Master Thesis

DOUBLE MASTER'S DEGREE IN INDUSTRIAL ENGINEERING AND MANAGEMENT ENGINEERING

An optimized routing algorithm for data collection in IoT network

REPORT

Author:	Marta Pàmies Morera
Director:	Prof. Kim Nguyen (ETS, Montréal)
Summons:	June 2017

Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

and

École de Technologie Supérieure
Montréal, Canada

Abstract

This thesis has as a principal objective the resolution of a problem present in the Internet of Things (IoT) network. This problem is the routing optimization of the data between the sensors and the gateway, which is not optimized these days in the current applications of the IoT. The project is attacked mainly from a theoretical point of view and so several hypotheses are done to carry it out, but it leads to a generalized solution valid for different IoT applications.

The network is pictured as a graph in which the sensors and the gateway are the nodes and the connections between them are the links. More specifically, the thesis uses an algorithm widely used in the transportation network, the Clarke & Wright's Savings Algorithm, and adapts it to the data network. This algorithm is based in the savings of putting two or more nodes in the same route comparing it to the case where all sensors send directly their data gathered to the gateway. The algorithm starts with the pair of nodes that imply the highest saving if both were at the same route, and with their constraints the code determines if this merge is feasible or not. The same process is done for all the pairs of nodes (picking them from maximum savings value to the minimum) until all nodes are connected to another one.

The solution of the algorithm is compared with three other solutions: two base lines and the exact solution extracted from the software CPLEX. The first base line depicts the solution in which every node sends directly to the gateway (no algorithm applied), and the second base line is similar to the actual algorithm proposed but with limiting to 2 nodes maximum connected before reaching the gateway.

After obtaining a total number of 500 solutions (from 2 to 100 nodes), the algorithm was 475 times better than the BL with an average percentage of improvement of 36,54% out of the 475, and 329 times better than the BL2 with the same percentage this time of 0,91% out of the 329. When comparing the algorithm with the CPLEX solution, a number of 52 cases were tested and this reduced number is due to its elevated execution time reaching the 100h of resolution for certain cases with less than 30 nodes. Out of these 52, only 13 of them the CPLEX solution was better than the algorithm and out of the 13 the average percentage of improvement is of 5,21%.

Table of contents

1. Vocabulary

IoT: Internet of Things

Sensor: electronic component which detects events in his environment and sends the information to other electronics.

Gateway: is the bridge between the sensors and the Cloud, where all data collected is stored and manipulated

Graph: mathematical structure formed by nodes and links which can represent a wide range of mathematical problems

BLE: Bluetooth Low Energy

WSN: Wireless Sensor Network

LAN: Local Area Network

WAN: Wide Area Network

PAN: Personal Area Network

DN: Data Network

TN: Transportation Network

VRP: Vehicle Routing Problem

Clarke & Wright's Savings Algorithm: one of the most known heuristics to solve the VRP in the transportation network

ICT: Information & Communications Technology

CPLEX: optimization software which solves linear programming problems using the simplex method

Parents: given a node i, their parents are the nodes who are after i in the chain to reach the gateway

Sons: given a node i, their sons are the nodes who are before i in the chain to reach the gateway

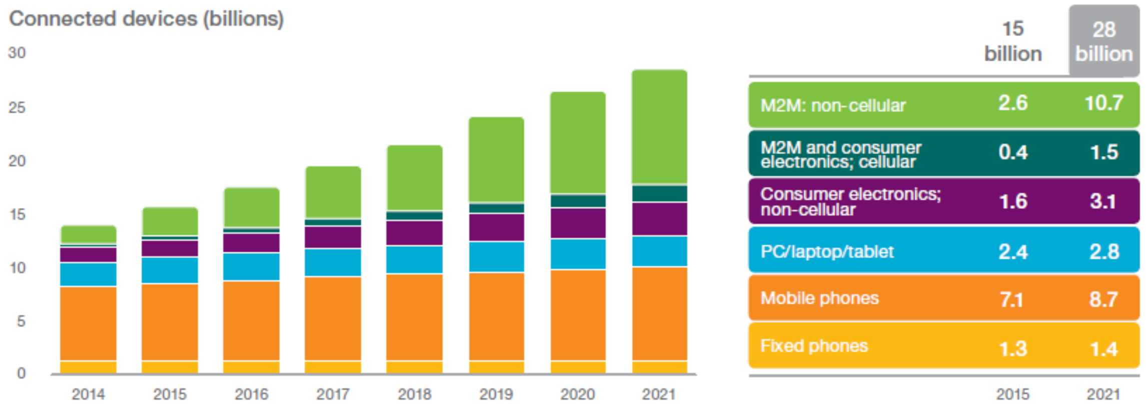
2. Introduction

2.1. Context and motivation

The frame of this project is the routing optimization in the data network in the Internet of Things technology. With no doubt, the optimization of processes and specially the routing optimization is what has been driving the author’s motivation for the past years. Also, the future which is oriented in this way proves the importance of the field.

Today, we find ourselves immersed in the technological era which does not stop influencing in our behavior in all aspects in life. In fact, we keep varying our actions in order to adapt to the changes coming each day and stay updated. The IoT has definitely arrived to stay and with it, a huge social change is coming. With the IoT, we will have better information, more control and insight into the everyday things and easier solutions to everything we need. For that, an enormous technological transformation is already happening and also some challenges come together with it.

Several experts have been predicting how may devices will be connected to the Internet the next years and the truth is that this number is very different one from other. But there is no doubt that it will be huge, and just to put an example, figures from Ericsson say that there will be 28 billion devices connected by 2021 (Figure 1). This is directly linked with the levels of energy consumption and the growing necessity to find ways to reduce them. For that, it is of great importance to study ways of routing optimization and that is what this thesis tries to solve.



	15 billion	28 billion
🟩 M2M: non-cellular	2.6	10.7
M2M and consumer electronics; cellular	0.4	1.5
Consumer electronics; non-cellular	1.6	3.1
PC/laptop/tablet	2.4	2.8
Mobile phones	7.1	8.7
Fixed phones	1.3	1.4
	2015	2021

Figure 1 Connected devices in 2021 according to Ericsson

Source: Ericsson Mobility Report (November 2015)

The challenge to carry out this project was enormous for the author, as her experience was not focused on the telecommunications field. But still, there were some aspects that working in a high-level stage, could be assimilated and permitted to work similar. The author's motivation is based on being able to bring her experience and knowledge of a different field, but applying it to the data network, also with an elevated gap for improvement.

2.2. Problem statement

The problem this project pretends to solve is how to optimize the path that data follows between the sensors and the gateway in the Internet of Things (IoT) network. We consider a network model in which there is a single gateway and a group of sensors that collect the data and send it to the gateway, as shown in Figure 2. Each sensor has some characteristics like: the amount of data it gathers, the maximal capacity of data it can gather, the lifetime of its battery, costs of sending the data, and maximum distance it can be connected to another node. The principal objective is to determine the total minimum cost by letting nodes sending their data to other nodes, as long as it is cheaper, instead of directly going to the gateway (Eq. 1). Such a network can be modelled as a graph in which the sensors and the gateway are the nodes and the connections between them are the links.

The mathematical problem statement is the following:

"Given a set of nodes-N (sensors), with certain characteristics-K (ex: data they gather), find the optimal path between the gateway (Node 0) and the rest of the nodes minimizing the total cost so that all data arrives to Node 0 regarding the nodes' constraints, C_{ik} . Distances between nodes are given, D_{ij}. It is supposed that every sensor sends the same type of data and this takes 1 unit of time (ex. 1 second) to send it."

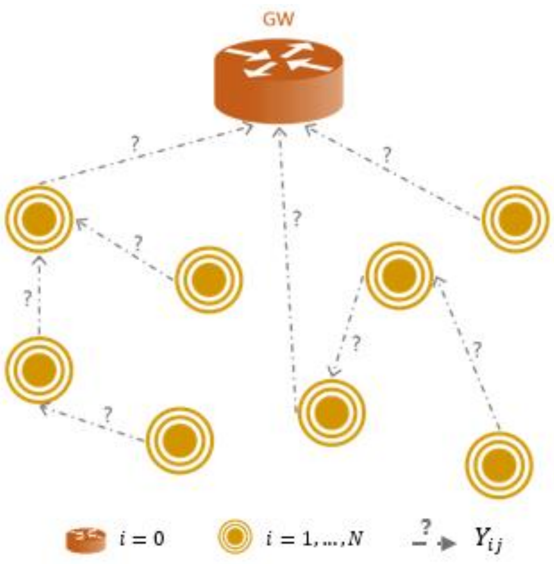


Figure 2 Example scheme of the problem

Source: Own

Data

$\boldsymbol{N}$: Set of nodes

$\boldsymbol{K}$: Number of nodes' characteristics

$\boldsymbol{C_{ik}} \in \mathbb{R}$: Nodes' characteristics $(i = 1, \dots, N ; k = 0, \dots, K)$:

- C_{i0}: Data node i gathers (Ex: $\frac{1}{5}$ msg/sec)
- C_{i1}: Capacity: data node i could reach to gather (Ex: $\frac{3}{5}$ msg/sec)
- C_{i2}: Remaining battery of node i (Ex: 5000 h = $18 \cdot 10^{6}$ sec)
- C_{i3}: Cost of sending 1 quantity of data for node i (Ex: $0{,}5 \cdot 10^{-8}$ \$/1 msg· 1 m)
- C_{i4}: Max. distance node i can be connected to another (Ex: 100 m)

$\boldsymbol{D_{ij}} \in \mathbb{R}$: Distance between nodes $(i, j = 0, \dots, N)$ [m]

Variables

$\boldsymbol{X_{ij}} \in \mathbb{R}$: Total data that node i will send to node j $(i, j = 0, \dots, N)$, (Ex: 1 msg/sec)

$\boldsymbol{Y_{ij}} \in \{\boldsymbol{0}, \boldsymbol{1}\}$: Takes value 1 if node i is sending data to node j $(i, j = 0, \dots, N)$

$\boldsymbol{Z_{ij}} \in \{\boldsymbol{0}, \boldsymbol{1}\}$: Takes value 1 if node i is linked to j through any other node. They will be indirectly linked but j will still carry i's data $(i, j = 0, \dots, N)$

Objective Function

$$[MIN] \left(\sum_{i=0}^{N} \sum_{j=0}^{N} C_{i3} \cdot X_{ij} \cdot D_{ij}\right) \quad \text{[Eq. 1]}$$

Constraints

$\boldsymbol{X_{ij}} \geq \boldsymbol{0}$; $\forall \boldsymbol{i}, \boldsymbol{j}$: Data being sent from node i to node j is either 0 or a positive value. [Eq. 2]

$\boldsymbol{Y_{ii}} = \boldsymbol{0}$; $\forall \boldsymbol{i}$: A node does not send data to itself. [Eq. 3]

$\boldsymbol{Y_{0j}} = \boldsymbol{0}$; $\forall \boldsymbol{j}$: Node 0 cannot send data to any other node j. [Eq. 4]

Le génie pour l'industrie ETSEIB

$\boldsymbol{D_{ij} \cdot Y_{ij} \leq C_{i4}}\ ;\ \forall \boldsymbol{i,j} \mid \boldsymbol{j} != \boldsymbol{0}$: If node i is sending data to node j, it will be at the most to the maximum feasible distance. This is not applied when j is node 0. **[Eq. 5]**

$\boldsymbol{\sum_{i=0}^{N} C_{i0} * Z_{ij} \leq C_{j1} - C_{j0}}\ ;\ \forall \boldsymbol{j}$: All data node j receives from other nodes i must not exceed the space j has left. **[Eq. 6]**

$\boldsymbol{\sum_{j=0}^{N} Y_{ij} = 1}\ ;\ \forall \boldsymbol{i} \mid \boldsymbol{i} != \boldsymbol{0}$: Every node i send to one other node j, except for node 0. **[Eq. 7]**

$\boldsymbol{X_{ij} \geq \left(C_{i0} + \sum_{l=1}^{N} C_{l0} * Z_{li}\right) * Y_{ij}};\ \forall \boldsymbol{i,j} \mid \boldsymbol{l} != \boldsymbol{i},\ \boldsymbol{l} != \boldsymbol{j}$: If i is sending its data to j, it will be at least the amount of: the data i has gathered + the data that node i has that is coming from other nodes l. **[Eq. 8]**

$\boldsymbol{Y_{ij} \leq Z_{ij}}\ ;\ \forall \boldsymbol{i,j}$: If Yij takes value 1, Zij must be 1 as well. But, if Yij takes value 0, Zij can still take values 0 or 1. **[Eq. 9]**

$\boldsymbol{Y_{ij} + Y_{ji} \leq 1};\ \forall \boldsymbol{i,j}$: If node i is sending its data to node j, j cannot send its data to i, or vice versa. **[Eq. 10]**

$\boldsymbol{Z_{li} + Y_{ij} \leq Z_{lj} + 1};\ \forall \boldsymbol{i,j,l} \mid \boldsymbol{i} != \boldsymbol{j}, \boldsymbol{i} != \boldsymbol{l}, \boldsymbol{j} != \boldsymbol{l}$: If both Zli and Yij take value 1, Zlj will be 1 as well. If none of both are 1 or just one takes value 1, Zlj can be either 0 or 1. **[Eq. 11]**

$\boldsymbol{C_{i3} \geq 1 + \sum_{l=0}^{N} Y_{li}};\ \forall \boldsymbol{i}$: Node i must have as much battery to receive all data coming from other nodes plus one extra second to send his data. **[Eq. 12]**

2.3. Research questions

To solve the present problem, some research questions are posed to drive its solution and understand the functioning of the Internet of things technology. These are:

- Which challenges is the IoT facing today and which ones are coming in a near future? Can the solution of the problem contribute to mitigate them?

  Despite all the benefits IoT is already bringing to the society, there are already issues that need to be considered in order to reduce their negative impacts on the environment. Part of this is that every time more and more devices are connected to the Internet and these carry enormous amounts of data which soon will saturate data centers and energy consumption will rocket to levels we cannot imagine nowadays. For that, it is of great importance that some routing optimization in IoT is made so to mitigate these adverse effects and this is what this thesis will try to accomplish.

- How was the problem addressed in the prior work? What are their drawbacks?

  From the article *L.A. Villas et al. (2013)* a solution to the problem is proposed comparing it with two other approaches that try to do the same. The thesis will analyze the method and compare it with the actual approach to find the pros and cons of both as well as evaluating if a merged solution of both could be reached.

