that indicates the presence of redundancy in the edges: k indicates the index of the edge between the nodes i and j. If there is only one edge connecting those nodes, k will only assume one possible value, 1; if there are two edges, k may assume the value 1 or 2; and so forth.

On the other hand, the constants $k_i, i \in \{0, 1, 2\}$ are related to the external events and factors that can cause a line failure:

[2]The function p has one argument for the flow intensity, which is always non-negative. Also, a probability value is always comprised between 0 and 1.

- k_0 is associated to the line *age*; thus it will have an increased value when facing the older lines;
- k_1 is more relevant when facing a line used during the night, since it is precisely when there exists a greater need and consumption of energy;
- k_2 is directly related to the proximity of vegetation. The association of this factor to the square of the intensity happens because of the importance that vegetation has in the possibility of line failures.

The following plots help to understand how the function evolves with an increase of flow intensity with two different values for the constants. Although in reality k_i assumes different values for different i, these plots serve only as example:

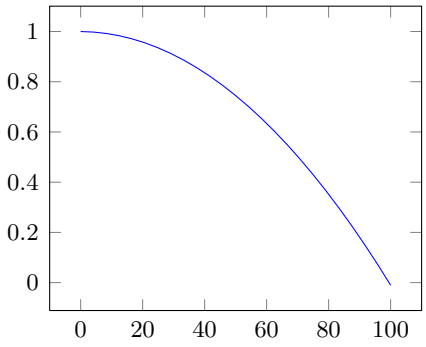

Figure 4.2: $k_i = 10^{-4}$

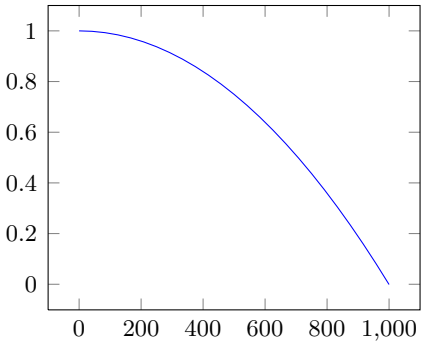

Figure 4.3: $k_i = 10^{-6}$

The plots above represent the function of Equation 4.2.1 with different values for k_i. The values for these constants were chosen *ad-hoc* in the order of 10^{-6} but, in reality, these constants vary from edge to edge. Also, depending on the event (POI) they represent, they can fluctuate between two or three orders of greatness - 10^{-4} to 10^{-6}. However, if $k_i = 10^{-4}$ for every i, the maximum value for the intensity is less than 100 units: for an intensity of 100 units, the POS would be negative - the function zero is approximately 99.5 (see Figure 4.2). On the other hand, if $k_i = 10^{-6}$ for every i, the maximum value for any edge is nearly 1000 as the function zero is approximately 999.5 - Figure 4.3. To facilitate the optimization, $k_i = 10^{-6}$ is considered to be the same for every i and it is independent from the edges.

It is quite simple to understand what will happen when this function is applied to the intensities of the several edges. Independently from the number of redundant connections between pairs of nodes, this function will always "privilege" the distribution of the flow through the maximum number of those connections.

As it was previously denoted, the idea of this problem is to promote the highest success of the sink nodes. Until now, only the edges would have associated a POS. In this formulation, however, there is an association of POS to each one of the network nodes p_i, where $i \in V$.

Assumption 3. *It is assumed that a source node has no failures, that is, a source node is always successful when transferring energy:*

$$p_i = 1, i \in S, \tag{4.2.2}$$

where S is the set of source nodes;

Assumption 4. *The POS of an intermediate (or final) node*[3] *depends on the POS of the arriving lines - $f(j,i)_k$ - and on the nodes from where those lines depart - p_j:*

$$p_i = \prod_{j \in \Gamma(i)} \left(p_j \times \prod_{k \in \mathcal{N}(j,i)} p(f(j,i)_k) \right), \tag{4.2.3}$$

where nodes $i, j \in V$, the edge $(j,i) \in E$, $\Gamma(i)$ is the set of nodes from where edges leave towards node i, and $\mathcal{N}(j,i)$ is the set of indexes of redundant edges between nodes j and i.

The following examples illustrate how the formula from Assumption 4 (together with the rule of Assumption 3) is applied.

Example 7. *One of the simplest possible graph is shown below, where the purpose is to show how to compute the POS of node B.*

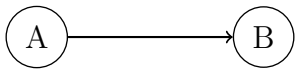

Figure 4.4: Graph of *Example 7*.

In this case node A is a source node, thus, $p_A = 1$. p_B would be simply a result of the multiplication of the POS of both node A and edge (A,B):

$$p_B = p_A \times p(f(A,B)_1). \tag{4.2.4}$$

Example 8. *Below it is presented a graph with two source nodes and a sink node. There is an edge connecting each one of the source nodes A and B to the sink node C.*

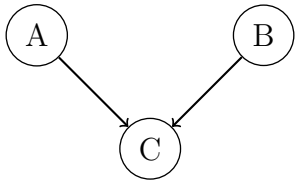

Figure 4.5: Graph of *Example 8*.

[3] *Intermediate* and *final* nodes include substations, derivation, sectioning/cut posts and sinks.

In this case, $p_A = p_B = 1$ because nodes A and B are source nodes.p_C would result from the multiplication of the POS of node A, edge (A, C), node B and edge (B, C):

$$p_C = p_A \times p(f(A, C)_1) \times p_B \times p(f(B, C)_1). \tag{4.2.5}$$

Example 9. *The last example shows a pair of redundant edges from source node A to sink node C. The graph also has a source node B with direct influence in the POS of (intermediate) node D. The POS of sink node C will depend also from that obtained from D.*

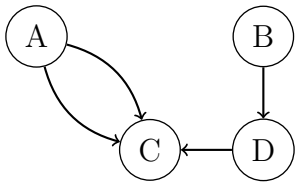

Figure 4.6: Graph of *Example 9*.

The list of probabilities of success is then:

$$\begin{aligned} &p_A = p_B = 1 \\ &p_D = p_B \times p(f(B, D)_1) \\ &p_C = p_A \times p(f(A, C)_1) \times p(f(A, C)_2) \times p_D \times p(f(D, C)_1). \end{aligned} \tag{4.2.6}$$

In this example, the set $\mathcal{N}(A, C)$, which appears in the general formula of Equation 4.2.3, would be defined as $\mathcal{N}(A, C) = \{1, 2\}$, because there are two different edges connecting nodes A and C. In the previous examples the sets $\mathcal{N}$ were defined by only one element - $\{1\}$.

As one can see, the product of Equation 4.2.3 is to become a constraint of the optimization problem. In this kind of problems it is always easy to work with summations than with products because this set of summations will be entering the matrix-form of the optimization problem formulation. The immediate suggestion is to make this process of transformation via logarithms, that is, replace the variable p_i by

$$l_i = \log p_i. \tag{4.2.7}$$

Using the logarithms rule enunciated in section 4.1 where the logarithm of a product is the sum of the logarithms, one can make the process of transformation viable. Also, the transformation is possible because the result obtained by optimization of a product is the same as the one obtained by optimizing its logarithm, assuming that, in the end of the

process, the base of the logarithm used is powered to the value obtained. Consequently:

$$\begin{aligned} l_i = \log p_i &= \log \left(\prod_{j \in \Gamma(i)} \left[p_j \times \prod_{\substack{k \in \\ \mathcal{N}(j,i)}} p(f(j,i)_k) \right] \right) \\ &= \sum_{j \in \Gamma(i)} \left(\log \left[p_j \times \prod_{\substack{k \in \\ \mathcal{N}(j,i)}} p(f(j,i)_k) \right] \right) \\ &= \sum_{j \in \Gamma(i)} \left(\log p_j + \sum_{\substack{k \in \\ \mathcal{N}(j,i)}} \log p(f(j,i)_k) \right) \\ &= \sum_{j \in \Gamma(i)} \log p_j + \sum_{j \in \Gamma(i)} \sum_{\substack{k \in \\ \mathcal{N}(j,i)}} \log p(f(j,i)_k) \\ &= \sum_{j \in \Gamma(i)} l_j + \sum_{j \in \Gamma(i)} \sum_{\substack{k \in \\ \mathcal{N}(j,i)}} \log p(f(j,i)_k) \end{aligned} \tag{4.2.8}$$

After the product manipulation into a summation, it is possible to also transform this expression into a set of non-linear equality-constraints of the form $c_i(x) = 0$, as it will be presented in the next subsection when making the problem formulation.

Before addressing the final formulation of the optimization problem, it is important to add another element for the problem to be solvable.

In every network-flow problem, this included, the flow conservation is an indispensable element to take into account. Being so, the notion of the node-edge incidence matrix must be used along with the vector of flow intensity of each edge. This guarantees that every node has a net flow of zero (except for the source nodes - exclusively "produce" the flow - and for the sink nodes - exclusively "consume" the flow), which means that they send the same amount of flow that they receive.

4.2.1 Formalizing the problem

When facing an optimization problem, the first thing to know is if the objective function or any of the constraints is non-linear. This will define the idea of linear or non-linear constrained optimization problem.

From Equation 4.2.8 the first conclusion is that the problem is non-linear. Hence, the `fmincon` procedure from MATLAB® is the right solver for this kind of problem (see section 4.3).

In the previous subsection, it was mentioned that the purpose of the problem was that sink nodes could receive the supposed amount of flow. However, the problem must also

take into account the POS that results from getting the maximum possible amount of flow to those nodes. This is a more important factor because of the risks associated to the maintenance of the lines connecting the nodes. Thus, the objective function to optimize will be directly bound to the overall POS of the network. In other words, the POS of the sink nodes.

Following the idea of Assumption 4, the objective function F can be computed as follows:

$$\prod_{i \in T} p_i, \tag{4.2.9}$$

where T is the set of sink nodes.

As it was performed in Equation 4.2.8, the objective function F will also be used in its logarithmic form and the purpose is to maximize the POS of the sink nodes. Thus:

$$\begin{aligned} \textbf{maximize} \quad F &= \log\left(\prod_{i \in T} p_i\right) \\ &= \sum_{i \in T} \log p_i \\ &\overset{Equation\ 4.2.7}{=} \sum_{i \in T} l_i \end{aligned} \tag{4.2.10}$$

Adding the constraints from Equation 4.2.8, from the flow conservation rule as well as the bounds for the variables, the problem will have the following form:

$$\begin{aligned} \textbf{maximize} \quad & F = \sum_{i \in T} l_i \\ \textbf{subject to} \quad & l_i - \sum_{j \in \Gamma(i)} l_j - \sum_{j \in \Gamma(i)} \sum_{\substack{k \in \\ \mathcal{N}(j,i)}} \log p(f(j,i)_k) = 0 \\ & \sum_{j \in \Gamma(i)} \sum_{\substack{k \in \\ \mathcal{N}(j,i)}} f(j,i)_k \leq c_i \\ & Af \leq -v_0 d \\ & 0 \leq f(i,j) \leq c(i,j) \\ & l_i \leq 0 \end{aligned} \tag{4.2.11}$$

The first line of the constraints is related to the non-linear equalities and refers to the POS associated to each node, as presented in Equation 4.2.8. The second one refers to the nodes capacities: the sum of the flow intensity of every edge that ends in a certain node must not be greater than the capacity of that node. The third line regards the flow conservation by using the notation of the node-edge incidence matrix A (from Definition 8). It is multiplied by the vector of the intensities - f. The vector v_0 indicates the

supposed net flow for each node (represented in each component of the vector) and d is defined in Equation 2.3.3.