- Can the savings algorithm used in the transportation network be adapted to build the solution to the present problem?

  The algorithm will try to adapt a concept which is widely used in the transportation network to optimize its routes and try finding the lowest total possible cost, which is described in the paper *Clarke & Wright's Savings Algorithm (1964)*, to solve a similar problem found in the data network between the sensors and the gateway of the IoT technology. Some changes in the concept will be made to fit the data network requirements.

2.4. Objectives

The objectives of this thesis can be summed up to 3 main ones:

- The design of an algorithm to solve the routing optimization problem in the IoT network between the sensors and the gateway, minimizing the total cost of sending data. To this end, a literature review will be carried out to understand the problem and the state-of-the-art solutions. Also, it is required to understand why these solutions are not ideal and try to create a solution which can be easily deployed to a wide range of use cases.

- Adopt the Clarke & Wright's Savings Algorithm, which has widely been used in the transportation network, to the actual problem in the data network. Some adaptation should be made to let the problem have sense and be feasible. For that, firstly the savings algorithm will be exposed and then the modifications will be detailed.

- The third objective of this thesis is the extraction of results and validation of the model built. For that, the solution of the algorithm will be compared with:

  - The exact solution extracted from the linear programming coded with the program CPLEX.

  - The base line (BL) solution which is when all nodes send their data directly to the gateway.
  - A second base line (BL2) solution which is similar to the actual algorithm, however it restricts the solution to not more than two nodes linked before reaching the gateway.

2.5. Hypotheses

It is worth to mention that the problem is attacked mainly from a theoretical point-of-view, and the proposed solution will be generalized for different types of IoT applications. Therefore, in this project many hypothesis and assumptions have been made, as follows:

- In order to simplify the model, the problem is treated in a high-level approach. It means the network is considered as a graph with nodes (the sensors) and links (the connections between the sensors) regardless other technical details like interference and link reliability.
- Similarly, the thesis takes into account only 5 characteristics of sensor nodes: data they gather, data they could gather in total regarding their capacity, battery they have left, cost of sending data and maximum distance they can be connected to other nodes. These metrics can represent sensor node in general. However, more specific metrics would also be considered in future.
- In this thesis, it is considered that all sensors gather the same type of data. Also, it is considered that data lasts 1 second to be sent from one node to another.

Other hypotheses have been made in the implementation part and are explained across Chapter 3.

2.6. Plan

The thesis will follow the next plan divided in five main sections:

- Background and motivation: in this first chapter, the background and experience of the author is exposed, as well as her motivation to fulfill a project, that a priori, was not her specialty, but she had the tools to achieve it in the end.

- Introduction to the IoT: a short introduction to the field has been done in order to understand this not-anymore-new technology together with a bit of its history and future prospects. Also, its architecture is explained.

- Literature review: after a problem is found, reviewing the existing literature is a crucial step to understand in which point is the problem, if there is someone that tried to solve it, if there is new research and discoveries about it. In this thesis, the author tries to find similar cases being solved and takes aspects of her Bachelor Thesis to make adaptations to fit the present problem and apply some of the things to reach the solution.

- Methodology: to achieve the objectives of the thesis, the author has followed a strict methodology which involves the adaptation of the Clarke & Wright algorithm used in the transportation network, programming the algorithm in Python and programming in CPLEX to compare results with the algorithm.

- Results and validation: this chapter will compare all solutions for a given data set. By all solutions, it means the exact solution found with the CLPEX, the solution of the two base lines defined (BL and BL2), and the solution given by the algorithm. This comparison will be made through graphs, tables and comments.

- Conclusions: this will be the final section where general conclusions of the project will be exposed, together with future steps that should be made to improve the algorithm with dynamic node composition or various types of data be sending at the same time through the set of nodes.

3. Chapter 1: Introduction to the IoT

3.1. History

The Internet of Things has been around for a while now, but where and when was it born? Since the early 1800s, machines have been communicating between them and the Internet has entered almost everyone’s lives year by year. It was not until 1999 that the Internet of Things was officially named in a presentation of Kevin Ashton, Executive Director of Auto-ID Labs at MIT, for Procter & Gamble. His speech transmitted the message that people were not capable of continuing capturing data about things in the real world because of their limited time, attention and accuracy. Instead, machines could be capable of knowing everything about things, using data they gathered without the help of humans and then, humans could track it, analyze it and be a step forward in replacing, repairing or recalling things.

One of the first real examples of the IoT was a fridge owned by Coca Cola, which was established at the Carnegie Melon University at the early 1980s. The programmers would connect to the machine through the Internet and were able to check if there was lack of any type of beverage.

Since then, the Internet of Things has been evolving into a full system manipulating multiple technologies ranging from the Internet to wireless communication and from micro-electromechanical systems (MEMS) to embedded systems. Components as wireless sensor networks, GPS and control systems all support the IoT.

3.2. The IoT today

New technologies evolve from the day they appear and the IoT is no different. The evolution of the IoT has been from a concept built around communication protocols and devices to a multidisciplinary domain in which devices, the Internet and people merge to create a full system to help business innovation and interoperability. In this way, research and development are crucial to make the creation of smart environments happen.

IoT enables objects surrounding humans to become active participants, sharing information and capable of recognizing changes in their surroundings and reacting autonomously to events. Nowadays, the Internet of Things has reached a level where the physical objects are integrated into the data network becoming participants in business processes and decisions. The main concept of the IoT is: Internet-connected devices everywhere in any time and any place. These devices are able to communicate with humans who monitor and control them taking into account security and privacy for the actual users.

Although many of the IoT systems and technologies are relatively novel, the current application areas of the IoT are already very diverse: smart homes, smart cities, industrial automation, smart mobility, health care, etc. In other words, IoT applications are changing the way we work and live by saving time and resources and opening new opportunities for growth, innovation and knowledge. However, there are still many application areas yet unexplored and a quantity of issues to solve.

The potential benefits of IoT are almost limitless. It allows public and private organizations to better manage assets, optimize their performance, reduce costs and exploit new business models. Other benefits are the improvement of perception that users feel in the optimization of mobility and transport, the increase of independence, getting better healthcare, enhancing their comfort or saving energy and costs.

The current areas of research are the following:

- Application in IoT: firstly, IoT applications relied on sensor network applications but new research considers smart monitoring systems with wireless sensors and actuator networks.
- Cloud services in IoT: the core function to provide valuable services in IoT. Reducing the amount of data stored at IoT devices is the way to build flexible and stable IoT systems.
- Protocols: routing protocols have an important role to realize practical wireless networks
- Security: together with privacy, are two of the main issues of the IoT systems which keep users being concerned and skeptical about the new technology.

3.2.1. IoT Architecture

To make IoT happen, several components must come together and stay synchronized. A simplified structure would be as shown in the Figure 3, where the main parts are: sensors, gateway, network, management services and the applications.

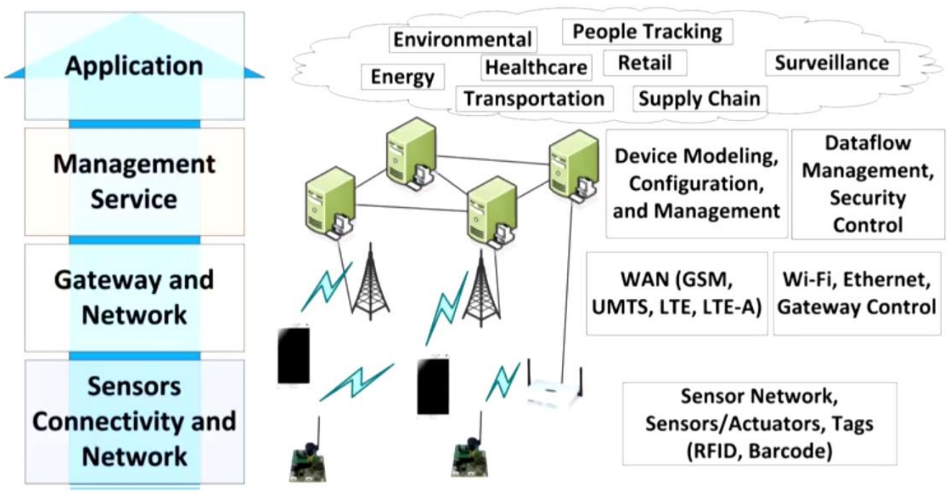

Application
Management Service
Gateway and Network
Sensors Connectivity and Network
Environmental
People Tracking
Energy
Healthcare
Retail
Surveillance
Transportation
Supply Chain
Device Modeling, Configuration, and Management
Dataflow Management, Security Control
WAN (GSM, UMTS, LTE, LTE-A)
Wi-Fi, Ethernet, Gateway Control
Sensor Network, Sensors/Actuators, Tags (RFID, Barcode)

Figure 3 IoT Architecture

Source: https://www.coursera.org/learn/iot-augmented-reality-technologies/lecture/8ZlnC/iot-architecture

3.2.1.1. Sensor Layer

The sensor layer is made up of sensors and smart devices which provide real time information that is collected and processed. Sensors use low power and low data rate connectivity such as Bluetooth Low Energy (BLE), which does not possess the same level of universal access to the Internet due to battery constraints and lifetime considerations of the sensors.

Sensors can be seen in everyday objects from any tactile button to lamps that vary the intensity of their light according to the light of the room. The truth is that most of them people are not aware of but they are surrounding our everyday lives.

Every sensor has some components that together with other sensors create the Wireless Sensor Network (WSN). These components are: a radio transceiver with an internal antenna or connection to an external antenna, a microcontroller, an electronic circuit and an energy source which is usually a battery or an embedded form of energy harvesting. Sensors are grouped according to data types, such as home sensors, surveillance sensors, environmental sensors, etc.

The main characteristics of the WSN are:

- Power consumption constraints for nodes using batteries or energy harvesting
- Ability to cope with sensor failures (resilience)
- Mobility of sensors
- Heterogeneity of sensors
- Scalability to large scale of deployment

- Ability to withstand harsh environmental conditions
- Ease of use
- Cross-layer design

The WSN is connected to the Internet through WAN or LAN networks so that collected data can be transmitted and analyzed for its use in the applications. However, retrieving data from each sensor is often challenging due to the sensor power constraints or because of poor wireless connectivity or expensive data links. One solution that has been extensively studied is to mesh-network sensors to allow data packets to jump through the network, but this is often unsatisfactory especially in areas with poor radio frequency signal. Also, the demands of data forwarding take a substantial quantity of sensor lifetime.

3.2.1.2. Gateway Layer

A gateway is a sensor aggregator which gathers all data sensors have collected through networking connectivity. This can be Local Area Network (LAN), which basically means Wi-Fi and Ethernet connections, and also it can be through Personal Area Network (PAN) which includes Bluetooth, ZigBee and 6LowPAN.

The gateway needs to include micro-controllers, a radio communication module, signal processors and modulators, an access point… to enable technologies or systems with disparate protocols interact with one another and integrate heterogeneous networks into a single IoT platform. Gateways must handle enormous amounts of data and that is why they require a robust and solid performance regarding both public and private networks. Also, they are low energy consumers like the sensors.

The gateway needs to be scalable to efficiently serve a wide range of services and applications over large-scale networks. Typically, the gateways speak a proprietary protocol between the connected devices and then allow connectivity through the gateway using a standard protocol such as HTTP.

3.2.1.3. Management Service Layer

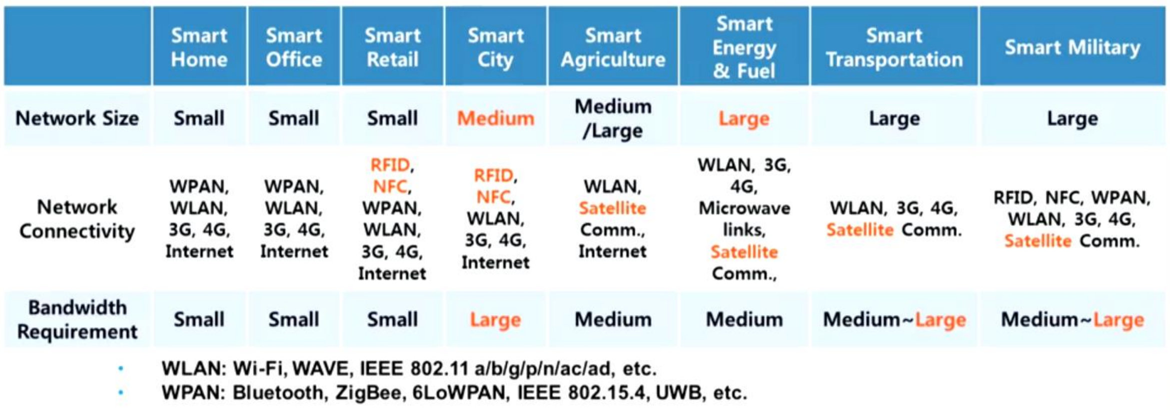
The management service layer is in charge of the data analytics, security control, process modeling and device management. It takes charge of part of the data management with tasks like filtering and control which data is needed in each case. On the other hand, it also takes care of other operations that require immediate response and delivery, for example, patient medical emergency sensor data.

The management service layer also extracts data to process, and provides an abstract view of the overall data for the application layer to show to the user.

3.2.1.4. Application Layer

As mentioned before, the IoT can be used for service enhancement in a wide range of areas ranging from environment to healthcare going through energy and transportation. Applications can be classified according to network availability, coverage, size, heterogeneity, business model, real-time or not.

Below, in Figure 4 and Figure 5, there is a classification of different IoT areas of service with their network characteristics and different offered services.