The last two lines are the definition of the variable bounds: each edge has a non-negative intensity value that must not exceed its capacity - $c(i, j)$. On the other hand, variables l_i are non-positive. This is due to the fact that each one of them depends on the computation of function p - Equation 4.2.1. As it was aforementioned, the co-domain of p lies between 0 and 1. Consequently, the logarithm of that function[4] will be, at most, equal to 0:

$$l = \log p :]0, 1] \rightarrow \mathbb{R}_0^- . \tag{4.2.12}$$

4.3 Formalization in MATLAB®

The notation used for `fmincon` procedure was already presented above, in Equation 2.4.10. The syntax used to call this procedure is the following:

```
[x fval ...]  = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

where `fun` is the function to optimize, `x0` is the point for the solver to start attempting to find a minimizer, `A, b, Aeq, beq, lb` and `ub` represent the notation used in Equation 2.4.10, `nonlcon` is the MATLAB® function where the vectors of non-linear constraints are defined and `options` is a structure where the algorithm options (such as tolerance values and the type of solver to use) are set.

On the other hand, `[x fval ...]` represents the output for the problem: `x` represents the solution vector, whereas `fval` is the value of the objective function `fun` at the solution `x`; the use of "..." is due to the fact that some more output elements can be set, such as `exitflag`, which describes the exit condition of the procedure, or `output`, which represents a structure with more detailed information about the optimization.

4.3.1 Objective Function

The actual expression for the objective function must be updated to also use the n variables of the edges intensities plus the m node variables regarding their probability of success. This is imperative when passing the actual formulation to its matrix form in the MATLAB® environment, because every component of the objective vector must have a value. It is also needed to invert the signal of the function. Equation 4.2.9 was stated as a *maximization* problem, whereas the `fmincon` procedure only solves the problem as a *minimization* one. Then, the actual objective function will be stated as:

$$F = -\sum_{i \in T} l_i - \sum_{i \notin T} 0 \times l_i - \sum_{\substack{(i,j) \in E, \\ k \in \mathcal{N}(i,j)}} 0 \times f(i,j)_k. \tag{4.3.1}$$

[4]The logarithm function does not assume values when the argument is 0.

In MATLAB® the vector `fun` will hold the coefficients of the $m + n$ variables. Thus, it will be a $m + n$ vector and it will be defined component-wise - $\texttt{fun}_i$ - as:

$$\texttt{fun}_i = \begin{cases} -1 & i \in T \\ 0 & \text{otherwise} \end{cases} . \tag{4.3.2}$$

4.3.2 Linear Inequalities

Since the formulation of the problem does not use any set of linear equalities, the matrix `Aeq` and the vector `beq` are empty - `[]`. On the other hand, the matrix `A` and the vector `b` have a very important role in this optimization problem.

In the `fmincon` procedure there is only a matrix for the linear inequalities - A. Consequently, the two sets of linear inequalities presented in Equation 4.2.11:

$$\sum_{j \in \Gamma(i)} \sum_{\substack{k \in \\ \mathcal{N}(j,i)}} f(j,i)_k \leq c_i \tag{4.3.3}$$

$$Af \leq v_0 \times d \tag{4.3.4}$$

must be composed into a unique matrix - `A`.

Matrix `A` will hold the information regarding the left-side of the linear inequalities constraints of Equation 4.3.3 and Equation 4.3.4. The optimization problem has $m + n$ variables - the first m represent node variables, whereas the remaining n are edge variables. Due to this fact, `A` will be an $(m + n)$-column matrix.

Flow Conservation

The first m rows of matrix `A` are related to the information that regards directly to the flow conservation rule. These m rows correspond directly to each node.

`A` has $m+n$ columns. However, the first m columns are related to the nodes variables l_i that do not influence the flow conservation. Consequently, the elements of the first m columns of the matrix are all set to 0. The real idea of node-edge incidence matrix can be seen in the second set of columns of `A`, from columns $m+1$ to $m+n$. Here, the entries $\texttt{A}_{ij}$ are set to 1 or -1, whether if i is the number of a start or end node of the edge numbered $j - m$.

Nodes Capacities

Equation 4.3.4 exists due to the fact that most nodes have a limit of flow that can receive form the arriving edges. To add this set of constraints to the first n rows of `A`, one must count the number of nodes that have this limit. Since the source nodes do not

have any arriving edges, they do not enter to this set of constraints. If k is the number of source nodes, then the number of constraints is equal to $m - k$, which is the number of rows that must be added to matrix `A`.

Each row represents the constraint of a node's capacity and the entries $\mathtt{A}_{ij} = 1$ whenever the edge $j - m$ ends in that node. Otherwise, $\mathtt{A}_{ij} = 0$.

4.3.3 Variable Bounds

The last two sets of constraints of Equation 4.2.11,

$$0 \leq f(i,j) \leq c(i,j) \tag{4.3.5}$$

$$l_i \leq 0, \tag{4.3.6}$$

represent the bounds of the variables. In MATLAB® the vectors `lb` and `ub` are the responsible for holding the lower and upper bounds of the variables. These two vectors have $m + n$ components and they are defined as follows:

$$\mathtt{lb}_i = \begin{cases} -\infty & i \leq m \\ 0 & \text{otherwise} \end{cases}, \tag{4.3.7}$$

$$\mathtt{ub}_i = \begin{cases} 0 & i \leq m \\ c_i & \text{otherwise} \end{cases}, \tag{4.3.8}$$

where c_i represents the maximum intensity that can be sent through the $(i-m)^{\text{th}}$ edge.

4.3.4 Non-linear Equalities

It is in the MATLAB® function `nonlcon` that the non-linear constraints are defined. In its simplest form, `nonlcon` accepts a vector (the vector of variables) and returns two different vectors, `c` and `ceq`. Vector `c` contains the non-linear inequalities and vector `ceq` contains the non-linear equalities. When passed as argument to `fmincon`, `nonlcon` must be specified as a MATLAB® function handle to a file or to an anonymous function (with the `'@'` symbol). For example:

```
[x fval ...]  = fmincon(fun,x0,A,b,[],[],lb,ub,@nonlcon,options).
```

For this problem `nonlcon` can be defined in a file named `nolcon.m` with the following structure:

```
function [c, ceq] = nonlcon(x);
c = [];
```

```
ceq = ...;
```

The fact that the `fmincon` procedure uses gradient approximations when solving the optimization problems can make it a little inaccurate. Consequently, the function handle `@nonlcon` allows the computation of the gradients of the constraints directly by the user. In this case, the `nonlcon` will return two different matrices beside the vectors `c` and `ceq`: `gradc` represents the matrix of the gradients of the non-linear inequalities (empty in this problem) and `gradceq` is the matrix of the gradients of non-linear equalities. Thus, the file `nonlcon.m` would have the following structure:

```
function [c, ceq, gradc, gradceq] = nonlcon(x);
c = [];
ceq = ...;
if nargout > 2
   gradc = [];
   gradceq = ...;
end
```

where the variable `nargout` represents the number of different elements of the output (in this case, four - `c, ceq, gradc` and `gradceq`).

As it was previously said, the matrix `gradceq` will hold the information of the gradients of each non-linear equality constraint. It is defined as:

$$\texttt{gradceq} = \begin{bmatrix} \frac{\partial c_1}{\partial x_1} & \cdots & \frac{\partial c_k}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial c_1}{\partial x_{m+n}} & \cdots & \frac{\partial c_k}{\partial x_{m+n}} \end{bmatrix}, \tag{4.3.9}$$

assuming that there exist k non-linear equalities.

When computing the partial derivatives of each constraint in order to a certain variable, one must pay attention to the term $\log p(I_{(j,i)_k})$. The derivative of that term in order to the intensity of the edge is defined as:

$$\begin{aligned} \frac{d}{df(j,i)_k}\left(\log p(f(j,i)_k)\right) &= \frac{d}{df(j,i)_k}\left(\log\left(1-\left(k_0+k_1 f(i,j)_k + k_2 f(i,j)_k^2\right)\right)\right) \\ &= \frac{2k_2 f(j,i)_k + k_1}{k_2 f(i,j)_k^2 + k_1 f(i,j)_k + k_0 - 1}. \end{aligned} \tag{4.3.10}$$

The derivatives of the other terms of the constraints will either be 1 or -1, whether it is l_i or one of the terms of the sum $-\sum_{j \in \Gamma(i)} l_j$.

Obviously, when the variable of the derivative is not in the constraint, the derivative will automatically be 0.

4.3.5 The `options` Structure

The minimization is made taking into account the options set in the `options` structure. It is the procedure `optimset` that sets these options:

```
options = optimset('option_name', 'value');
```

where `option_name` is the option and `value` is the value setting that option.

There are several options to choose from, since the algorithm to be used (the default is not SQP), the tolerance values for the problem, the number of iterations of the `while`-cycle of the SQP framework algorithm, *etc.* A simple example is to set the algorithm to SQP:

```
options = optimset('Algorithm', 'sqp');
```

4.4 Data Extraction

4.4.1 Contents

The data that contains the information regarding the electrical grid is distributed through a set of spreadsheets:

- *Circuits table:* contains information about the existing electrical lines and it is divided into two sections:
  - *Circuits:* in this section the electrical lines are identified by its full name, by a specified identifier (ID) of the format `Lxxxx`, where `x` represents a number, and by each node that composes the line;
  - *Nodes:* each node has a number ID that also identifies it in the previous section; this second section is constituted by the list of pairs "number ID-node".
- *Nodes:* contains information related to each one of the structural elements of the grid:
  - *Power plants:* section that describes each one of the power plants - number ID, name, installed power of production;

  - *Substations:* section that gives information related to each one of the substations - number ID, name, installed power of production and number of associated sinks;
  - The other sections are *derivations*, *border nodes*[5] and sectioning/cut posts. The nodes of these sections only have their number ID and name because they do not have power of production or associated sinks. Every section has other elements of information that are not relevant for this problem.

- *Capacities:* this last spreadsheet had information regarding the maximum capacities of each electrical line. Each one of the lines was also identified with the same ID presented in *Circuits table* spreadsheet - `Lxxxx`.

4.4.2 Extraction

Not every electrical line that is represented in the *Circuits* section of the first spreadsheet is available in the real network. It was the last spreadsheet - *Capacities* - that had information about the lines capacities bounds. Consequently, only the lines represented in *Capacities* are considered to the optimization problem.

Since both files use the same identifiers for the lines, it was easy to eliminate those that appear in the section *Circuits* so that they would not be considered.

4.5 MATLAB® engine

MATLAB® engine is a library that contains routines that allow one to call the MATLAB® software from one's programs in C/C++ or Fortran. The engine programs communicate with a MATLAB® process via pipes, on UNIX systems.

After getting the spreadsheets with the data, the idea was to transform it into the matrix form of MATLAB®, which is explained in the next section. With this engine it was possible to program all data extraction and parsing in C while the back end analysis was performed in MATLAB®, with the specific prodecures of MATLAB® toolboxes. The engine has a set of C routines that allow the programmer to start or close a session, to send MATLAB® arrays to the engine or even execute specific MATLAB® commands:

- `engOpen:` start a MATLAB® engine session;
- `engClose:` close a MATLAB® engine session;
- `engPutvariable`: send a MATLAB® array to the engine;
- `engEvalString`: execute a MATLAB® command.

[5]Nodes that make the connection with Spain; it is considered that they have infinity capacity of consumption.

The list of C functions is not constituted only by these four. However, these were the ones that were exclusively used to program the data extraction and consequent transformation to the MATLAB® environment.

4.5.1 Sending Arrays

The syntax of `engPutVariable` is as follows:

```
#include "engine.h"
mxArray *engPutVariable(engine *ep, const char *name, const mxArray *pm);
```

where `ep` is the engine pointer, `name` is the name of the `mxArray` in the MATLAB® workspace and `pm` is the `mxArray` pointer. On the other hand, `mxArray` is a C language opaque type and it is the fundamental type underlying MATLAB® data.

Before copying data to the `mxArray` type, one creates a common `double` variable and manipulates data into this variable. After the variable has all the needed data, one creates the `mxArray`. In MATLAB®, almost every number variable is in the matrix form and thereby when creating the `mxArray`, one uses the function `mxCreateDoubleMatrix`, which has the following syntax:

```
#include "matrix.h"
mxArray *mxCreateDoubleMatrix(mwSize m, mwSize n, mxComplexity Flag);
```

where `m` is the number of rows and `n` is the number of columns of the created matrix; `Flag` is a flag that determines whether the `mxArray` is complex (has elements with imaginary part) or not.

To copy the contents from the original `double` pointer to the `mxArray` one can simply use the `memcpy` function of the `string.h` C library. The syntax would be the following:

```
mxArray *var;
double *Cvar;
...
var = mxCreateDoubleMatrix(m, n, mxREAL);
memcpy((void *)mxGetPr(var), (void *)Cvar, sizeof(Cvar));
```

where it is assumed that in `"..."` the `double` variable `Cvar` is manipulated with the extracted data and the function `mxGetPr` returns the real[6] data of the `mxArray` that `var` points to.

[6]The *real* part of a number (even if is a complex one).

4.5.2 Executing Commands

To complement the previous command - `engPutVariable` - the engine provides the function `engEvalString` to evaluate the string that is given as argument as if it was a command to execute inside the MATLAB® environment. The syntax of this function is the following:

```
#include "engine.h"
int engEvalString(engine *ep, const char *string);
```

where `ep` is the engine pointer and `string` is the command to execute. It returns `1` when the MATLAB® session is no longer running or the engine pointer is invalid or with the value `NULL`; otherwise it returns `0` even if it cannot evaluate `string` correctly.

For example, if it is important to perform a simple arithmetic computation, one should use the following:

```
engEvalString(ep, "1+2+3");
```

On the other hand, to transform the data from the spreadsheets into the matrix form of MATLAB®, one must assign a variable to the arguments of the `fmincon` procedure that are in the form of matrices (except for the unused matrices `Aeq` and `beq`). Consequently, the arguments for `fmincon` that would need a variable assignment would be `x0`, `A`, `b`, `lb` and `ub`.

On the other hand, the argument `fun` represents an anonymous function. However, it needes a variable `obj` that was presented in subsection 4.3.1, what implies that `obj` would also need a variable assignment.

After all the variables were assigned, the function `engEvalString` was used to create the `options` structure with detailed information about the algorithm iterations,

```
engEvalString(ep, "options = optimset(‘Algorithm’, ‘sqp’, ‘Display’,
‘iter-detailed’);");,
```

and ultimately to save the workspace into a `.mat` file:

```
engEvalString(ep, "save(‘problem.mat’);");.
```

After this last instruction, the file `problem.mat` would contain the information of every variable needed to the problem. This last decision was made so that any modification could be done directly inside the MATLAB® environment. This allows the use of ev-

ery MATLAB® toolboxes available and avoids code compilation every time one needs to change an element of any variable of the problem.

In Appendix B, the function responsible for sending the parsed data from the spreadsheets to MATLAB® is completely listed.

Chapter 5

Evaluation

This chapter is dedicated to the discussion of the obtained results mainly by applying the MATLAB® `fmincon` procedure with the formulation of the last approach - Optimization II - to the data set that represents the real network. However, it will also show the differences between applying the different approaches to a minor example.

5.1 Differences

The algorithmic approach was only tested for small examples and it is assumed that it has the same results as the first optimization approach. This happens due to the fact that both of them are based on the same principle: the POS was computed taking into account the *use* of an edge and a predefined POS. The great difference between these two approaches is the upper bound on the theoretical time they would take to solve a problem.

Leaving the algorithmic approach aside, it is important to compare the optimization approaches:

- *Computation of the POS:* in the first approach a predefined value is used whereas in the second approach the real value is used - computation of the quadratic function represented in Equation 4.2.1;
- *Objective function:* in Optimization I, the objective function was a sum of the logarithms of the POS of every *used* edge; in the final optimization approach, the objective function is a sum of the logarithms of the POS of each one of the sink nodes;
- *Probability associated to nodes:* in the final approach, following Assumptions 3 and 4, every node has an associated POS: it is assumed that every element in a path from source to sink node contributes to the POS; this does not happen in the first approach, where the probability is confined to the edges.

5.2 Test Graph

5.2.1 Integral Graph

Every approach was initially tested using a simple modification of the example graph from [2],

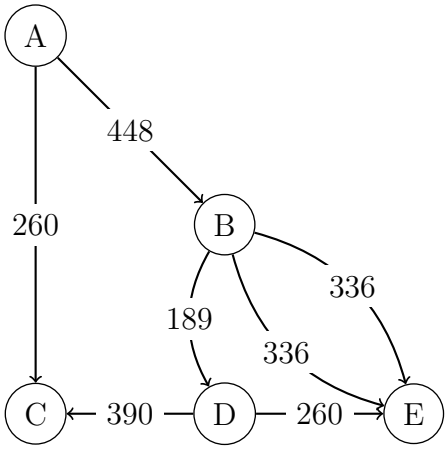

Figure 5.1: First test graph.

where each edge label represents its upper capacity bound, and one can denote the redundancy that was added between nodes B and E - lines BE_1 and BE_2.

As it was mentioned in the start of chapter 4, the first two approaches - Optimization I and Algorithmic - use the modified graph with a super-sink T and obtain the same results. Consequently, the following table[1] shows the results for both optimization approaches:

Approach	**Edge POS**	$\mathbf{k_i}$	**Total Edges**	***Used* Edges**	**Global POS**	**Node C POS**	**Node E POS**
Opt II	Equation 4.2.1	10^{-6}	7	6	86.7%	94.2%	92.0%
Opt I	99%	-	7	3	97%	99%	98%
	98.5%				95.6%	98.5%	97.0%
	97.5%				92.7%	97.5%	95.1%
	95%				85.7%	95%	90.2%
	90%				72.9%	90%	81%

Table 5.1: Running both optimization approaches for the graph of Figure 5.1.

From the previous table it is possible to infer that with a POS of 97.5% for each edge, the result is already lower than the one obtained with the real quadratic function of Equation 4.2.1. Other important aspect is the difference between the number of *used* edges:

[1] When the approach is the Optimization I, the *used* edges do not count the virtual edges connected to the super-sink (even if the result presents that they have flow coursing through them); k_i represents the constants used in the quadratic function of probability.

while Optimization I is more worried with the rule of flow conservation, Optimization II tries to maximize the distribution throughout the number of available edges, so that the POF can decrease (also respecting the rule of flow conservation). The following graphs represent how the flow is distributed according to the results presented in the table:

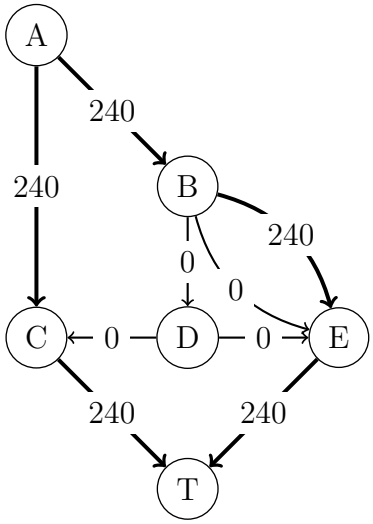

Figure 5.2: Optimization I results for Table 5.1.

and

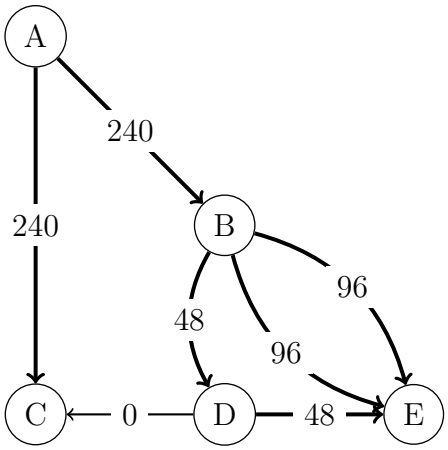

Figure 5.3: Optimization II results for Table 5.1.

where the labels are the units of flow coursing through the edges. With different distributions, both approaches show that nodes C and E get the desired 240 units of flow, performing the total of 480 that is produced in node A.

While the Optimization I uses a set of predefined values for each edge POS, Optimization II also assumes that the constants k_i of Equation 4.2.1 have the value of 10^{-6}. This last value is not the same for every edge, but it is much nearer to reality than the values used for Optimization I.

5.2.2 Removing Edges

One of the goals to achieve when dealing with this problem was the possibility of removing edges in certain moments of time. This situation can happen when there is maintenance occurring on an edge and thereby it needs to be taken out of the grid momentarily. For example, from the previous example graph it is possible to remove one of the edges that connect the nodes B and E (obtaining the original graph of [2]):

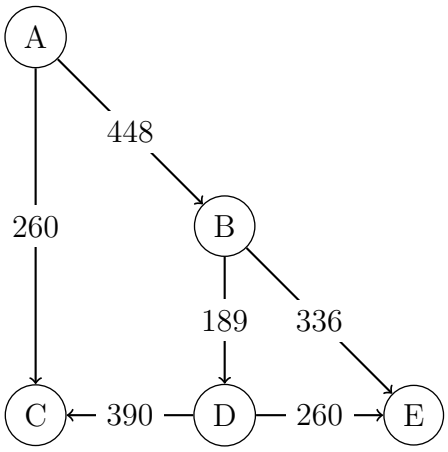

Figure 5.4: Graph of Figure 5.1 with one edge removed.

With the redundancy removed, the second approach is then "obliged" to redistribute the flow. The following table presents the results from running both optimization approaches to the graph above:

Approach	**Edge POS**	$\mathbf{k_i}$	**Total Edges**	***Used* Edges**	**Global POS**	**Node C POS**	**Node E POS**
Opt II	Equation 4.2.1	10^{-6}	6	5	85.4%	94.2%	90.6%
Opt I	99%	-	6	3	97%	99%	98%
	98.5%				95.6%	98.5%	97.0%
	97.5%				92.7%	97.5%	95.1%
	95%				85.7%	95%	90.2%
	90%				72.9%	90%	81%

Table 5.2: Running both optimization approaches for the graph of Figure 5.4.

As one can conclude from the table above, using the first approach there will be no differences: even with two redundant edges between B and E (or between any pair of nodes), this approach is interested, in first place, in fulfilling the rule of the flow conservation. Consequently it fills an edge until it can (following the idea of the augmenting path - Definition 6); since it needs only one edge between B and E to transfer the flow, taking one of the edges of Figure 5.1 out will not influence the results. However, the second optimization approach does not work that way: the removal of one of the redundant

edges will modify the flow distribution and therefore the global POS. The following graph shows how the redistribution of flow is made:

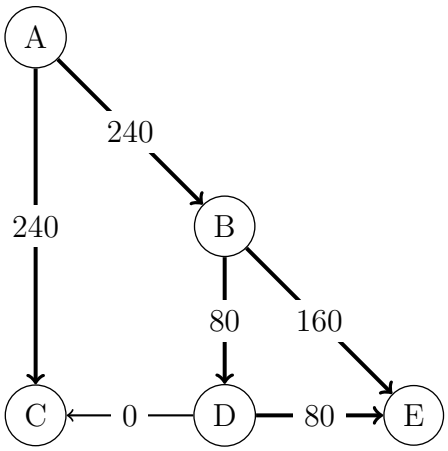

Figure 5.5: Optimization II results for Table 5.2.

In the previous example the edges connecting nodes B and E would each transfer only 96 units of flow, whereas in this case, edge (B, E) has 160 through it. The calculation is quite easy to perform and it proves why the POS decreases (even without taking into account the increase of flow in edges (B, D) and (D, E)):

$$e^{\log p(96)} = e^{-0.00936} = 0.996$$
$$e^{\log p(160)} = e^{-0.02610} = 0.989$$
$$e^{2\times\log p(96)} = e^{-0.01871} = 0.992 > e^{\log p(160)}.$$

Removing another edge

It is not possible to continuously remove edges from the network without affecting the desired sink consumption. However, it is still possible to remove one more edge - (D, E) - without disrespecting the rule of flow conservation, obtaining the following graph:

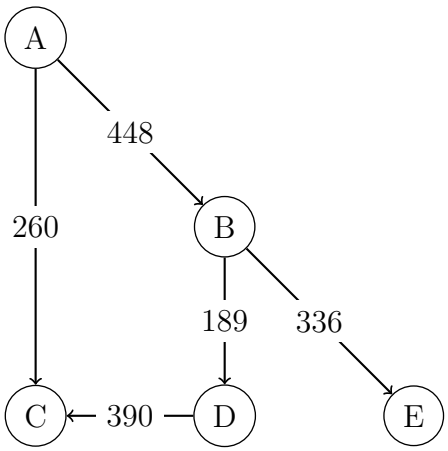

Figure 5.6: Graph from Figure 5.4 without edge (D, E).

The removal of the edge does not affect again the POS obtained by the first optimization approach. However, there is a slight decrease in the obtained POS when running the problem with the second approach, whose flow distribution will be the same as the one by Optimization I. The following table is a comparison between the three graphs, when running the Optimization II approach:

Approach	**Edge POS**	$\mathbf{k_i}$	**Total Edges**	***Used* Edges**	**Global POS**	**Node C POS**	**Node E POS**
Opt II	Equation 4.2.1	10^{-6}	5	3	83.6%	94.2%	88.8%
			6	5	85.4%	94.2%	90.6%
			7	6	86.7%	94.2%	92.0%

Table 5.3: Results for Optimization II.

Plots

It is possible to create a plot based on the graph's total number of edges and see the evolution of each POS. The same is made with the results obtained by the use of Optimization I approach. In this case the x-axis will be the variation of the edge POS, since it is the reason why the global, node C and node E POS changes:

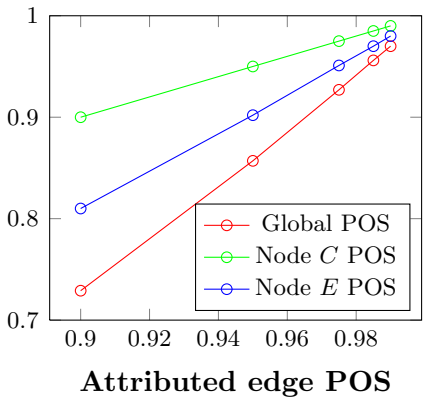

Figure 5.7: Results for Optimization I.

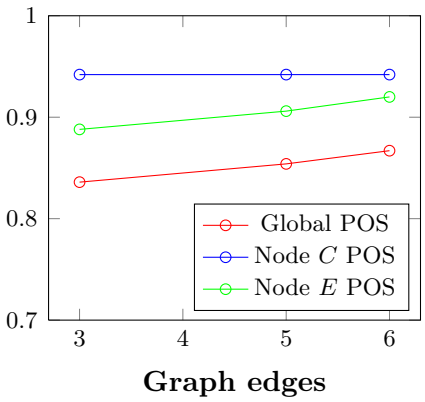

Figure 5.8: Results for Optimization II.

The analysis that one can make using the plots is the same as the tables. The creation of these graphics has the purpose of making easier to see the evolution of the several POS: when each edge POS is modified (in Optimization I) or when there is a removal of edges with a subsequent redistribution of flow through the existent ones (in Optimization II).

5.3 The Real Network

In the first place it is important to make an overview of the real network. As it was described in chapter 3, the real network can have power plants (source nodes), substations,

sinks, sectioning/cut posts and derivations. The following table presents the quantity of each structural element in the network:

Structural Element	**Quantity**
Power Plants	28
Substations	80
Sectioning/Cut Posts	11
Derivations	30
Sinks	65
Total Nodes	**214**
Total Edges	**349**

Table 5.4: Real Network Constitution.

As it was also said, many of this elements are virtual. For example, with the derivations there is an addition of 30 nodes that, in fact, do not exist physically. The actual network to transform into the matrix form to use with the `fmincon` procedure, has then a total of 214 nodes and 349 edges. It looks like the following picture:

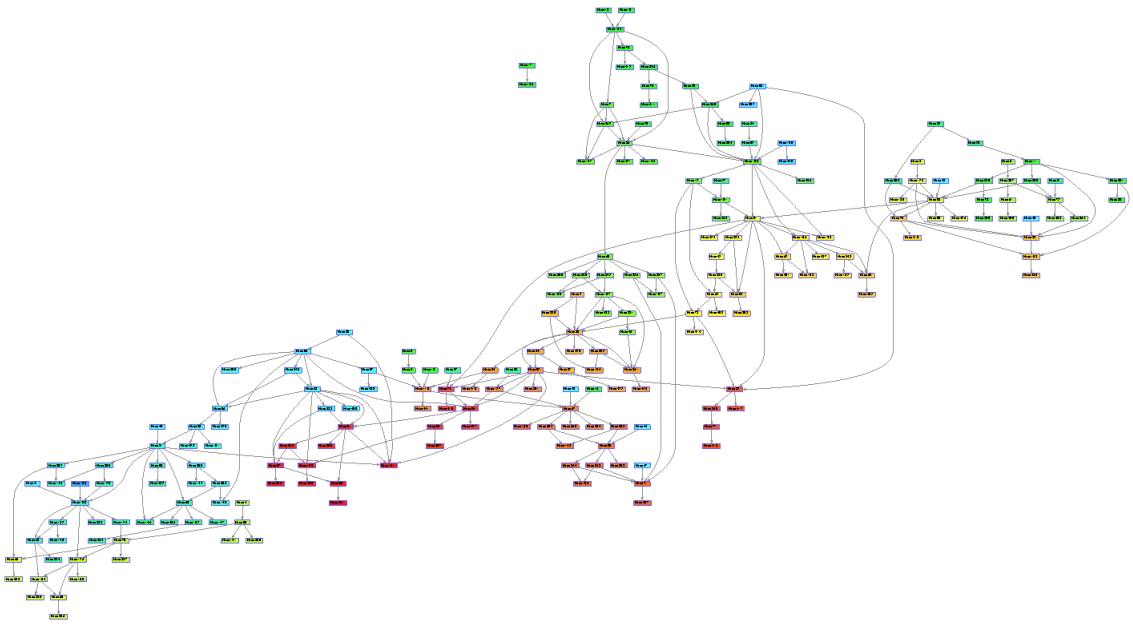

Figure 5.9: Overview of the graph that represents the real network.

By Table 5.4, one can denote a great difference between the number of power plants (sources) - 28 - and the number of sink nodes - 65. However, the substations play a very important role to balance the amount of production and consumption. As it happens with power plants, substations have also the capacity of producing electrical energy that is ultimately sent to the outer grid sinks.

Another important aspect is the fact that from the 80 substations, 61 of them have connections with the sinks from the outer grid. A substantial percentage of each substation production is directly transferred to the respective sink: what happens in the real network is that each substation may have more than one sink associated to it. For the problem, it is considered that each set of sinks is condensed into one. When making calculations, the value for each set of sinks is then divided by the number of sinks in the set to get the amount of electrical energy that each real sink gets. Also, 9 out of the 65 sinks are from the inner grid. It is with the results (flow distribution and POS) obtained from these 9 sinks that the analysis will be made.