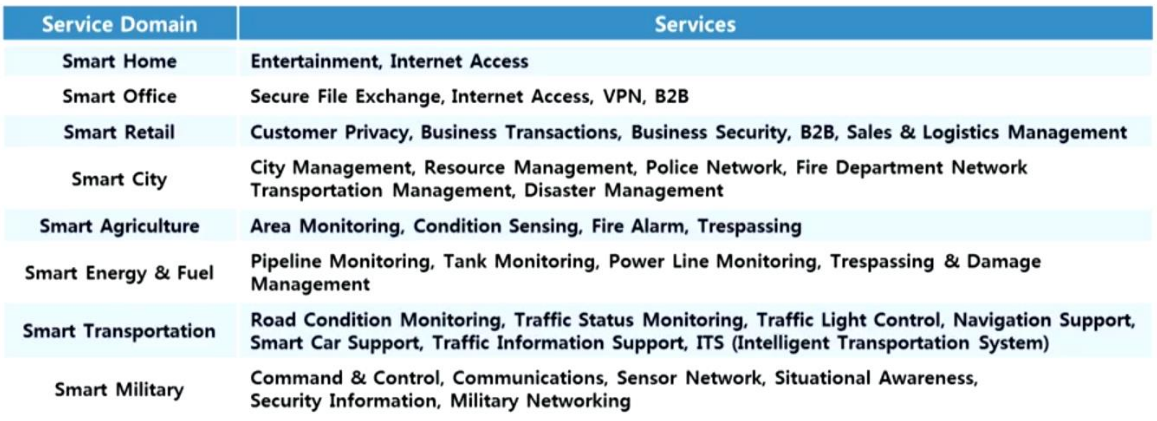


	Smart Home	Smart Office	Smart Retail	Smart City	Smart Agriculture	Smart Energy & Fuel	Smart Transportation	Smart Military
Network Size	Small	Small	Small	Medium	Medium /Large	Large	Large	Large
Network Connectivity	WPAN, WLAN, 3G, 4G, Internet	WPAN, WLAN, 3G, 4G, Internet	RFID, NFC, WPAN, WLAN, 3G, 4G, Internet	RFID, NFC, WLAN, 3G, 4G, Internet	WLAN, Satellite Comm., Internet	WLAN, 3G, 4G, Microwave links, Satellite Comm.,	WLAN, 3G, 4G, Satellite Comm.	RFID, NFC, WPAN, WLAN, 3G, 4G, Satellite Comm.
Bandwidth Requirement	Small	Small	Small	Large	Medium	Medium	Medium~Large	Medium~Large

- WLAN: Wi-Fi, WAVE, IEEE 802.11 a/b/g/p/n/ac/ad, etc.
- WPAN: Bluetooth, ZigBee, 6LoWPAN, IEEE 802.15.4, UWB, etc.

Figure 4 IoT Areas of service and Network characteristics

Source: Source: https://www.coursera.org/learn/iot-augmented-reality-technologies/lecture/8ZInC/iot-architecture



Service Domain	Services
Smart Home	Entertainment, Internet Access
Smart Office	Secure File Exchange, Internet Access, VPN, B2B
Smart Retail	Customer Privacy, Business Transactions, Business Security, B2B, Sales & Logistics Management
Smart City	City Management, Resource Management, Police Network, Fire Department Network Transportation Management, Disaster Management
Smart Agriculture	Area Monitoring, Condition Sensing, Fire Alarm, Trespassing
Smart Energy & Fuel	Pipeline Monitoring, Tank Monitoring, Power Line Monitoring, Trespassing & Damage Management
Smart Transportation	Road Condition Monitoring, Traffic Status Monitoring, Traffic Light Control, Navigation Support, Smart Car Support, Traffic Information Support, ITS (Intelligent Transportation System)
Smart Military	Command & Control, Communications, Sensor Network, Situational Awareness, Security Information, Military Networking

Figure 5 Different service domains in IoT

Source: https://www.coursera.org/learn/iot-augmented-reality-technologies/lecture/8ZlnC/iot-architecture

3.3. Future prospects

In a short future, the IoT could permeate the whole economy and society if the general concerns are addressed properly and with warranties and the potential market demand assimilates with any doubt the new technology as a new player in everyone's lives.

However, the evolution of the IoT is a constant challenge involving aspects like communication, data processing and storage, self-adaptation, resilience, cloud computing and IoT governance. Besides, there are still fields that have not yet been explored that could participate in improvements or the solving of other issues.

In the future, the number of connected devices will grow exponentially, therefore the inter-operability between devices will be crucial and so less power and lower cost will be more and more required as well as an improvement of battery efficiency. The overall challenge is to expect that networks of devices, sensors and actuators will work in complete synergy and will be dynamically configured to improve the quality of our lives. Given the giant expected growth of network usage and the number of user nodes that it will imply, there is the real need to minimize the resources to implement all network elements and the energy being used for their daily operation.

Like said before, two of the main concerns of the application of the IoT are security and privacy. The full potential of the IoT will come with specific strategies that respect the privacy rights and individual security and addressing these two aspects and ensuring the intimacy of the customers is a fundamental priority in the development of the IoT. Users will need to trust in IoT devices as secure services away from vulnerabilities especially because it is a technology that pretends to be fully integrated in our daily lives.

In this way, the IoT is in process of redefining the principles of these two main concerns as many of the implementations can dramatically change the way personal data is collected, analyzed and used.

4. Chapter 2: Literature review

4.1. DRINA: A routing approach for In-network aggregation in WSN

To solve the routing optimization problem in the IoT, some approaches have been carried out. In particular, the paper of *L.A. Villas et al. (2013)* presents a new approach to the problem by offering a brand-new solution. For them, the increase of connected devices will lead to redundant data captured and sent and so they propose a method in which the redundant data is aggregated in intermediate nodes reducing the number of exchanged messages and so the energy consumption as well as extending the network lifetime.

The key features of their method are: reduced number of messages by setting a routing tree, maximizing the number of overlapping routes, the high aggregation rate and reliable data aggregation and transmission. The authors compared their solution to two existing solutions, which are the Shortest Path Tree (SPT) and the Information Fusion-based Role Assignment (InFRA).

It is believed that in sensors the energy consumption is normally associated with the amount of gathered data since communication is often the most expensive activity in terms of energy cost. For that, a possible strategy to routing optimization is to use intermediate sensors which will aggregate data coming from other nodes (sensors) and will report data by making local decisions. This strategy is called as data-centric routing or in-network data aggregation.

The authors of the paper consider different types of nodes, which are the following:

- Collaborator: the one that detects the event and sends the gathered data to a coordinator node

- Coordinator: it has the function of detecting the events and also of gathering all data received, aggregating it and sending it to the sink node

- Sink node: node that receives data from coordinators and collaborators

- Relay: node that forwards data to the sink node

And so, the main goal of the algorithm is to build a routing tree with the shortest paths which connect all nodes to the sink, maximizing data aggregation. This is done in three phases:

- Phase 1: building of the hop tree from the sensors to the sink node. Also, the sink node starts the building of the hop tree that coordinators will use to send their data.

- Phase 2: formation of the cluster and election of the cluster-head among the nodes that detected events in the network.

- Phase 3: setting up of a new route for the sending of data and updating the hop tree.

All this corresponds to a cluster-based approach where nodes are divided into clusters and some nodes (cluster-heads) are elected to aggregate the data coming from others and send it to the sink node. It is different from the SPT approach where every node that detects an event on the network sends it directly to the sink node by using the shortest path. Also, the approach takes into account different types of data collected and a dynamic network in continuous change by the detection of new events. In this way, nodes route data based on their content and choose the next hop that maximizes the overlap of routes in order to fulfill the in-network data aggregation.

The algorithm presented solves the case when for each new event detected, the nodes that detected the same one are clustered and the cluster-head is elected. Afterwards, routes are created by calculating the shortest path in each case to the nearest node that is already part of an existing routing structure where this node will be the aggregation point. The same algorithm tries to maximize the number of aggregation points.

4.2. Clarke & Wright's Savings Algorithm

The thesis is based on a concept widely used in the transportation network, and adapts it to the data network. The concept is referred to as the Vehicle Routing Problem.

Also known as VRP, the Vehicle Routing Problem is a problem present in the field of logistics and transportation. The problem arises when from a distribution center, a fleet of vehicles has to deliver products to a list of clients, spread out in a geographical area. The problem is making the decision of which are going to be the delivery routes, which clients will be in which routes, in which sequence clients should be visited and how many vehicles are necessary to deliver everything. All these decisions must be made in order to accomplish the principal objective of reaching the minimum total cost possible.

The VRP first appeared in a paper by *George Dantzig and John Ramser (1959)* and since then, a wide range of varieties have been explored by multiple authors. Widely, the solutions proposed are heuristics and there is one especially known for its speed in the resolution process and also for its good results when comparing it with exact solutions. This heuristic is named *Clarke & Wright's Savings Algorithm (1964)* and it is based in the savings concept.

The savings algorithm compares an initial situation where two clients are visited in separate routes and compares its cost with the cost in case the pair of clients were visited in the same route. In both cases, the vehicle starts from the distribution center and finishes at the same point. The Figure 6 illustrates this process and the calculus is the following:

- The cost of the two initial routes is: $\mathbf{D_a = c_{0i} + c_{i0} + c_{0j} + c_{j0}}$ **[Eq. 13]**

- The cost of the single route with both clients: $\mathbf{D_b = c_{0i} + c_{ij} + c_{j0}}$ **[Eq. 14]**

If the two options are combined, one obtains the savings that imply the fact of putting both clients in the same route:

$$\mathbf{S_{ij} = D_a - D_b = c_{i0} + c_{0j} - c_{ij}} \quad \textbf{[Eq. 15]}$$

Note that these savings can be both positive or negative, depending on each pair of nodes, but the ones that one cares are those who are positive and bigger the better, which means there is an actual saving in merging the two routes and client j will be visited just after client i. This process is done by all pairs of nodes in a given transportation network and so the savings turn to be a matrix of savings in which every cell tells the savings in merging two particular nodes.

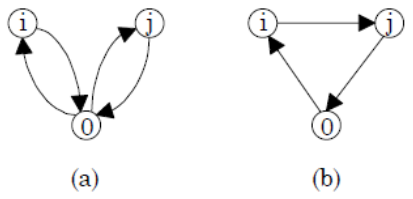


Figure 6 Illustrating the Clarke & Wright's Savings Algorithm
Source: Clarke & Wright's Savings Algorithm. 1964

To apply this concept to the data network, especially in the Internet of Things, some modifications should be made, and they will be explained in the next section.

After having the saving's matrix, one takes the biggest value, that means that if the two nodes associated were in the same route, the maximum safe will be accomplished. These two nodes will be in the same route only if the constraints can be fulfilled. If these two nodes cannot be together because of capacity issues or other constraints, the next big value of the savings matrix is picked and the algorithm tries to do the same with the new pair of nodes. This process is repeated until every node is linked to another.

5. Chapter 3: Methodology

5.1. Adaptation of the savings concept

As shown, the problem has been modeled using integer linear programming (ILP), which is known to be highly complex. Its complexity has been proven NP-hard, and so a solution cannot be done in real-time. In this thesis, we propose adapting the savings concept in the transportation network (TN) to the data network (DN) to solve the problem. In this section, the modifications made are explained:

- The main difference between both networks is that in the TN, if a truck does not carry any product but it is moving from one client to the distribution center, there are still associated costs (fuel consumption, CO_2 emissions, …). In the DN there is only cost if some data is sent between a pair of nodes.

- In the present case, data is only one-way, from the sensors to the gateway. In the TN, the trucks go from the distribution center to the clients and vice versa.

- In the TN the capacity relies on the types of vehicles, capable of carrying a certain amount of product. In the DN, the capacity relies on each node, and this can be different every time.

- If in the TN clients have some constraints like time-windows when they can be served, the constraints in the DN nodes are: the space left they have to receive more data from other nodes, the battery left they have, the cost that implies for each one of them the shipment of data and the maximum distance they can be connected to other nodes.

With these considerations set out, the savings equation changes and is the following:

$$S_{ij} = C_{i3}*D_{i0}*C_{i0} – (C_{i3}*D_{ij}*C_{i0} + C_{j3}*D_{j0}*C_{i0}) \quad \text{[Eq. 16]}$$

Note that node 0 is the gateway. The explanation of the savings matrix is the following:

- The first term is the cost it will imply if data from node i is sent directly to node 0: **[cost of sending for node i * distance from i to 0 * quantity of data node i has]**

- The second term is the cost it will imply if data from node i is sent to node 0 through node j:

  [cost of sending for node i * distance from i to j * quantity of data node i has]

  +

  [cost of sending for node j * distance from j to 0 * quantity of data node i has]

- If the subtraction is positive, it means that sending the data from i to 0 directly is more expensive than send it through j, so there is saving in sending it through j. Finally, the link between i and j will be feasible only if all constraints are accomplished.

In the Figure 7 the theory explained is shown graphically so to understand it better. The first term is represented by the light-colored line and the second term by the dark one:

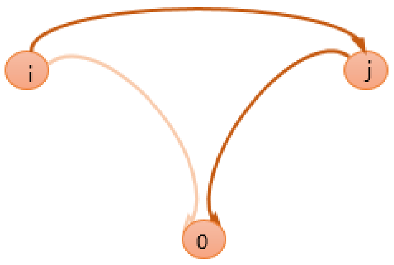


Figure 7 Illustrating the savings concept in the present algorithm
Source: Own

If what has been explained above is for one pair of nodes, the same calculus is made for every pair of nodes the data set has in each case. When the savings are calculated for every pair of nodes, the savings turn into a matrix that is then sorted from the maximum value to the minimum. When this is done, the algorithm starts picking the pair of nodes that imply the maximum savings value and looks if the first node can actually send its data to the second one, regarding the constraints of each node. If the answer is positive, a link has just been set and all the variables need to be recalculated. That means, the characteristics of these two nodes have changed, and also a recalculation of the savings implying these two nodes will need to be recalculated. The functioning of the algorithm will be widely explained in the following section.

5.2. The algorithm in Python

5.2.1. General explanation

The implementation of the proposed algorithm is composed of 3 Python files: algorithm_new.py, algorithm_new_2.py and ejec.py.

The solution of the algorithm is compared with two base lines: BL and BL2. The first one is the solution where all nodes are sending directly to the gateway. The second one, which corresponds to the file algorithm_new_2.py, is the solution where at most there are two steps before reaching the gateway: a node is sending to another, and this second is sending to the gateway. No path having more than two steps is considered. In the Annex 1, all the code is attached.

5.2.1.1. Algorithm_new.py

It is the file where the entire algorithm is coded and it is composed by 5 functions (note that N is the total number of nodes, including the gateway):

- *max_savings(SS,N)*: given SS as the savings matrix (calculated in the ejec.py file), the *max_savings* function sorts from the maximum value to the minimum, saving also both nodes involved in each value. For example: `Sm = max_savings(SS,N) = [[1*,2*,10],[2,3,6],[3,2,5],[1,3,4],[3,1,3],[2,1,2],[1,0,0],[2,0,0],[3,0,0]]` Where each sub-list is composed by: source node, end node, saving value.