On the other hand, each edge that connects the substations to the respective sink is virtual. This means that it does not have any possibility of failure and it is not considered to the set of *used* edges. If any problem occurs with the distribution network, it is considered to be exclusively related to the substation that feeds the sink nodes.

Making the first runs of the `fmincon` procedure, the following results are obtained:

Inner Grid Sink Nodes	**Edge POS**	$\mathbf{k_i}$	**Consumption Inner Sink**[2]	***Used* Edges**	**Global POS**
9	Equation 4.2.1	10^{-6}	100	38	89.3%

Table 5.5: Running Optimization II for the real the network.

The table below[3] describes with more detail the POS of each inner sink node and gives information of how many edges were *used* in the process of flow distribution:

Sink Node	***Used* Edges**	**MDN**	**Ratio**	**POS**
1	4	4	1	96.0%
2	6	1	6	99.8%
3	2	1	2	99.5%
4	4	2	2	99.0%
6	16	4	4	99.2%
7	13	4	3.25	98.5%
8	1	1	1	99.0%
9	2	2	1	98.0%

Table 5.6: POS of each inner sink node.

Table 5.6 does not take into account node 5 because it is the one that only consumes 44 units *versus* the 100 units of every other node. The following plot is a representation

[2]There is one of the inner sinks that only consumes 44 units.

[3]The **MDN** - Maximal Distant Node - column refers to the furthest node from the inner sink node that still produces energy that is consumed by it. **Ratio** is the quotient between the *used* edges and the MDN.

of the previous table:

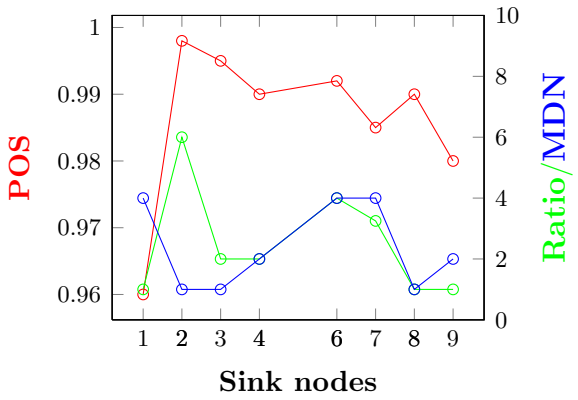

Figure 5.10: Correlation between each sink node's POS, the MDN and the computed ratio between MDN and *used* edges.

From Figure 5.10 it is possible to find a correlation between the POS of each sink node and the ratio between the MDN and the *used* edges.

- For the same MDN, the optimization approach returns a higher POS for the sink nodes that have a higher ratio: this is a proof that the formulation promotes the maximum division of flow that is possible - nodes 1, 6 and 7 with MDN equal to 4 or nodes 2, 3 and 8 with MDN equal to 1;
- For the same ratio, the approach returns a higher POS for the sink nodes that have the smallest MDN: the flow distribution is the same between each pair of nodes; however, the POS decreases node by node, which implies that with more nodes to travel, there is a slightly larger chance to occur a failure between the first and last nodes.
- In general, with a higher MDN one can expect a lower POS; at the same time, a wider distribution of flow (more redundancy) promotes a higher POS, because the POS of and edge with less flow coursing through it is higher.

The flow distribution of each node can be seen in detail in Appendix C. In the same appendix, one can see the same flow distribution for different consumption values.

5.3.1 Removing Edges

As it was said in subsection 5.2.2, it is possible to remove edges from the network until it no more respects the rule of flow conservation. For example, node 8 from Table 5.6 is fed by only one edge: if this edge is removed, the sink won't get any flow and the rule of flow conservation will be broken - see Figure C.7. On the other hand, node 2 has an MDN value of 1 and is fed by 6 edges at the same time - see Figure C.2.

Consequently, it can endure the removal of, at most, 5 edges. The next table and plot present the evolution of the POS of node 2, taking into account the number of *used* edges:

Used **Edges**	**POS**	**Units per Edge**
6	99.82%	16.(6)
5	99.79%	20
4	99.73%	25
3	99.66%	33.(3)
2	99.49%	50
1	98.99%	100

Table 5.7: Node 2 - POS with different number of *used* edges.

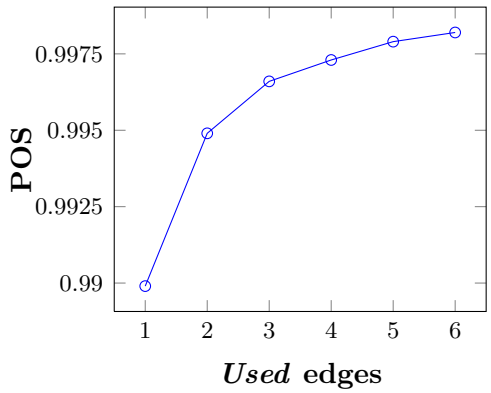

Figure 5.11: Node 2 - POS/Edge Removal.

Whenever there is an edge removal, the redistribution is always made in order to divide the flow through every redundant edge. It is noticeable from Table 5.7 and Figure 5.11 that the POS increases with the number of *used* edges, which is also according to the function of probability, itself. However, this is all representing a very linear idea of what can really happen in reality. Even if a pair of nodes has redundant edges[4], they may not present the same values for the constants k_i of Equation 4.2.1, which will alter the values from Table 5.7. Nonetheless, this approach presents the theoretical idea of how the flow should be distributed when redundant edges appear between nodes.

The difference between redundant edges of the same pair of nodes does not resume by the different functions of probability. In terms of maintenance, the length of each edge is one of the most important characteristics. For a fixed cost of maintenance per kilometre of line, it is easier to perform any kind of inspection and maintenance in a shorter line. However, the determination of an optimal point of maintenance involves many factors and they go beyond the theoretical ideas of the optimization formulation for calculating the best flow distribution: according to [2], one should take into account the value of customers on each injection point or the cost of energy in each energy source (which is considered to be constant for every source in this formulation), among others. Using the example of node 2, it may be cheaper to use 3 lines instead of 5 or 6 even if this decision decreases the final POS.

The type of decision that involves a trade-off between the probabilistic and economical results must obviously take into account the results obtained by the Optimization II approach. Nonetheless, to perform a complete and accurate decision of which set of lines

[4]In the case of node 2, every edge that "arrives" in 2 comes from a different node.

should be used in a certain moment (or those that need a more emergent maintenance), one must take into account the above mentioned factors:

- *Cost of energy at each energy source:* a formulation with coefficients associated to each node representing their *weight* (importance) in the network;
- *Sink node importance:* like the previous item, the association of *weights* to each sink node would present a more accurate approach of what happens in reality;
- *Edge cost:* taking into account the cost associated to the traversal of flow through each edge will make any formulation more realistic. Financially speaking, it can be much better to perform the distribution of flow through only one or two edges instead of four or five, even if that means a decrease of the node's POS.

5.3.2 Evolution of POS - Constants k_i

Two of the constants that are considered in the function of probability of each electrical line are its age - k_0 - and the influence of near vegetation - k_2. It is trivial to understand that the values associated to each one of the constants will vary with time. Consequently, for a constant value of flow intensity in an edge, its POS will decrease with time.

On the other hand, the function of probability from Equation 4.2.1 is dependent on the intensity on the edge. However, one can fix the intensity value and function dependent on the constants k_i. If every k_i decreases at the same rate, assuming that each one starts at 10^{-6}, the obtained function will be linear on k_i. For example, let $f(i,j)_k = 100$ and k_i assume the same value for every i; the function will be:

$$p(k_i) = 1 - (k_i + 100 \times k_i + 10000 \times k_i) = 1 - 10101 \times k_i, \tag{5.3.1}$$

which has its zero at 9.9×10^{-5} and is represented in the following plot:

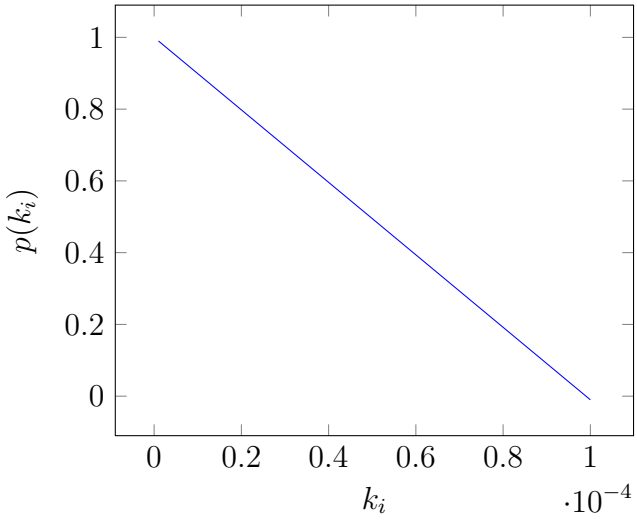

Figure 5.12: Equation 5.3.1.

The previous plot is the proof that the increase in the constants k_i, when made with the same rate, creates a linear decrease of the POS. The fact is that this is only an example and although there may be a linear evolution of the constant k_0 with time, since it represents the age of a line, that does not happen with k_1 (or k_2).

Even if k_0 is the constant that increases more rapidly with time, the evolution of the POS regarding that increase will not be as abrupt as with an increase of k_1 and k_2. Assuming again $f(i,j)_k = 100$, $k_1 = k_2 = 10^{-6}$ and that k_0 increases from 10^{-6} until 10^{-4} in five different time moments:

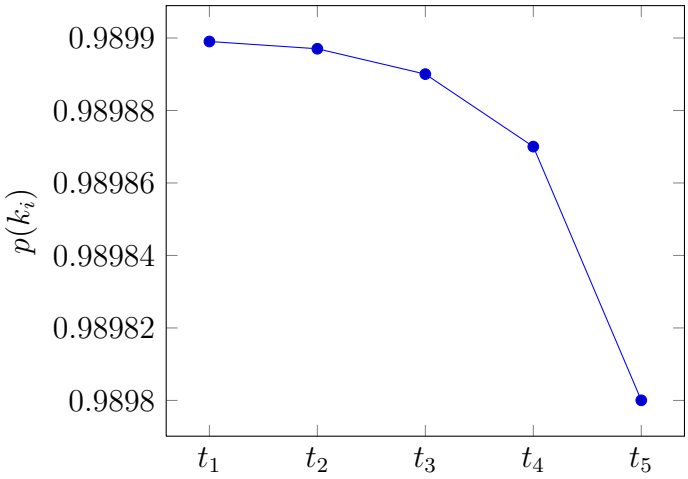

Figure 5.13: Evolution of POS when increasing k_0.

One can see from the previous plot that with an increase of two orders, the value for the POS would only decrease one tenth of a thousand. On the other hand, k_1 is a constant that changes within different periods of the same day: during daytime it will have a higher value since it is when there is the use of electrical energy; during the night when people are asleep it will be associated with a lower value - the used order of 10^{-6}.

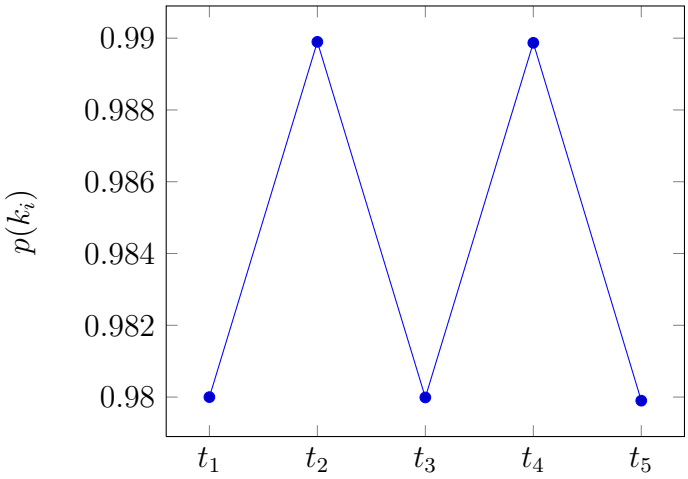

Figure 5.14: Evolution of POS when changing k_1 from day to night-time.

From the plot above it is possible to notice the differences of the POS when a certain

line is being used during day or night-time: the higher POS values assume night-time, whereas the lower values assume daytime. It is possible to create a plot that shows a more detailed evolution with real data. The values are used only to show the evolution of POS in three different days, and also with an evolution of parameter k_0. One should not assume that the three days are consecutive, as they use the evolution of Figure 5.13.

Even k_2 may not have a linear evolution, since this constant is related with the vegetation growth: this growth may not be proportional to the line's ageing, which creates a different evolution of the respective parameter. However, the evolution of this parameter is much rougher than k_0's. Using the same five distinct moments of time, and assuming that $k_0 = k_1 = 10^{-6}$, one will have:

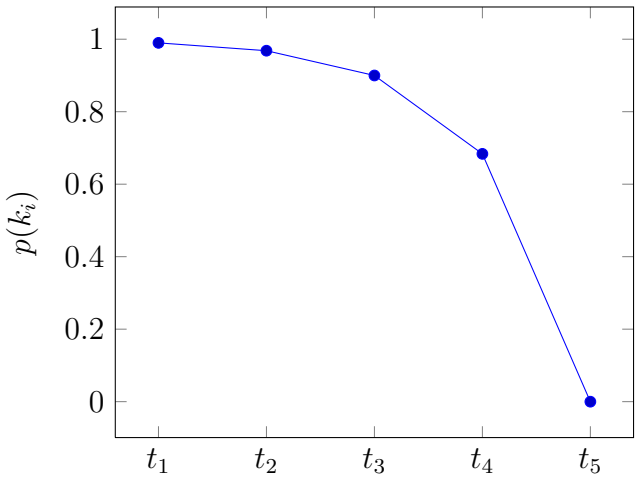

Figure 5.15: Evolution of POS when increasing k_2.

If one compares the plots of Figure 5.13 and Figure 5.15 without looking to the scale, it would probably say that was the same. However, while the increase of two orders in k_0 decreases the POS by 0.0001, the same increase in k_2 will make the associated electrical line completely unavailable[5].

Not having the real evolution of the parameters, the analysis can only be made with the theoretical evolution of the values. However, it is possible to understand how each of the parameters influence the evolution of the POS from the plots above.

[5]Actually, if $k_2 = 10^{-4}$, the POS will assume an "infeasible" negative value.

Chapter 6

Conclusions and Future Work