- *depth(N)*: it creates a list of length N with all zeros. In the *algor* function, zeros will increase when a node starts having sons.

- *parents_sons(N)*: it creates an empty list of length N. The list will be used for filling both parents and sons independently for each node in the *algor* function.

- *XY(N)*: it creates a matrix NxN with all zeros. They will become ones in the *algor* function when a pair of nodes is connected through a link.

- *algor(D,C,N,S,Info_left,Cost_i)*: this is the main function where the whole algorithm is coded. Parameters of the function are:

    - D: distance matrix (calculated in the ejec.py file)
    - C: characteristics matrix (calculated in the ejec.py file)
    - N: total number of nodes (calculated in the ejec.py file)
    - S: savings matrix (calculated in the ejec.py file)
    - Info_left: list of length equal to the number of nodes that initially is the characteristic C_{i0}, and is updated to 0 through the algorithm when the node in question is sending its data to another.

- Cost_i: list of length equal to the number of nodes that represents the cost for a node to send its data, initially to the gateway (calculated in the ejec.py file). It will be updated through the algorithm if the node is sending its data to another. This cost is calculated as: **Cost_i = C_{i0} * C_{i3} * D_{i0}** **[Eq. 17]**

A detailed picture of what it does is explained through a Block Diagram in next section (5.1.2.2). However, in general terms what it does is from the matrix resultant from the *max_savings* function, it tries to send the info of the **first node*** to **the second*** regarding their constraints. If the merge is possible, the savings affected are re-calculated and the *max_savings* is called another time. If it is not possible, the algorithm tries the same with the **next element*** of the initial matrix of *max_savings*. And it does the same until all nodes are sending to another one. More detailed would be:

- The function calls the other functions of the code and initialize some other variables, which are the following:
    - Links[]: list that will include all links of the solution. Each element of the list will be composed of: [node origin, end node, quantity of data carried]
    - Space[]: list of length equal to the number of nodes and it reflects the space each node has originally calculated as total capacity – data gathered (C_{i1}-C_{i0}). It is updated every time a link is set, reducing the space the node has with the info added that it is carrying.
    - bat[]: list of length equal to the number of nodes and it reflects the remaining battery each node has. Originally is the characteristic C_{i2} but it is updated every time a link is set, subtracting the depth of the node sending + 1 second it needs to actually send the data.
    - IC[]: list of length equal to the number of nodes and it reflects the data one node is carrying. Originally is the characteristic C_{i0} but it is updated every time a link is set, accumulating the info added that it is carrying.
- A loop starts and does not end until all links are set. Inside the loop, several things are done for each iteration. An iteration responds to a pair of nodes. Like said before, it starts with the first element of *max_savings* and tries to send the data of the **first node*** to **the second***. The main things done for each iteration are:

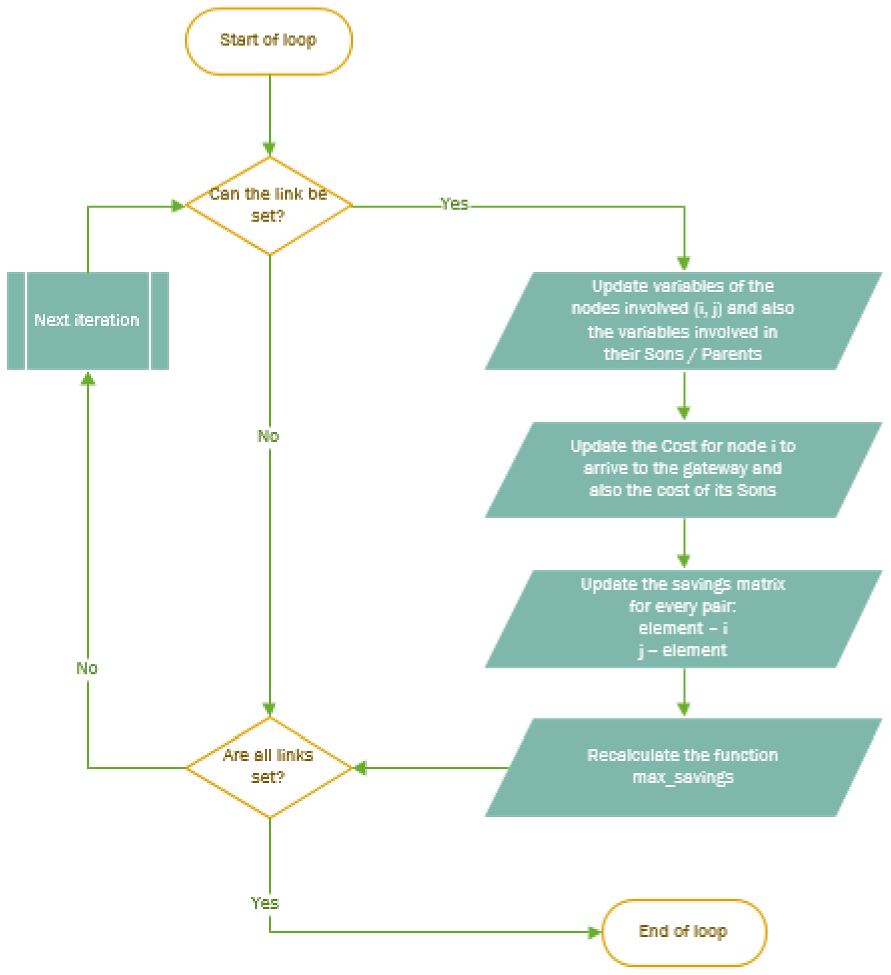

Start of loop
Can the link be set?
Yes
Update variables of the nodes involved (i, j) and also the variables involved in their Sons / Parents
Update the Cost for node i to arrive to the gateway and also the cost of its Sons
Update the savings matrix for every pair: element – i j – element
Recalculate the function max_savings
Next iteration
No
No
Are all links set?
Yes
End of loop

Figure 8 Illustrating the loop in the algor function

Source: Own

- After the loop, first what is done is that all nodes who do not have Parents means that they are sending all their data to the gateway directly. And so, these links are added to the Links matrix.

- Finally, what is done is the calculation of the total cost with all links set. It is calculated like the sum of the following equation (Eq. 18) for every element in the Links matrix (always i as origin node and j as end node):

  C_{i3} * D_{ij} * Info carried from $\underline{i}$ to $\underline{j}$ **[Eq. 18]**

5.2.1.2. Algorithm_new_2.py

In this file, the base line 2 is coded. The program is very similar to algorithm_new.py but it has two new conditions inside the main loop which are:

[length of j's parents less than 1 and length of i's sons less than 1]

Because in a general case, trying to send info from i to j: if node i has already a son, then i must go directly to the gateway. Or, if j has a parent, j cannot have sons because the chain would be longer than two steps before reaching the gateway. In both cases, the algorithm assures that a chain of nodes has at maximum two steps.

5.2.1.3. Ejec.py

In this one, both prior files are imported in order to display the different solutions. In ejec.py, two things are asked at the beginning: the user has to type how many times he wishes the algorithm will run and for which N's (the user types a number and the algorithm runs for random N's between the number the user has typed and the same number – 10 units). For example, if the user has typed N=50 and 50 times, the algorithm will run for 50 times for N's between 40 and 50 randomly picked.

Once the range of N is known, a characteristic of this file is that, the whole data set is generated randomly. That means: the exact N in each case, each one of the characteristics of each node and the distance matrix. After the distance matrix is set, the savings matrix is calculated in the same file.

The characteristics of each node are calculated with certain limited ranges. These are the following:

- Data a node gathers, C_{i0}: Random (1,100) [data/sec]
- Capacity of the node, C_{i1}: C_{i0} + Random (1,100) [data/sec]
- Remaining battery, C_{i2}: Random (1,10000) [sec]
- Cost of sending data, C_{i3}: Random (0,1) * 0,00001 [$/1data*1m]
- Max. distance, C_{i4}: Random (1,500) [m]

It is considered that node 0 (gateway): it gathers 0 data itself, it has infinite capacity, infinite battery, 0 cost of sending data and infinite maximum distance to be connected to other nodes.

The characteristic C_{i4} is applied only when considering the link from a node with another one which is not the gateway. When it comes to link whichever node to the gateway, this maximum distance is not considered as it is thought that the gateway can reach every node in the network.

Afterwards, both algorithm_new.py and algorithm_new_2.py are called and their solutions are written in a file like this (for example: Solution25.txt):

```
Data set: 25
Execution time: 0.126000165939 seconds
Cost BL: 1.31548952971 $
Cost BL2: 0.699080983883 $
Cost Algorithm: 0.697014240175 $
Links: [[11, 8, 86], [10, 1, 43], [3, 4, 54], [12, 7, 28], [5, 7, 49], [15, 9,
55], [2, 13, 28], [16, 10, 1], [1, 0, 91], [4, 0, 100], [6, 0, 44], [7, 0,
127], [8, 0, 183], [9, 0, 79], [13, 0, 101], [14, 0, 9]]
Links_2: [[11, 8, 86], [10, 1, 42], [3, 4, 54], [12, 7, 28], [5, 7, 49], [15,
9, 55], [2, 13, 28], [1, 0, 90], [4, 0, 100], [6, 0, 44], [7, 0, 127], [8, 0,
183], [9, 0, 79], [13, 0, 101], [14, 0, 9], [16, 0, 1]]
```

- Data set: number of the current data set (out of the total number of data sets that the user has decided to run).
- Execution time: time in seconds that the computer needed to run all the program.
- Cost BL: cost of the base line, initial case where all nodes send directly to the gateway.
- Cost BL2: cost of the base line 2, where there are at the most two stages before reaching the gateway.
- Cost Algorithm: cost of the real algorithm where chains of more than 2 nodes can be unified.
- Links: actual links of the algorithm's solution which involve origin node, end node, quantity of data carried.
- Links_2: actual links of the BL2 solution.

This output file is created for every data set. Besides, a global output file is also created with the same information showed but with all files together in one (for example: AllSolutions N 10_20.txt).

Also, the data of each data set is also recorded in separate files (for example: DataSet25.txt) and in one single file (for example: AllData N10_20.txt). The first one looks like this:

```
Data set: 25
17
5
{'C':[[48,   97,   6103,   7.419036162921356e-07,   321],   [28,   60,   4904,
6.363697232753913e-06, 104], [54, 142, 215, 5.27460602666664e-06, 175], [46,
140, 3104, 2.2448207250324204e-06, 275], [49, 124, 3633, 7.0913624004409336e-
06,  483],  [44,  90,  5101,  6.2735591090989455e-06,  220],  [50,  143,  3478,
9.22962330191946e-06, 189], [97, 196, 5903, 2.4080985353418783e-06, 251], [24,
108, 728, 3.7204135682750507e-06, 247], [42, 111, 9195, 9.508927587725927e-06,
379],   [86,   116,   4569,   7.975083324209085e-06,   278],   [28,   125,   916,
7.498307280893712e-06, 274], [73, 160, 3607, 3.964219808011128e-06, 461], [9,
35, 9659, 3.3154040432458636e-06, 98], [55, 89, 3110, 4.8307231427667635e-06,
331], [1, 57, 788, 5.687843562179042e-06, 73]]}
{'D':[[0.0, 247.0, 363.0, 385.0, 211.0, 332.0, 234.0, 117.0, 182.0, 241.0,
404.0, 483.0, 400.0, 401.0, 240.0, 414.0, 490.0], [247.0, 0.0, 80.0, 481.0,
356.0, 243.0, 239.0, 73.0, 248.0, 206.0, 17.0, 75.0, 127.0, 157.0, 460.0, 19.0,
250.0], [363.0, 80.0, 0.0, 435.0, 364.0, 315.0, 466.0, 368.0, 328.0, 303.0,
468.0, 319.0, 172.0, 98.0, 458.0, 446.0, 271.0], [385.0, 481.0, 435.0, 0.0,
58.0, 171.0, 39.0, 364.0, 23.0, 99.0, 314.0, 128.0, 328.0, 226.0, 228.0, 162.0,
190.0], [211.0, 356.0, 364.0, 58.0, 0.0, 116.0, 438.0, 100.0, 144.0, 363.0,
159.0, 273.0, 258.0, 454.0, 394.0, 253.0, 251.0], [332.0, 243.0, 315.0, 171.0,
116.0, 0.0, 304.0, 61.0, 192.0, 128.0, 35.0, 133.0, 78.0, 126.0, 89.0, 336.0,
87.0], [234.0, 239.0, 466.0, 39.0, 438.0, 304.0, 0.0, 272.0, 365.0, 275.0,
237.0, 73.0, 449.0, 102.0, 11.0, 65.0, 264.0], [117.0, 73.0, 368.0, 364.0,
100.0, 61.0, 272.0, 0.0, 150.0, 426.0, 147.0, 311.0, 10.0, 346.0, 438.0, 209.0,
448.0], [182.0, 248.0, 328.0, 23.0, 144.0, 192.0, 365.0, 150.0, 0.0, 70.0,
164.0, 21.0, 181.0, 50.0, 349.0, 289.0, 235.0], [241.0, 206.0, 303.0, 99.0,
363.0, 128.0, 275.0, 426.0, 70.0, 0.0, 191.0, 150.0, 61.0, 382.0, 246.0, 126.0,
296.0], [404.0, 17.0, 468.0, 314.0, 159.0, 35.0, 237.0, 147.0, 164.0, 191.0,
0.0, 39.0, 414.0, 166.0, 292.0, 48.0, 66.0], [483.0, 75.0, 319.0, 128.0, 273.0,
133.0, 73.0, 311.0, 21.0, 150.0, 39.0, 0.0, 349.0, 402.0, 73.0, 3.0, 416.0],
[400.0, 127.0, 172.0, 328.0, 258.0, 78.0, 449.0, 10.0, 181.0, 61.0, 414.0,
349.0, 0.0, 423.0, 198.0, 115.0, 170.0], [401.0, 157.0, 98.0, 226.0, 454.0,
126.0, 102.0, 346.0, 50.0, 382.0, 166.0, 402.0, 423.0, 0.0, 151.0, 410.0,
105.0], [240.0, 460.0, 458.0, 228.0, 394.0, 89.0, 11.0, 438.0, 349.0, 246.0,
292.0, 73.0, 198.0, 151.0, 0.0, 32.0, 398.0], [414.0, 19.0, 446.0, 162.0,
253.0, 336.0, 65.0, 209.0, 289.0, 126.0, 48.0, 3.0, 115.0, 410.0, 32.0, 0.0,
135.0], [490.0, 250.0, 271.0, 190.0, 251.0, 87.0, 264.0, 448.0, 235.0, 296.0,
66.0, 416.0, 170.0, 105.0, 398.0, 135.0, 0.0]]}
```

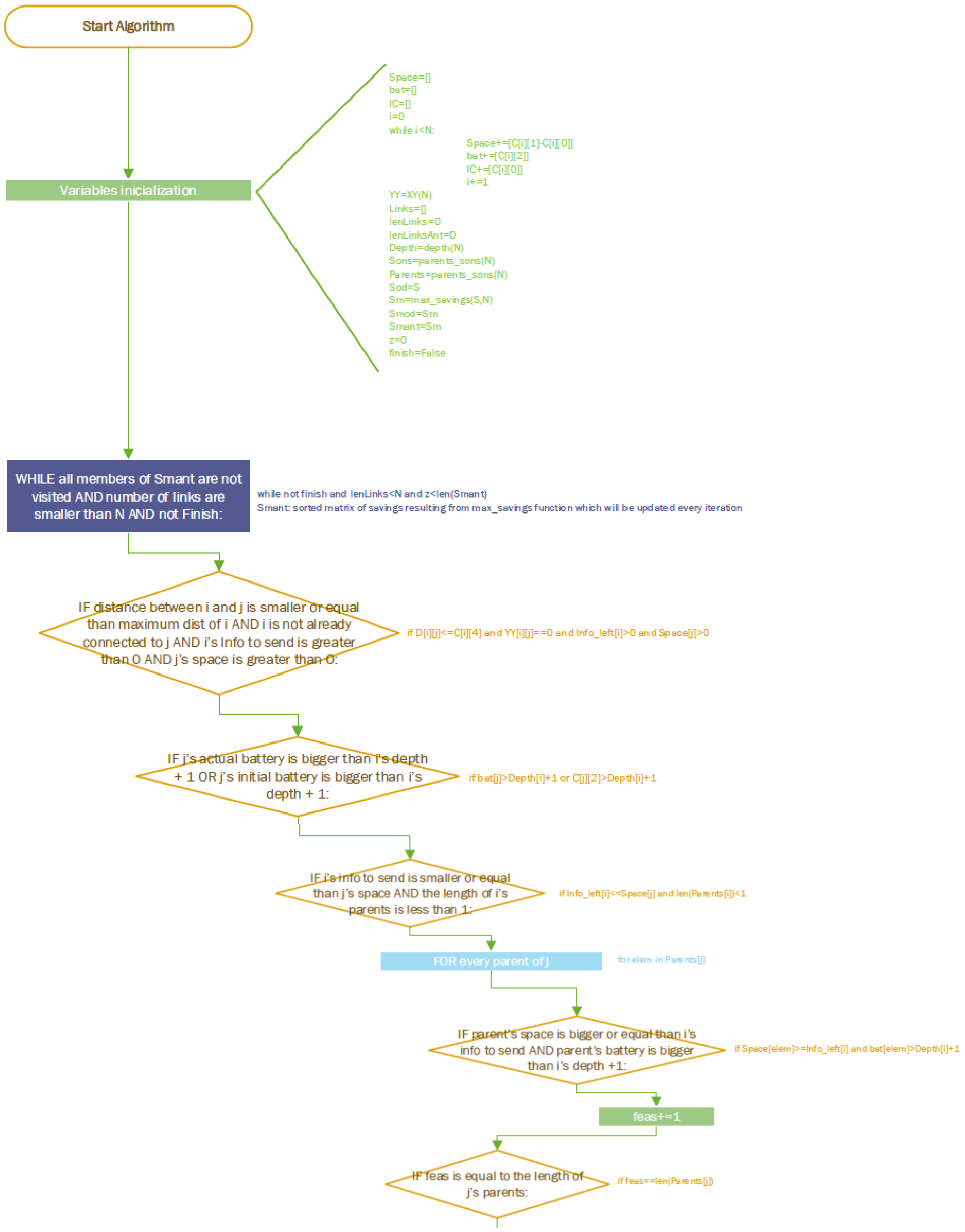
- Data set: number of the current data set (out of the total number of data sets that the user has decided to run).
- 17: number of nodes for this data set including the gateway.
- 5: number of characteristics nodes have (fixed number for each data set).
- "C": dictionary of all nodes with their characteristics (gateway characteristics are not shown). Each node is represented by a list like this: `[48, 97, 6103, 7.419036162921356e-07, 321]`. The node's characteristics are defined in the same way as described in the Problem Statement section (2.2).

- "D": dictionary of all distances between nodes, including distances to the gateway. The distances of each node are represented by a list like this: `[0.0, 247.0, 363.0, 385.0, 211.0, 332.0, 234.0, 117.0, 182.0, 241.0, 404.0, 483.0, 400.0, 401.0, 240.0, 414.0, 490.0]`

It is worth to mention that the file ejec.py was used to create an .exe file so that the program could be run in any operating system.

5.2.2. Block Diagram

In this section, a block diagram of the *algor* function of the file algorithm_new.py is presented (Figure 9) in order to understand the flow that the algorithm.


Start Algorithm
Variables inicialization
WHILE all members of Smant are not visited AND number of links are smaller than N AND not Finish:
IF distance between i and j is smaller or equal than maximum dist of i AND i is not already connected to j AND i's Info to send is greater than 0 AND j's space is greater than 0:
IF j's actual battery is bigger than i's depth + 1 OR j's initial battery is bigger than i's depth + 1:
IF i's info to send is smaller or equal than j's space AND the length of i's parents is less than 1:
FOR every parent of j
IF parent's space is bigger or equal than i's info to send AND parent's battery is bigger than i's depth +1:
feas+=1
IF feas is equal to the length of j's parents:

```
Space=[]
bat=[]
IC=[]
i=0
while i<N:
                    Space+=[C[i][1]-C[i][0]]
                    bat+=[C[i][2]]
                    IC+=[C[i][0]]
                    i+=1
YY=XY(N)
Links=[]
lenLinks=0
lenLinksAnt=0
Depth=depth(N)
Sons=parents_sons(N)
Parents=parents_sons(N)
Sod=S
Sm=max_savings(S,N)
Smod=Sm
Smant=Sm
z=0
finish=False
```

```
while not finish and lenLinks<N and z<len(Smant)
Smant: sorted matrix of savings resulting from max_savings function which will be updated every iteration
```

```
if D[i][j]<=C[i][4] and YY[i][j]==0 and Info_left[i]>0 and Space[j]>0
```

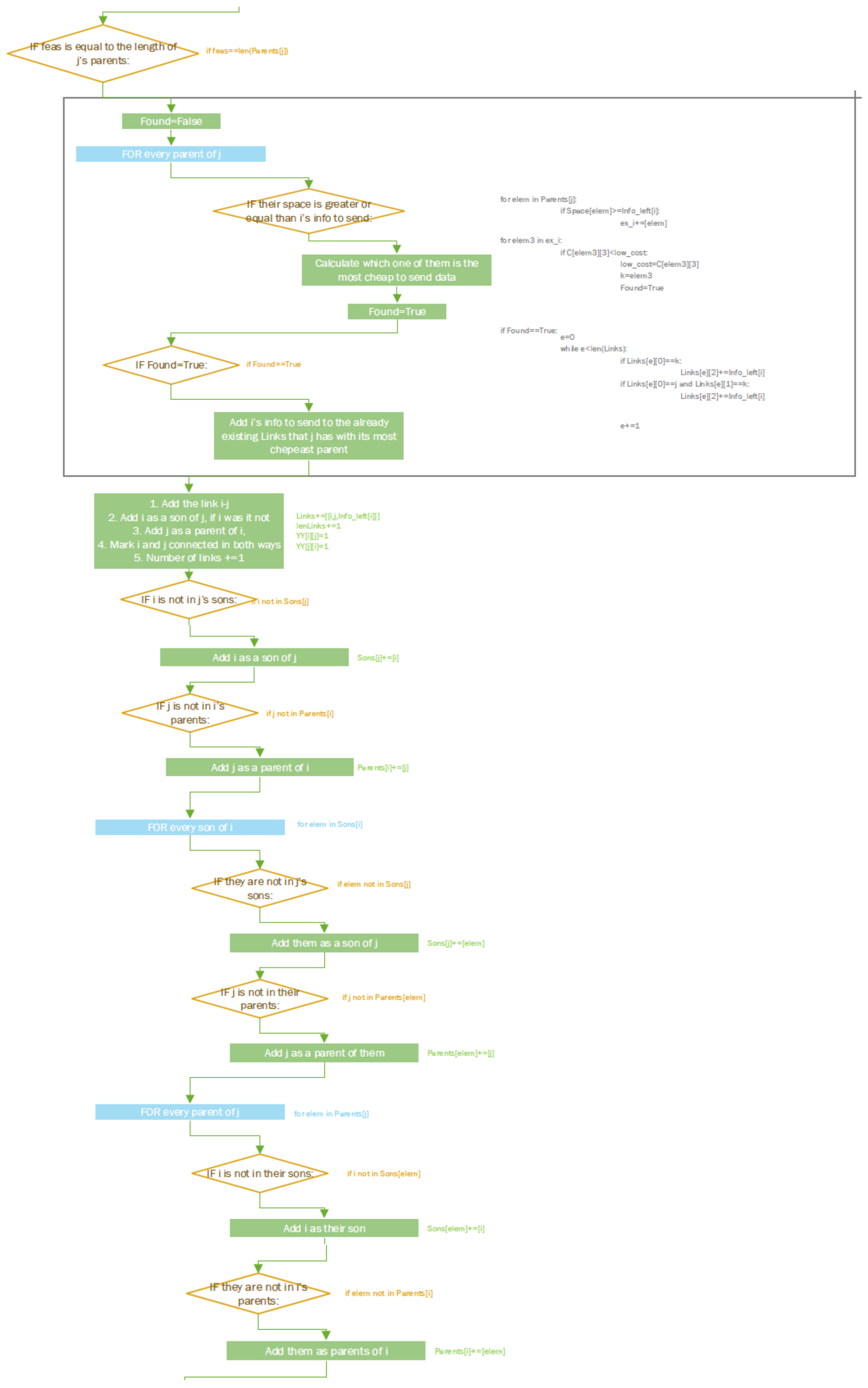
```
if bat[j]>Depth[i]+1 or C[j][2]>Depth[i]+1
```

```
if Info_left[i]<=Space[j] and len(Parents[i])<1
```

```
for elem in Parents[j]
```

```
if Space[elem]>=Info_left[i] and bat[elem]>Depth[i]+1
```

```
if feas==len(Parents[j])
```


IF feas is equal to the length of j's parents:
Found=False
FOR every parent of j
IF their space is greater or equal than i's info to send:
Calculate which one of them is the most cheap to send data
Found=True
IF Found=True:
Add i's info to send to the already existing Links that j has with its most chepeast parent
1. Add the link i-j
2. Add i as a son of j, if i was it not
3. Add j as a parent of i,
4. Mark i and j connected in both ways
5. Number of links +=1
IF i is not in j's sons:
Add i as a son of j
IF j is not in i's parents:
Add j as a parent of i
FOR every son of i
IF they are not in j's sons:
Add them as a son of j
IF j is not in their parents:
Add j as a parent of them
FOR every parent of j
IF i is not in their sons:
Add i as their son
IF they are not in i's parents:
Add them as parents of i

```
if feas==len(Parents[j])
```

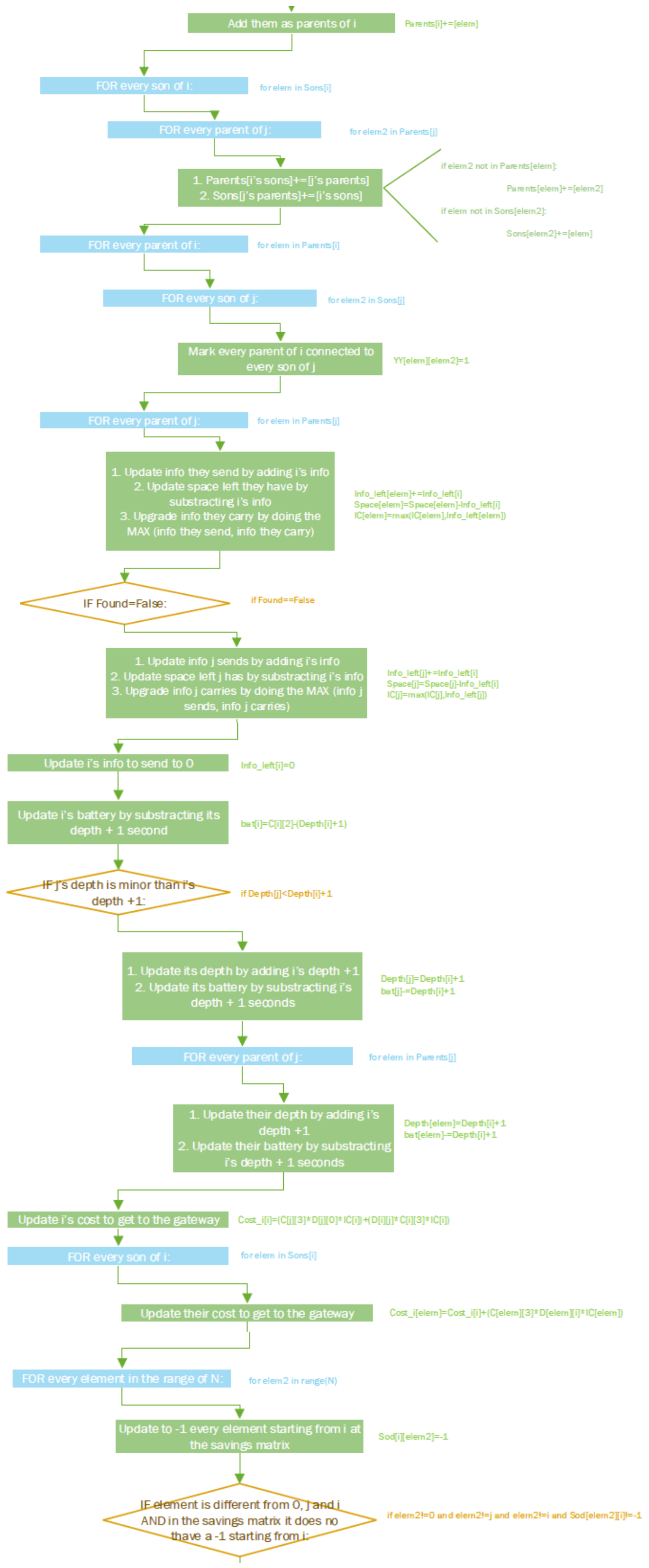
```
for elem in Parents[j]:
        if Space[elem]>=Info_left[i]:
                ex_i+=[elem]

for elem3 in ex_i:
        if C[elem3][3]<low_cost:
                low_cost=C[elem3][3]
                k=elem3
                Found=True

if Found==True:
        e=0
        while e<len(Links):
                if Links[e][0]==k:
                        Links[e][2]+=Info_left[i]
                if Links[e][0]==j and Links[e][1]==k:
                        Links[e][2]+=Info_left[i]

                e+=1
```