In this dissertation three approaches to compute the POS of several elements of a network were studied and presented. The last approach is the most stable of the two optimization approaches described: it easily computes global and sink nodes POS using an accurate function of probability; it accepts the removal of edges to recompute probabilities and redistribute the flow through the remaining edges; it can also analyse the evolution of each element's POS throughout the time.

The development of an accurate analysis for the evolution of probability throughout the time is approached in subsection 5.3.2. Although the results presented are only based in theoretical terms, the analysis made can be easily completed with real data sets regarding the evolution of the constants k_i existent in the real function of probability. For example, the quadratic element associated to k_2 is the main factor to the decrease of the probability with the time and that is noticeable from the plot in Figure 5.15.

With data sets regarding the real evolution of the parameters, it is possible to ascertain which edge is the most problematic in short period of times or which edge should be inspected/maintained if it reaches a certain POS. At the same time, it is possible to redistribute the network flow by constantly run the `fmincon` procedure when needed: from daytime to night-time if there is a sudden increase of k_1; from one season to another, when vegetation grows and k_2 is subject to a problematic increase.

On the other hand, the algorithmic approach presented in subsection 4.1.1 is also an alternative to the Optimization II approach. Since the best results were obtained in an optimization fashion of the problem, this algorithmic approach was delegated to second place. However, if one used the idea of searching for augmenting paths by increasing the flow one by one unit (assuming a topology where only integer flows were allowed), the results obtained could be proximal to the optimization approaches. Although this could be an improvement to the approach described in terms of result optimality, one should take into account that the real function of probability varies with the flow that courses through an edge. Moreover, the increase of flow only by one unit at a time will lead to a

complexity equivalent to the worst-case of Ford-Fulkerson method - $O(E|f_{\max}|)$ - which is far from interesting in computational terms.

6.1 Future Work

Although the Optimization II approach presents the desired stability for the calculation of the POS and consequent flow distribution, there is still room for evolution regarding several aspects that could not be addressed during the time frame of this work. Some of them were set as goals of this work but there was no chance to analyse them:

- *Real parameter evolution:* the function of probability that computes each electrical line POS is subject to variations - from edge to edge there is differences in the constants k_i;
- *Constants k_i evolution:* the evolution of k_i throughout the time must be addressed with attention; as it was aforementioned, the analysis made depends on theoretical values but it can easily extended when facing real data sets;
- *Cost of energy production and consumption:* each source node has a cost of production and each sink node a cost of consumption, which is not contemplated in the Optimization II approach; this can be addressed by simply associating to each node a coefficient representing their relative importance in the network;
- *Budgetary items:* given the POS calculated for each edge and taking into account the evolution of parameters when making inspections and maintenance, it is possible to choose which edges are cheaper to maintain; on the other hand, it is possible to compute the minimum budget to spend and the location of inspections by taking into account each line's length.

These items can be addressed mainly by analysing the evolution of each edge POS throughout the various periods of time. Moreover, while the approaches studied defend a selection of edges that promotes a higher global (and for each sink) POS, in reality it can be cheaper to use a lesser number of edges. This decision can only be made if there is an addition of parameters that represent the real cost of using a certain edge.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, Upper Saddle River, New Jersey, United States of America, 1993.

[2] F. Azevedo and J. Gomes-Mota. *A Time and Space Framework for Overhead Grid Maintenance Optimisation. Modern Energy Review*, 2(1):95–99, 2010.

[3] F. Azevedo and J. Gomes-Mota. *Electrical grid modelling for overhead maintenance cycle optimisation.* In T. Hadzic and R. Marinescu, editors, *Proceedings of the 1st Workshop on Constraint Reasoning and Graphical Structures - CRaGS 2010*, pages 75–82, 2010.

[4] T. H. Cormen, C. E. Leiserson, Rivest, and C. Ronald, Stein. *Introduction to Algorithms*. MIT Press, Massachusetts, United States, 3rd edition, 2009.

[5] E. A. Dinic. *Algorithm for solution of a problem of a maximum flow on a network with power estimation.* In *Soviet Math. Doklady*, volume 11, pages 1277–1280. 1970.

[6] J. Edmonds and R. M. Karp. *Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. Journal of the ACM*, 19(2):248–264, April 1972.

[7] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[8] A. V. Goldberg, E. Tardos, and R. E. Tarjan. *Network Flow Algorithms.* In B. Korte, L. Lovasz, H. J. Proemel, and A. Schrijver, editors, *Algorithms and Complexity*, volume 9 - Paths, Flows and VSLI-Layout, pages 101–164. Springer Verlag, 1990.

[9] A. V. Goldberg and R. E. Tarjan. *A new approach to the maximum flow problem.* In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 136–146, New York, NY, USA, 1986. ACM.

[10] A. V. Goldberg and R. E. Tarjan. *Finding minimum-cost circulations by cancelling negative cycles. Journal of the ACM*, 36(4):873–886, 1989.

[11] A. V. Goldberg and R. E. Tarjan. *Finding Minimum-cost Circulations by Successive Approximation. Mathematics of Operations Research*, 6(3):430–466, 1990.

[12] J. Gomes-Mota. *Integrated, Flexible and Real Time Inspection of Overhead Lines. Power Industry International*, 2(1), 2008.

[13] J. Gomes-Mota. *Unified status evaluation for reliability assessment of overhead power grids.* Bucharest, Hungary, September 2011. International Conference on Condition Monitoring, Diagnosis and Maintenance - CMDM 2011.

[14] J. Gomes-Mota, M. Matos, and A. Matos-André. *Geographical Information Tools For Overhead Lines Preventive Maintenance.* In *Proceedings of Session 42 of International Council on Large Electric Systems - CIGRÉ08*, Paris, France, August 2008.

[15] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer-Verlag, New York, United States of America, 2nd edition, 2006.

[16] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications, Mineola, New York, United States of America, 1998.

[17] H. Röck. *Scaling techniques for minimal cost network flows.* In U. Pape, editor, *DiscretesStructures and algorithms*, pages 181–191. Carl Hansen, 1980.

Appendix A

Algorithmic Approach Example

Having the network of Figure 4.1 as basis, where source node A distributes 480 units of flow, the following POS are assumed for the edges:

- $p_{AC} = 0.1$
- $p_{AB} = p_{DC} = p_{DE} = 0.995$
- $p_{BD} = p_{CT} = p_{ET} = 1$
- $p_{BE} = 0.8$

Consequently, the symmetric of the probabilities logarithms are:

- $-\log(p_{AC}) = 1$
- $-\log(p_{AB}) = -\log(p_{DC}) = -\log(p_{DE}) = 0.0022$
- $-\log(p_{BD}) = -\log(p_{CT}) = -\log(p_{ET}) = 0$
- $-\log(p_{BE}) = 0.0969$

Starting the algorithm, one can compute the augmenting paths of the network. Applying Equation 4.1.8, the respective costs are also calculated:

- $A \to C \to T \Rightarrow 1$
- $A \to B \to E \to T \Rightarrow 0.0991$
- $A \to B \to D \to C \to T \Rightarrow 0.0044$
- $A \to B \to D \to E \to T \Rightarrow 0.0044$

By lines 5 and 6 of the algorithm, it is possible to choose between the third or fourth path presented above (it depends on the implementation of the sorting algorithm). Choosing the path $A \to B \to D \to C \to T$, the residual network G'_T obtained from line 7 is:

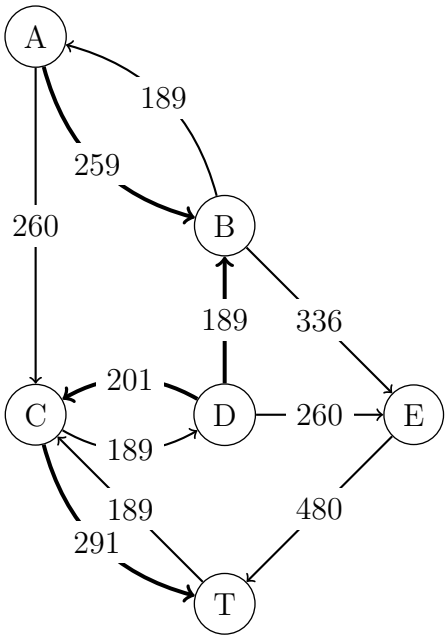

Figure A.1: Residual network G'_T after first flow update - $|f_1| = 189$.

In the figure, the ***bolded*** edges are the traversed ones. The edge (D, B) is traversed in (B, D) direction. It is represented this way because of the definition of residual network. The same happens in Figure A.2 and Figure A.5. After the first iteration the new possible augmenting paths in the new G_T are (without possible loops):

- $A \to C \to T \Rightarrow 1$
- $A \to B \to E \to T \Rightarrow 0.0991$
- $A \to C \to D \to E \to T \Rightarrow 1.0044$
- $A \to C \to D \to B \to E \to T \Rightarrow 1.1013$

Again, by lines 5 and 6 of the algorithm, p turns out to be the second path of the list - $A \to B \to E \to T$. The update with the path flow returns the residual network of Figure A.2.

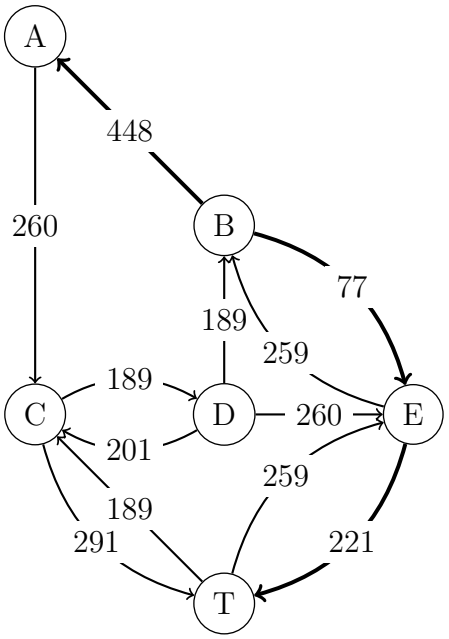

Figure A.2: Residual network G'_T after second flow update - $|f_2| = 259$.

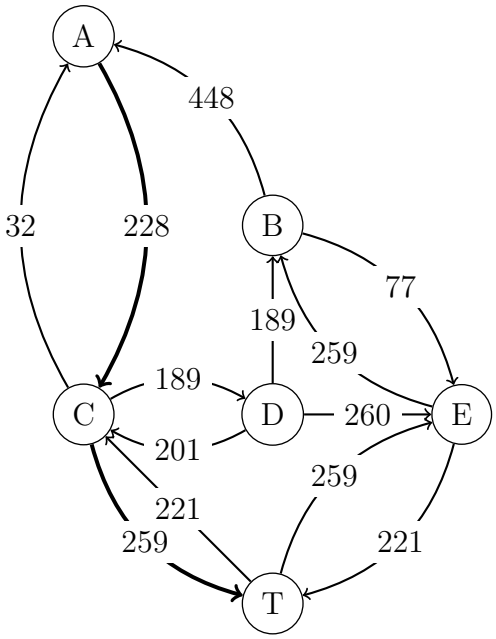

Figure A.3: Residual network G'_T after last iteration - $|f_{max}| = 480$

After the second iteration, the residual network presents only two augmenting paths and the chosen path will be the first one - $A \to C \to T$:

- $A \to C \to T \Rightarrow 1$
- $A \to C \to D \to E \to T \Rightarrow 1.0044$

The update of the flow will result into the network of Figure A.3. After the computation of the maximum-flow it is possible to remove from the original network every unused edge - line 11 of the algorithm. The only unused edge was (D, E) and the resulting network G''_T is the one in Figure A.4.

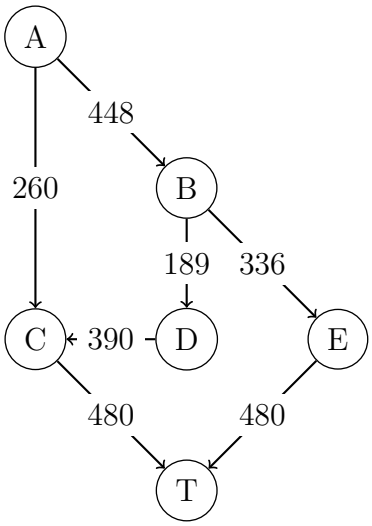

Figure A.4: Updated network G''_T without edge (D, E).

If one computes the sum of the costs of each used edge will get value 1.1013. However, that is not the best sum of negative logarithms for the same maximum-flow and that is the reason to apply the Edmonds-Karp algorithm.

That application is then made onto the network of Figure A.4 and after two iterations, the maximum-flow and the optimal POS are obtained. One can verify the iterations of Edmonds-Karp algorithm in Figure A.5 and Figure A.6.

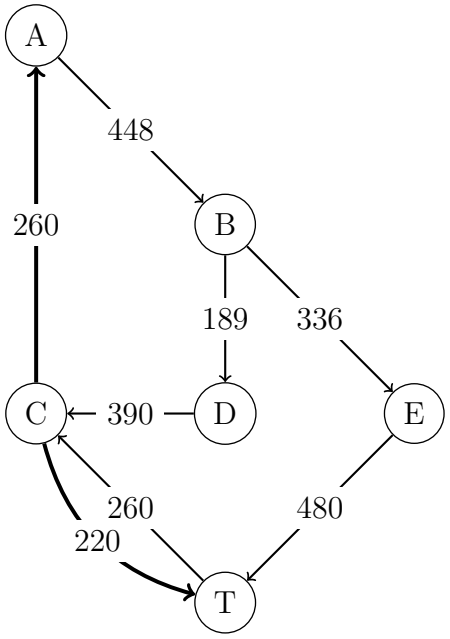

Figure A.5: First iteration of Edmonds-Karp algorithm over G''_T - $|f_1| = 260$

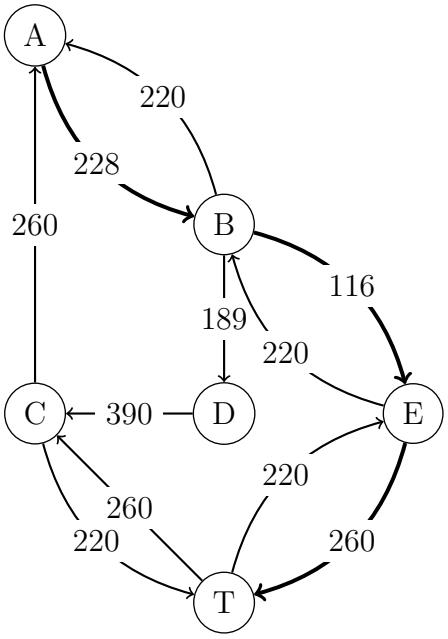

Figure A.6: Second iteration of Edmonds-Karp algorithm over G''_T - $|f_{max}| = 480$

For this example, the maximum-flow is $|f_{max}| = 480$ and the POS is $p_{global} = 10^{1.0991} = 12,56\%$. The exponent for the calculation of p_{global} results of the sum of the negative logarithms of the POS of each used edge. The paths chosen were:

- $A \rightarrow C \rightarrow T \Rightarrow 1$
- $A \rightarrow B \rightarrow E \rightarrow T \Rightarrow 0.0991$

whose sum performs the exponent 1.0991.

Appendix B

From C to MATLAB®

The responsible function for passing the parsed data from C to MATLAB® is below detailed:

```
int setupMatlab(Circuit **circuits, int *sourceNodes, int sources,
int *sinkNodes, int sinks, int edges, int *existentNode, FILE *spoint) {

   Engine *ep;
   mxArray *obj = NULL, *lb = NULL, *ub = NULL, *Aineq = NULL,
*bineq = NULL, *x0 = NULL;
   mxArray *x = NULL, *fval = NULL, *exitflag = NULL;

   int i, j = 0, size = edges+NODES_VAR;
   char buffer[BUFSIZE+1], xTemp[10], equal[5];

   double objective[size], lowerBound[size], upperBound[size],
matrix[NODES_VAR*size], values[NODES_VAR], startingPoint[size];

   if (!(ep = engOpen(""))) {
      fprintf(stderr, "Can't start MATLAB engine\n");
      exit(1);
   }

   for (i = 0; i < NODES_VAR; i++) {
      objective[i] = 0;
      if (binarySearch(sourceNodes, sources, i+1) || !existentNode[i])
         lowerBound[i] = 0;
      else lowerBound[i] = -0.5;
      upperBound[i] = 0;
```

```
      if (i+1 == sinkNodes[j]) {
         objective[i] = -1;
         j++;
      }
   }

   for (; i < size; i++) {
      objective[i] = 0;
      lowerBound[i] = 0;
      upperBound[i] = circuits[i-NODES_VAR]->winterMax;
   }

   for (i = 0; i < NODES_VAR*size; i++) {
      matrix[i] = 0;
   }

   for (i = 0; i < edges; i++) {
      matrix[getIndex(i+NODES_VAR, circuits[i]->startNode)] = 1;
      matrix[getIndex(i+NODES_VAR, circuits[i]->endNode)] = -1;
   }

   for (i = 0; i < NODES_VAR; i++) {
      if (binarySearch(sourceNodes, sources, i+1))
         values[i] = 100;
      else if (binarySearch(sinkNodes, sinks, i+1))
         values[i] = -100;
      else values[i] = 0;
   }

   x0 = mxCreateDoubleMatrix(size, 1, mxREAL);
   memcpy((void *)mxGetPr(x0), (void *)startingPoint,
sizeof(startingPoint));

   obj = mxCreateDoubleMatrix(1, size, mxREAL);
   memcpy((void *)mxGetPr(obj), (void *)objective, sizeof(objective));

   lb = mxCreateDoubleMatrix(size, 1, mxREAL);
   memcpy((void *)mxGetPr(lb), (void *)lowerBound, sizeof(lowerBound));
```