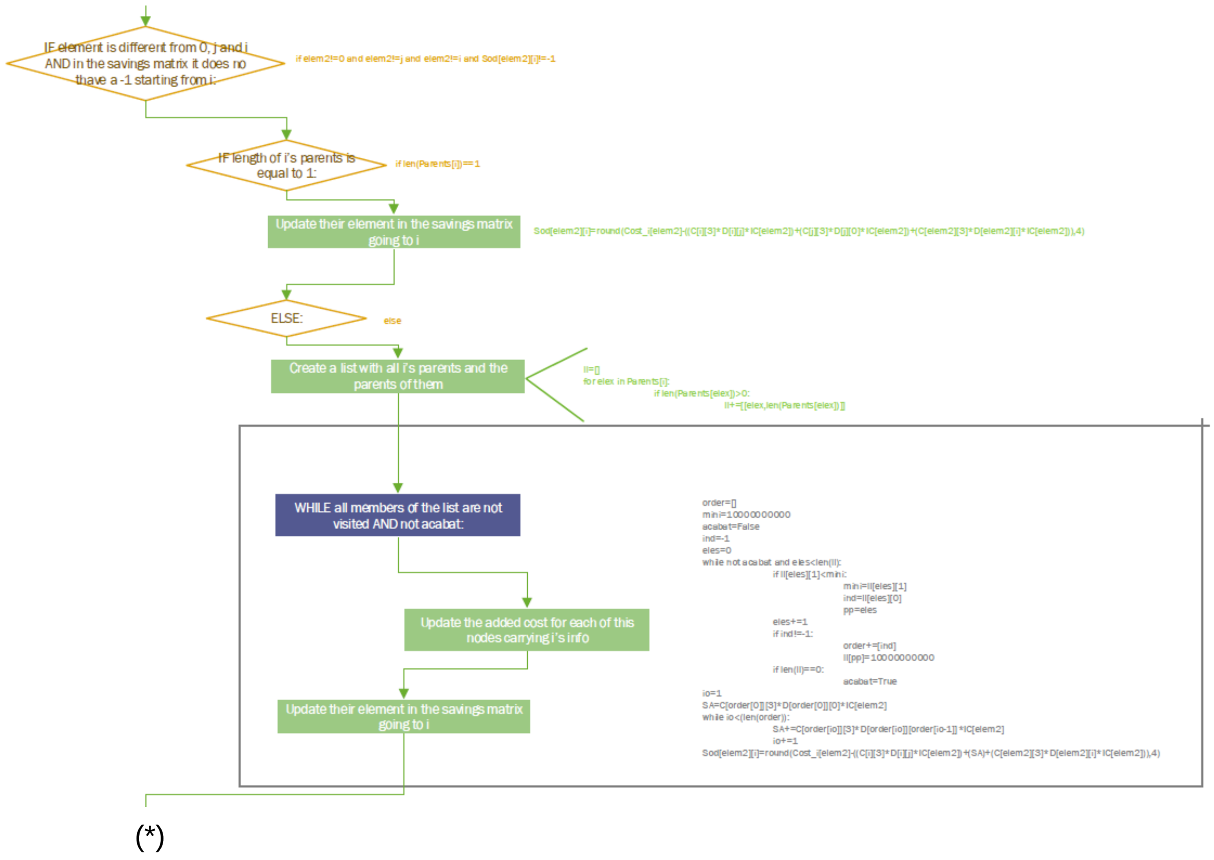
```
if Found==True
```

```
Links+=[[i,j,Info_left[i]]]
lenLinks+=1
YY[i][j]=1
YY[j][i]=1
```

```
if i not in Sons[j]
Sons[j]+=[i]
if j not in Parents[i]
Parents[i]+=[j]
for elem in Sons[i]
if elem not in Sons[j]
Sons[j]+=[elem]
if j not in Parents[elem]
Parents[elem]+=[j]
for elem in Parents[j]
if i not in Sons[elem]
Sons[elem]+=[i]
if elem not in Parents[i]
Parents[i]+=[elem]
```

Le génie pour l'industrie
ETSEIB


Add them as parents of i
Parents[i]+=[elem]
FOR every son of i:
for elem in Sons[i]
FOR every parent of j:
for elem2 in Parents[j]
1. Parents[i's sons]+=[j's parents]
2. Sons[j's parents]+=[i's sons]
if elem2 not in Parents[elem]:
Parents[elem]+=[elem2]
if elem not in Sons[elem2]:
Sons[elem2]+=[elem]
FOR every parent of i:
for elem in Parents[i]
FOR every son of j:
for elem2 in Sons[j]
Mark every parent of i connected to every son of j
YY[elem][elem2]=1
FOR every parent of j:
for elem in Parents[j]
1. Update info they send by adding i's info
2. Update space left they have by substracting i's info
3. Upgrade info they carry by doing the MAX (info they send, info they carry)
Info_left[elem]+=Info_left[i]
Space[elem]=Space[elem]-Info_left[i]
IC[elem]=max(IC[elem],Info_left[elem])
IF Found=False:
if Found==False
1. Update info j sends by adding i's info
2. Update space left j has by substracting i's info
3. Upgrade info j carries by doing the MAX (info j sends, info j carries)
Info_left[j]+=Info_left[i]
Space[j]=Space[j]-Info_left[i]
IC[j]=max(IC[j],Info_left[j])
Update i's info to send to 0
Info_left[i]=0
Update i's battery by substracting its depth + 1 second
bat[i]=C[i][2]-(Depth[i]+1)
IF j's depth is minor than i's depth +1:
if Depth[j]<Depth[i]+1
1. Update its depth by adding i's depth +1
2. Update its battery by substracting i's depth + 1 seconds
Depth[j]=Depth[i]+1
bat[j]-=Depth[i]+1
FOR every parent of j:
for elem in Parents[j]
1. Update their depth by adding i's depth +1
2. Update their battery by substracting i's depth + 1 seconds
Depth[elem]=Depth[i]+1
bat[elem]-=Depth[i]+1
Update i's cost to get to the gateway
Cost_i[i]=(C[j][3]*D[j][0]*IC[i])+(D[i][j]*C[i][3]*IC[i])
FOR every son of i:
for elem in Sons[i]
Update their cost to get to the gateway
Cost_i[elem]=Cost_i[i]+(C[elem][3]*D[elem][i]*IC[elem])
FOR every element in the range of N:
for elem2 in range(N)
Update to -1 every element starting from i at the savings matrix
Sod[i][elem2]=-1
IF element is different from 0, j and i AND in the savings matrix it does no thave a -1 starting from i:
if elem2!=0 and elem2!=j and elem2!=i and Sod[elem2][i]!=-1


IF element is different from 0, j and i AND in the savings matrix it does no thave a -1 starting from i:
IF length of i's parents is equal to 1:
Update their element in the savings matrix going to i
ELSE:
else
Create a list with all i's parents and the parents of them
WHILE all members of the list are not visited AND not acabat:
Update the added cost for each of this nodes carrying i's info
Update their element in the savings matrix going to i
(*)

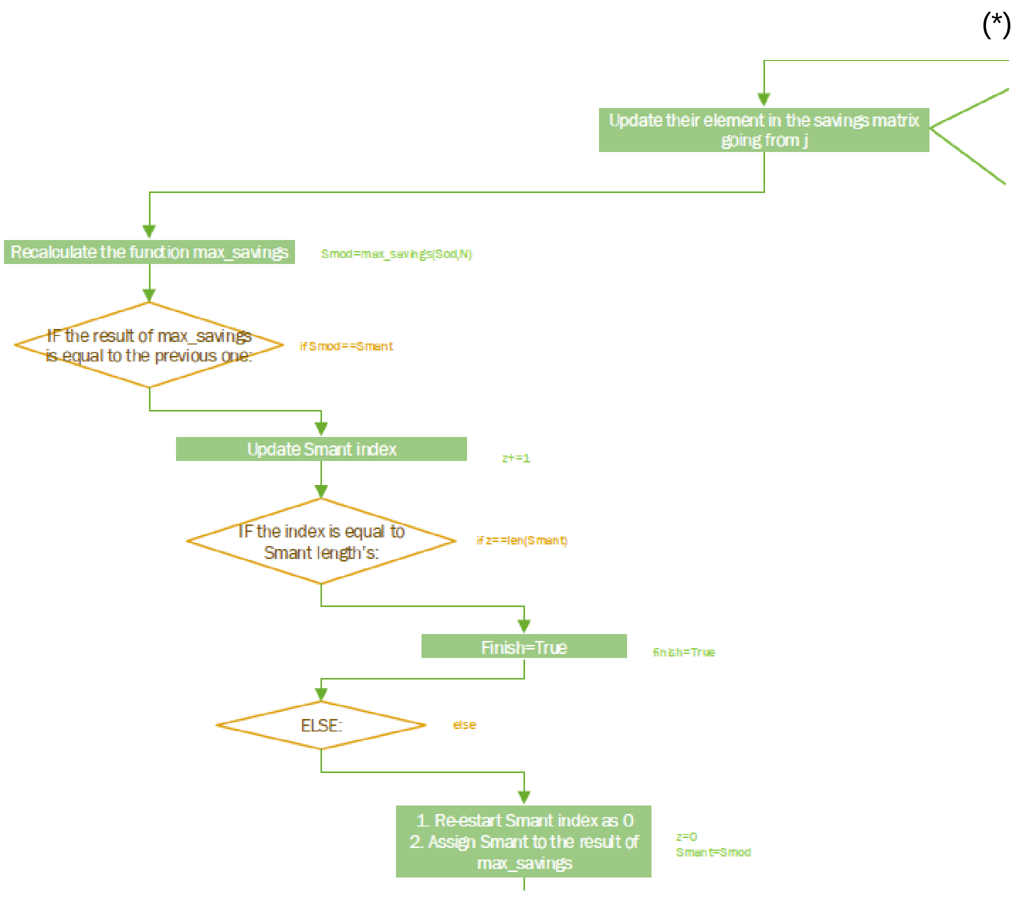
```
if elem2!=0 and elem2!=j and elem2!=i and Sod[elem2][i]!=-1
```

```
if len(Parents[i])==1
```

```
Sod[elem2][i]=round(Cost_i[elem2]-((C[i][3]*D[i][j]*IC[elem2])+(C[j][3]*D[j][0]*IC[elem2])+(C[elem2][3]*D[elem2][i]*IC[elem2])),4)
```

```
ll=[]
for elex in Parents[i]:
        if len(Parents[elex])>0:
                ll+=[[elex,len(Parents[elex])]]
```

```
order=[]
mini=10000000000
acabat=False
ind=-1
eles=0
while not acabat and eles<len(ll):
        if ll[eles][1]<mini:
                mini=ll[eles][1]
                ind=ll[eles][0]
                pp=eles
        eles+=1
        if ind!=-1:
                order+=[ind]
                ll[pp]=10000000000
        if len(ll)==0:
                acabat=True
io=1
SA=C[order[0]][3]*D[order[0]][0]*IC[elem2]
while io<(len(order)):
        SA+=C[order[io]][3]*D[order[io]][order[io-1]]*IC[elem2]
        io+=1
Sod[elem2][i]=round(Cost_i[elem2]-((C[i][3]*D[i][j]*IC[elem2])+(SA)+(C[elem2][3]*D[elem2][i]*IC[elem2])),4)
```


(*)
Update their element in the savings matrix going from j
Recalculate the function max_savings
IF the result of max_savings is equal to the previous one:
Update Smant index
IF the index is equal to Smant length's:
Finish=True
ELSE:
else
1. Re-estart Smant index as 0
2. Assign Smant to the result of max_savings

```
Smod=max_savings(Sod,N)
```

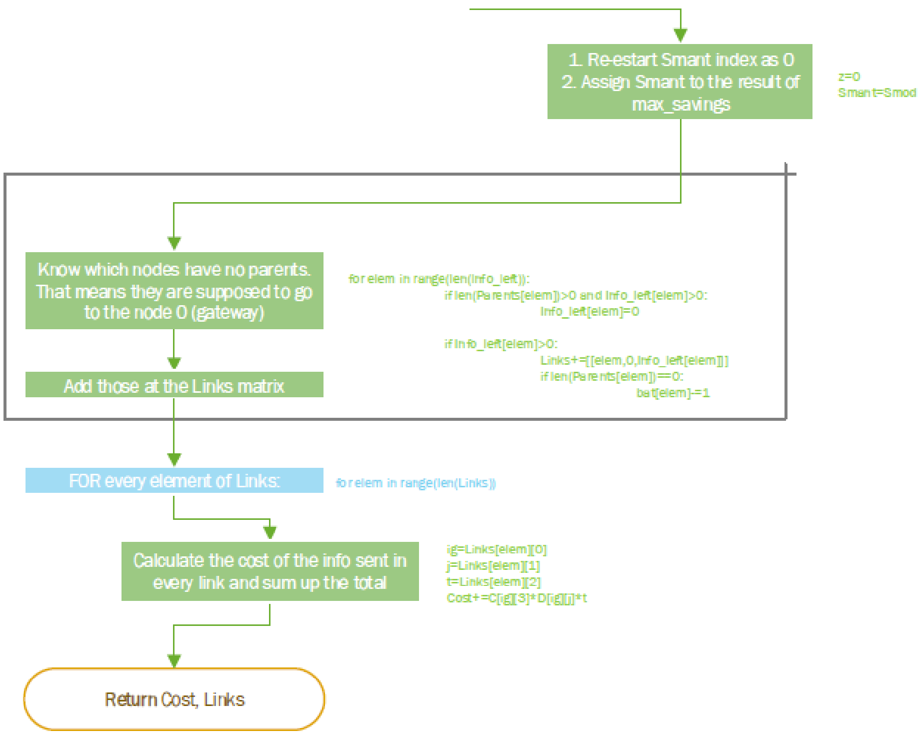
```
if Smod==Smant
```

```
z+=1
```

```
if z==len(Smant)
```

```
finish=True
```

```
z=0
Smant=Smod
```


1. Re-estart Smant index as 0
2. Assign Smant to the result of max_savings
Know which nodes have no parents. That means they are supposed to go to the node 0 (gateway)
Add those at the Links matrix
FOR every element of Links:
Calculate the cost of the info sent in every link and sum up the total
Return Cost, Links

```
z=0
Smant=Smod
```

```
for elem in range(len(Info_left)):
        if len(Parents[elem])>0 and Info_left[elem]>0:
                Info_left[elem]=0

        if Info_left[elem]>0:
                Links+=[[elem,0,Info_left[elem]]]
                if len(Parents[elem])==0:
                        bat[elem]-=1
```

```
for elem in range(len(Links))
```

```
ig=Links[elem][0]
j=Links[elem][1]
t=Links[elem][2]
Cost+=C[ig][3]*D[ig][j]*t
```

Figure 9 Block Diagram of the function algor from the algorithm_new.py file

Source: Own

5.3. The Linear Programming in CPLEX

In order to analyze the accuracy of the algorithm, a linear programming model has been built. The model has been solved using CPLEX [18].

Basically, what the program in CPLEX is, is typing the linear programming explained in the section 2.2 in the CPLEX language (the CPLEX program is attached in Annex 2). The advantage of it is that the program gives the exact solution for a given data set. However, an Excel file with all data (N, node's characteristics and distances matrix) is needed for every data set, and also the solution is written in the Excel file, which in the end it requires more time. But, what is more important is that the execution time is a lot higher than the solutions given by a programmed algorithm. That is why algorithms are widely used in many fields, in spite of their non-exact solutions. But the ones who are reliable are those who give solutions which are not far from the exact ones. To understand the huge difference in execution times, for a specific data set (Data set number 10 with 25 nodes):

- Execution time with the algorithm: 2,97 seconds
- Execution time with the CPLEX: 98h with still a GAP of 10% (the execution was stopped at this time)

Regarding the execution time CPLEX takes to give its solutions, a restricted amount of data sets has been done. In particular, only those with N (number of nodes) inferior to 30. In Chapter 4, the results of the comparison between the algorithm and the CPLEX are shown and a deep analysis of them is made.

5.4. Comparing the algorithm with the one in literature, the DRINA

In the next table (Figure 10), a general picture of the differences between the algorithm proposed in this thesis and the DRINA extracted from the literature are exposed:

DRINA	**Algorithm based on the savings concept**
Different types of data	One type of data
Dynamic network	Static network
Different categories of nodes	All nodes are same category

Figure 10 Principal differences between DRINA and the algorithm
Source: Own

Although both approaches try to solve the same situation, they have big differences which are worth to mention:

- In reality data gathered by different nodes can be very different one from each other, like DRINA considers. However, in the algorithm presented in this thesis only one standard type of data is considered, and it would be part of the future steps to include different types of data. That would mean that data from one node could only be considered to be sent to other nodes which gather the same data or simpler one, but never more complex data. In practical terms, would mean to add some constraints in the algorithm.

- A significant difference is the fact that the algorithm considers a static network while in reality data network is dynamic and at one fixed second there could be a number of sensors sending data different from the number that will be in the next second, like DRINA does. This would be also part of the future steps where a simulation of different types of events should be made, and decide for a single second, which is the optimal route that present sensor's data should follow. This does not change the fact of applying the savings concept that could be applied either way.

- Regarding sensor coverage, which is limited, sometimes more than one sensor captures the same data and so there are data redundancies. The DRINA algorithm considers this fact and creates a category of node which will gather this data and aggregate it to just forward the necessary one. The algorithm proposed in this thesis does not consider this fact as one of the premise was to work in a high-level approach and the sensor coverage limit was out of the scope. However, it also could be added together with the other considerations to a next version of the algorithm.

Despite these characteristics in which the DRINA algorithm analyzes further details that the algorithm presented in this thesis does not contemplate, there is an aspect worth considering and this is the complexity analysis. In the next table (Figure 11) the best and worst cases for both algorithms are shown:

	Best case	**Worst case**
Thesis Algorithm	n-1	n^3
DRINA	2n + m	$(2n + ((k-1)n - \sum_{i=2}^{k} \lvert U_i \rvert) + m)$

Figure 11 Best and worst cases for the algorithm and DRINA in terms of complexity

Source: Own & [9]

Where all parameters mean:

- n: number of nodes (for both algorithms)
- m: number of transmissions to create the cluster
- k: number of events
- $|U_i|$: the cardinality of the set of nodes outside the scope-limited flooding for the event i

In the case of the DRINA algorithm, it can be said that complexity can increase with the number of events ($k \geq n$), and so it can reach complexity n^2. In the case of our algorithm, complexity is fixed to n^3 disregarding the number of events (illustrated in Annex 4). Clearly, DRINA keeps being better in that sense too. However, as a further step of our algorithm, a deep analysis should be done in order to try to reduce the algorithm's complexity and reach complexity n^2.

6. Chapter 4: Results and validation

In this chapter, the results of the algorithm created will be presented together with their comparison between the base lines and the CPLEX program.

6.1. Comparing the algorithm with the base lines

The purpose of the thesis is to build the algorithm so to minimize the total sending cost of a given data network formed by a group of sensors and a gateway. For this, is not only important to build the algorithm but also to compare its results with some baselines to prove its feasibility and reliability.

In this thesis, there are two baselines already mentioned:

- Base line 1: reflects the initial case where all nodes send directly its data to the gateway, this means no optimization has been applied.
- Base line 2: this one works similar to the algorithm but with the constraint that at the most there are two nodes linked before reaching the gateway.

In this section, the thesis explains and compares the three solutions for the same given data set. The following graphics (Figures 12-21) show the differences in cost separated in groups of N's (from N=2 to N=100) and 50 data sets tested in each group.

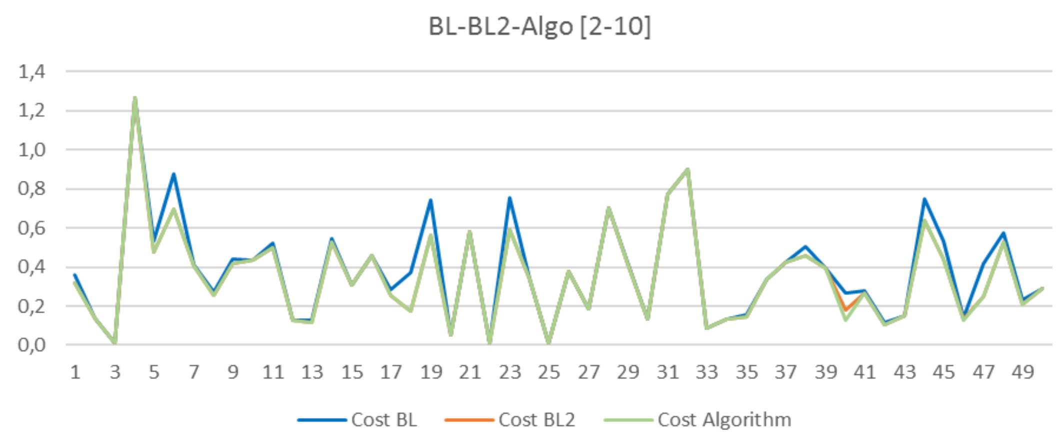


Figure 12 BL-BL2-Algorithm comparison for N [2-10]
Source: Own

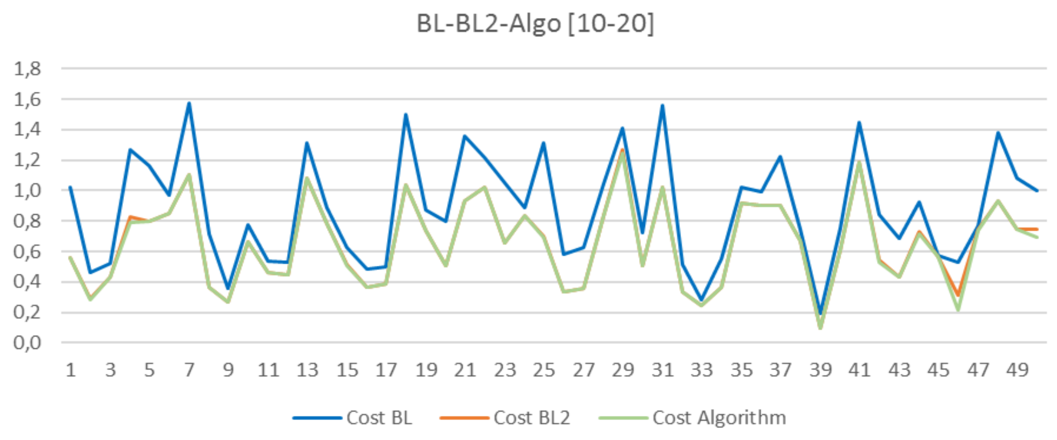


Figure 13 BL-BL2-Algorithm comparison for N [10-20]

Source: Own

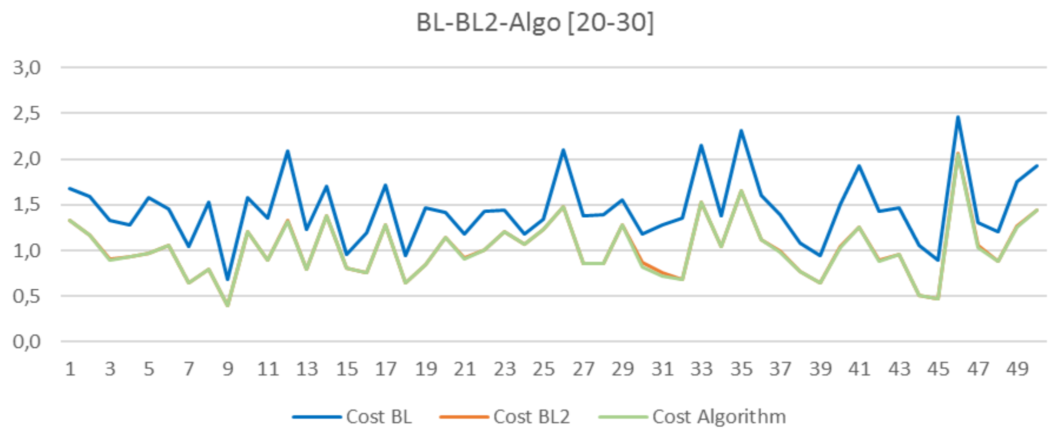


Figure 14 BL-BL2-Algorithm comparison for N [20-30]

Source: Own

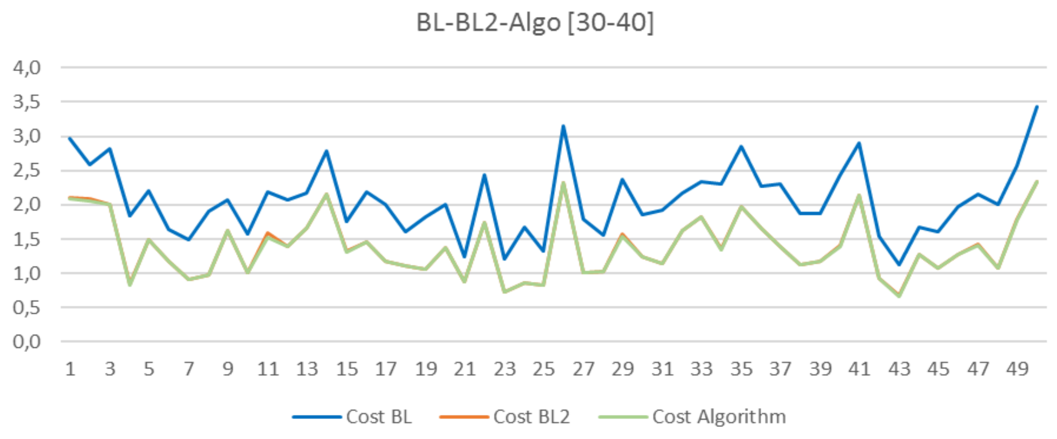


Figure 15 BL-BL2-Algorithm comparison for N [30-40]