```
ub = mxCreateDoubleMatrix(size, 1, mxREAL);
memcpy((void *)mxGetPr(ub), (void *)upperBound, sizeof(upperBound));

Aineq = mxCreateDoubleMatrix(NODES_VAR, size, mxREAL);
memcpy((void *)mxGetPr(Aineq), (void *)matrix, sizeof(matrix));

bineq = mxCreateDoubleMatrix(NODES_VAR, 1, mxREAL);
memcpy((void *)mxGetPr(bineq), (void *)values, sizeof(values));

engPutVariable(ep, "x0", x0);
engPutVariable(ep, "obj", obj);
engPutVariable(ep, "lb", lb);
engPutVariable(ep, "ub", ub);
engPutVariable(ep, "Aineq", Aineq);
engPutVariable(ep, "bineq", bineq);

engEvalString(ep, "save('problem.mat');");

mxDestroyArray(x0);
mxDestroyArray(obj);
mxDestroyArray(lb);
mxDestroyArray(ub);
mxDestroyArray(Aineq);
mxDestroyArray(bineq);

engClose(ep);

return 0;
```

The listing of this function serves the purpose of showing how the functions of MATLAB® work in the C environment. It is assumed that the `.c` file has all the required headers and functions used - `binarySearch()` and `getIndex()`.

Appendix C

Inner Sink Nodes Flow Distribution

C.1 Initial Conditions

Sink nodes are yellow and will always be identified with the number that was used in Table 5.6. Every other node in the flow distribution will be named with a letter.

Node 1:

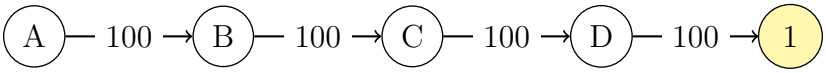

Figure C.1: Node 1 flow distribution - real network with 100 units of flow.

Node 2:

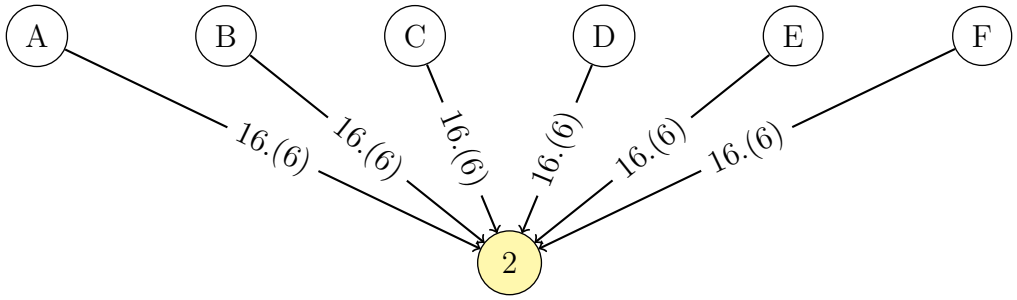

Figure C.2: Node 2 flow distribution - real network with 100 units of flow.

Node 3:

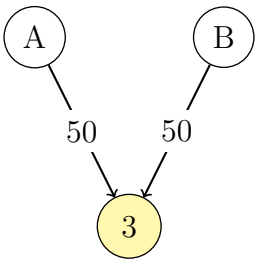

Figure C.3: Node 3 flow distribution - real network with 100 units of flow.

Node 4:

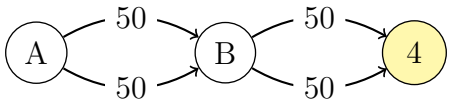

Figure C.4: Node 4 flow distribution - real network with 100 units of flow.

Node 5:

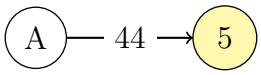

Figure C.5: Node 5 flow distribution - real network with 44 units of flow.

Nodes 6 & 7:

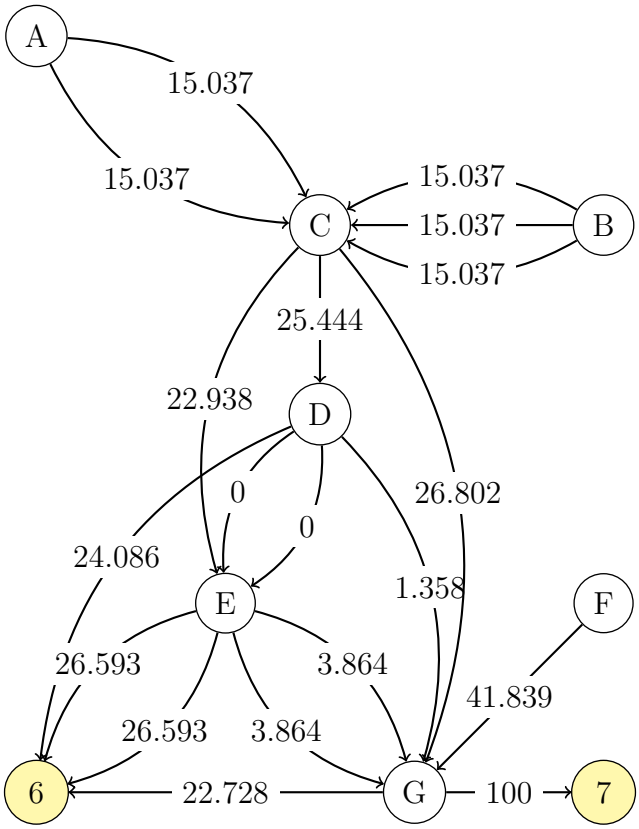

Figure C.6: Nodes 6 and 7 flow distribution - real network with 100 units of flow.

Node 8:

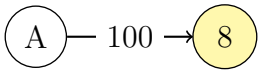

Figure C.7: Node 8 flow distribution - real network with 100 units of flow.

Node 9:

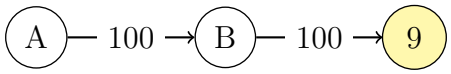

Figure C.8: Node 9 flow distribution - real network with 100 units of flow.

C.2 Modifying the Consumption Requirements

In this section, the consumption for each node is raised to 200 units. However, some of the nodes are unable to get that amount of flow because of the restrictions. The following graphs represent the flow distribution for the nodes that can actually consume the 200 units of flow. The table in the end of the section is similar to Table 5.6.

Node 2:

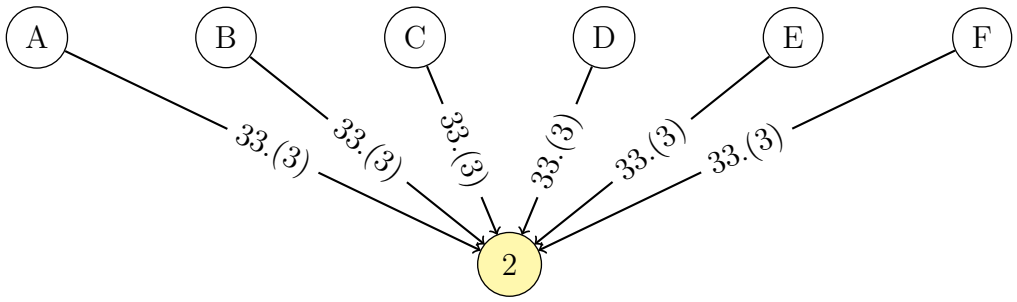

Figure C.9: Node 2 flow distribution - real network with 200 units of flow.

Node 3:

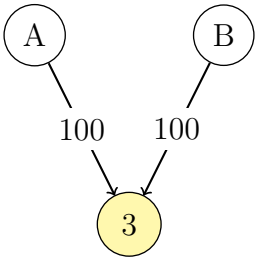

Figure C.10: Node 3 flow distribution - real network with 200 units of flow.

Node 4:

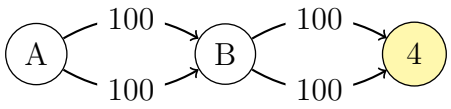

Figure C.11: Node 4 flow distribution - real network with 200 units of flow.

Nodes 6 & 7:

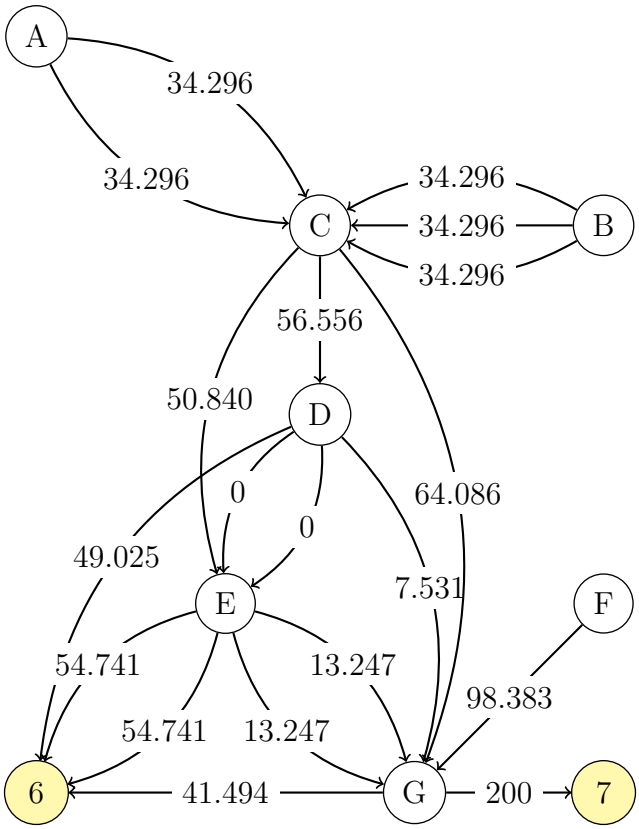

Figure C.12: Nodes 6 and 7 flow distribution - real network with 200 units of flow.

Node 8:

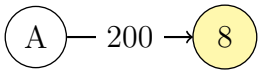

Figure C.13: Node 8 flow distribution - real network with 200 units of flow.

Node 9:

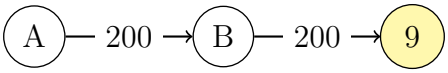

Figure C.14: Node 9 flow distribution - real network with 200 units of flow.

Sink Node	***Used* Edges**	**MDN**	**Ratio**	**POS**
2	6	1	6	99.3%
3	2	1	2	98.0%
4	4	2	2	96.0%
6	16	4	4	96.4%
7	13	4	3.25	93.5%
8	1	1	1	96.0%
9	2	2	1	92.1%

Table C.1: POS of each inner sink node with consumption of 200 units.