Source: Own

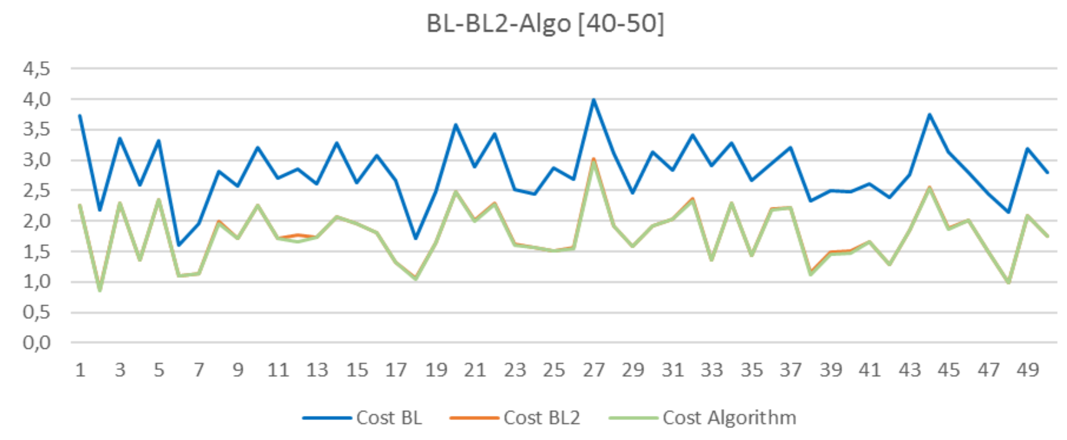


Figure 16 BL-BL2-Algorithm comparison for N [40-50]
Source: Own

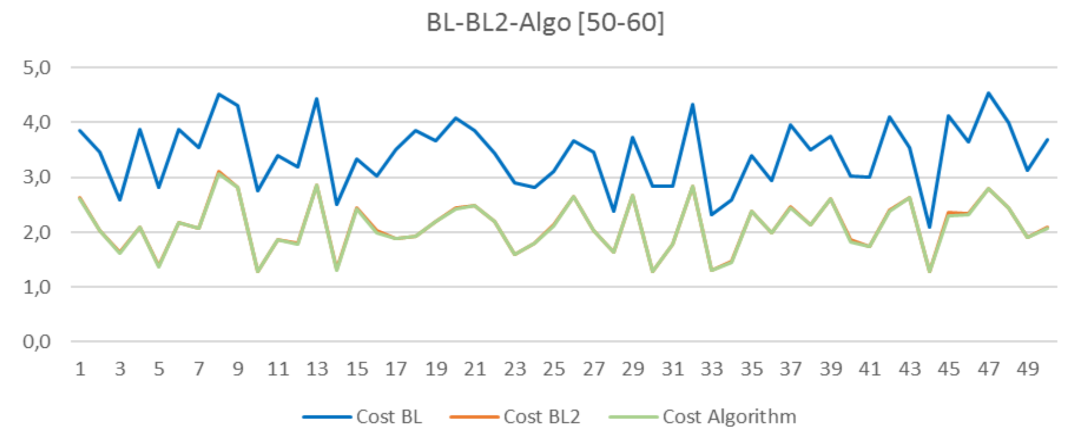


Figure 17 BL-BL2-Algorithm comparison for N [50-60]
Source: Own

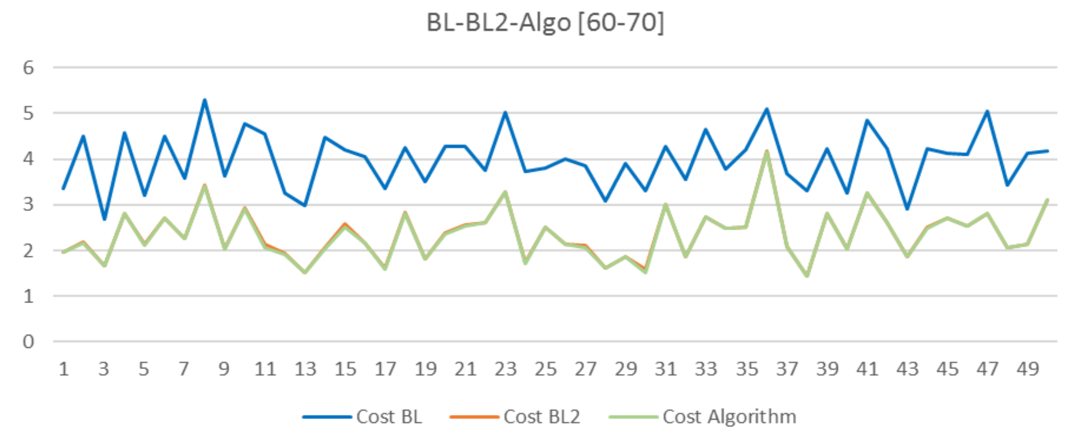


Figure 18 BL-BL2-Algorithm comparison for N [60-70]
Source: Own

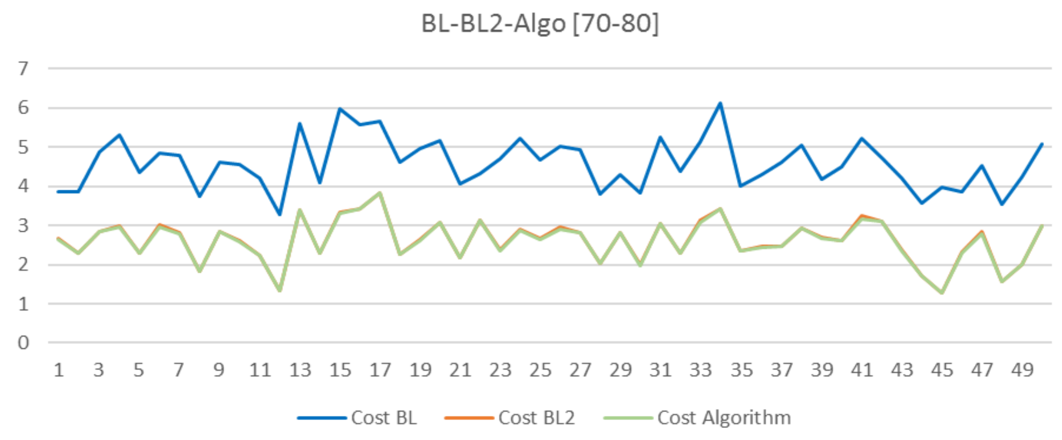


Figure 19 BL-BL2-Algorithm comparison for N [70-80]
Source: Own

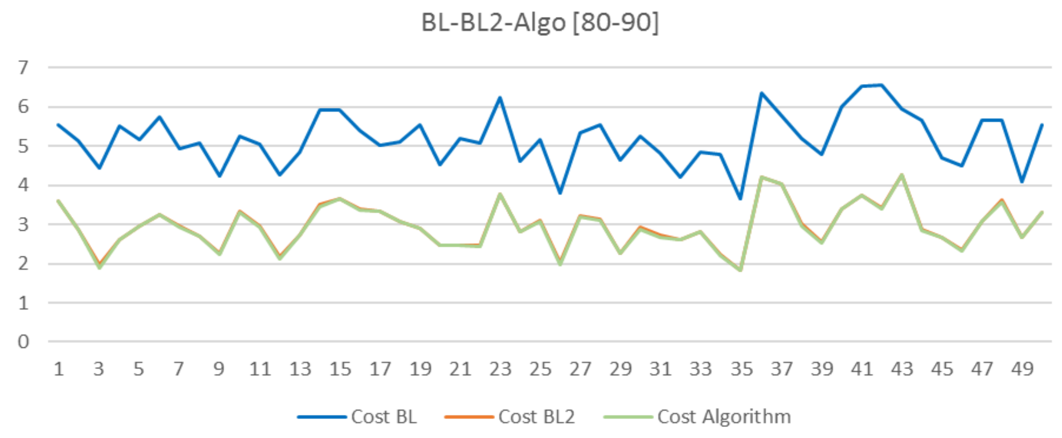


Figure 20 BL-BL2-Algorithm comparison for N [80-90]
Source: Own

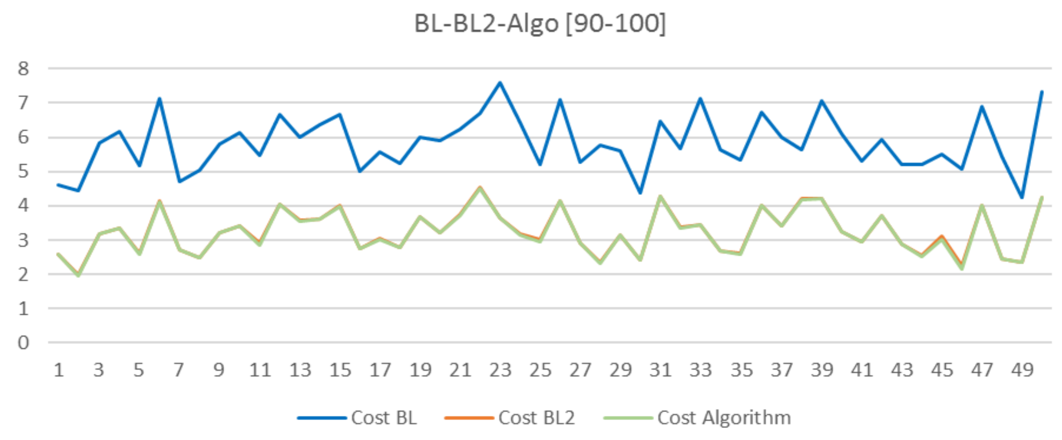


Figure 21 BL-BL2-Algorithm comparison for N [90-100]
Source: Own

From this total of 500 cases solved in the three ways, tt can be seen that as N grows in size, the differences between the solution of the base line (BL) and the algorithm are more plausible. Out of the 500, 475 times the algorithm was found to be better than the BL and the average percentages of improvement for each range of N are (Figure 22):

N	Algorithm vs BL
2-10	7,18%
10-20	26,15%
20-30	30,22%
30-40	34,05%
40-50	37,48%
50-60	39,00%
60-70	41,34%
70-80	43,59%
80-90	43,26%
90-100	44,90%

Figure 22 Average percentages of improvement Algo vs BL for different N

Source: Own

This difference is way far from the difference one can found between the second base line (BL2) and the algorithm that also does not have the same behavior as before. In this case (Figure 23), average percentages are low and don't grow as N grows like before. It is true that building long chains of nodes may not be cheap and sometimes the cheapest is not more than two nodes (like BL2). This time, the algorithm was better in 329 times.

N	Algorithm vs BL2
2-10	0,665%
10-20	1,093%
20-30	0,365%
30-40	0,45%
40-50	0,661%
50-60	0,439%
60-70	0,635%
70-80	0,506%
80-90	0,614%
90-100	0,585%

Figure 23 Average percentages of improvement Algo vs BL2 for different N

Source: Own

6.2. Comparing the algorithm with the CPLEX

Regarding the elevated amount of time the CPLEX program requires to solve data sets, data sets of more than 30 nodes were not capable of being solved with the CPLEX, as some of the ones of +20 nodes were solved days after. However, still some analyses can be made and they are explained in this section.

A number of 52 data sets were solved with CPLEX. From this total, 18 are between 2 and 10 nodes, 20 between 10 and 20 nodes and 14 between 20 and 30 nodes. From the 52 data sets, only in 13 of them the CPLEX solution was better than the solution of the algorithm. All these 13 cases are pictured in Annex 3.

In the next graphics, Figure 24 shows in percentage the improvements the algorithm has made versus both base lines and also the improvements the CPLEX has made versus the algorithm, for the 52 data sets. And Figure 26 shows the difference in cost for the 4 solutions. Also, the next table (Figure 25) show the average numbers in percentage, in this case just counting the cases when the first method is better than the second being compared.

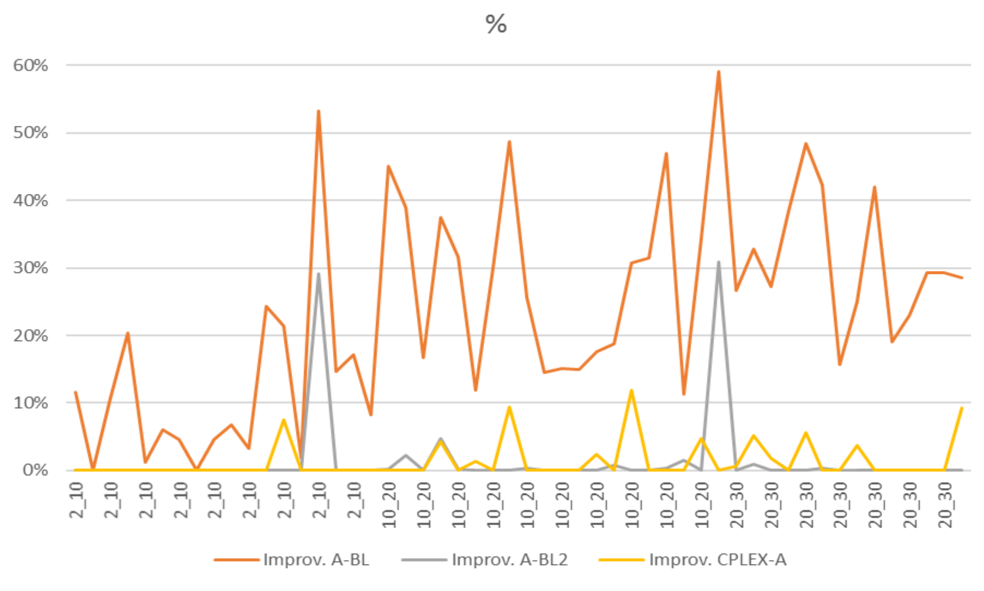


Figure 24 Percentages of improvement between different solutions

Source: Own

Methods	Avg. % of improvement
A vs BL	24,37
A vs BL2	4,79
CPLEX vs A	5,21

Figure 25 Average percentage of improvement for the cases the algorithm was better than BL and BL2 and the CPLEX better than the algorithm

Source: Own

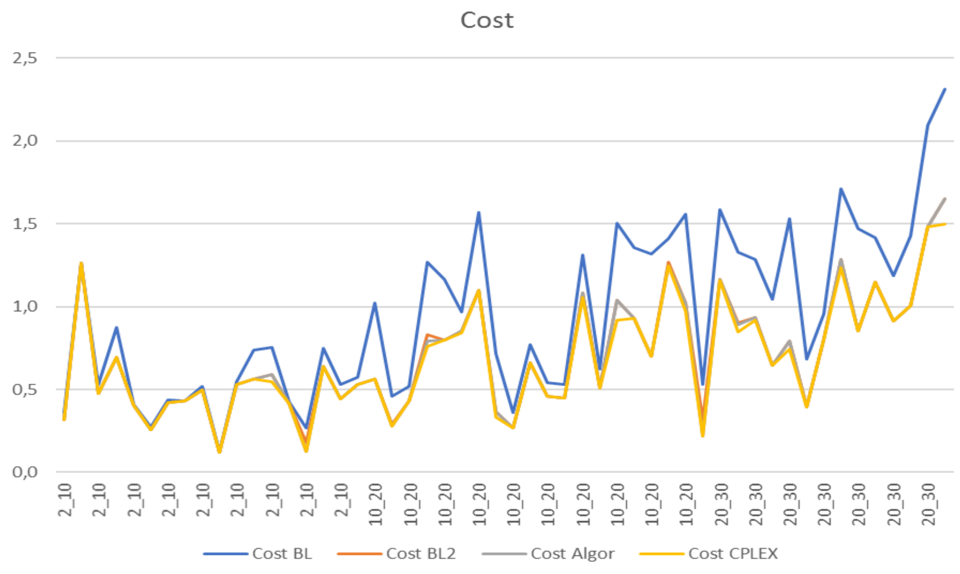


Figure 26 Differences in total cost between solutions

Source: Own

It can be supposed that with bigger number of nodes (bigger N's), the margins of improvement will increase, as the options of sending to other nodes also increase. It would be interesting to see if with bigger N's the average percentage of improvement CPLEX vs Algorithm would continue being higher than the one Algorithm vs BL2, or if both would increase at the same proportion.

Another thing to take into account, is the difference presented in topology solutions of the Algorithm and the CPLEX. The 13 cases in which CPLEX was found better than the Algorithm, do not present significant differences in topology. In the next figures (Figure 27) two clear examples are shown:

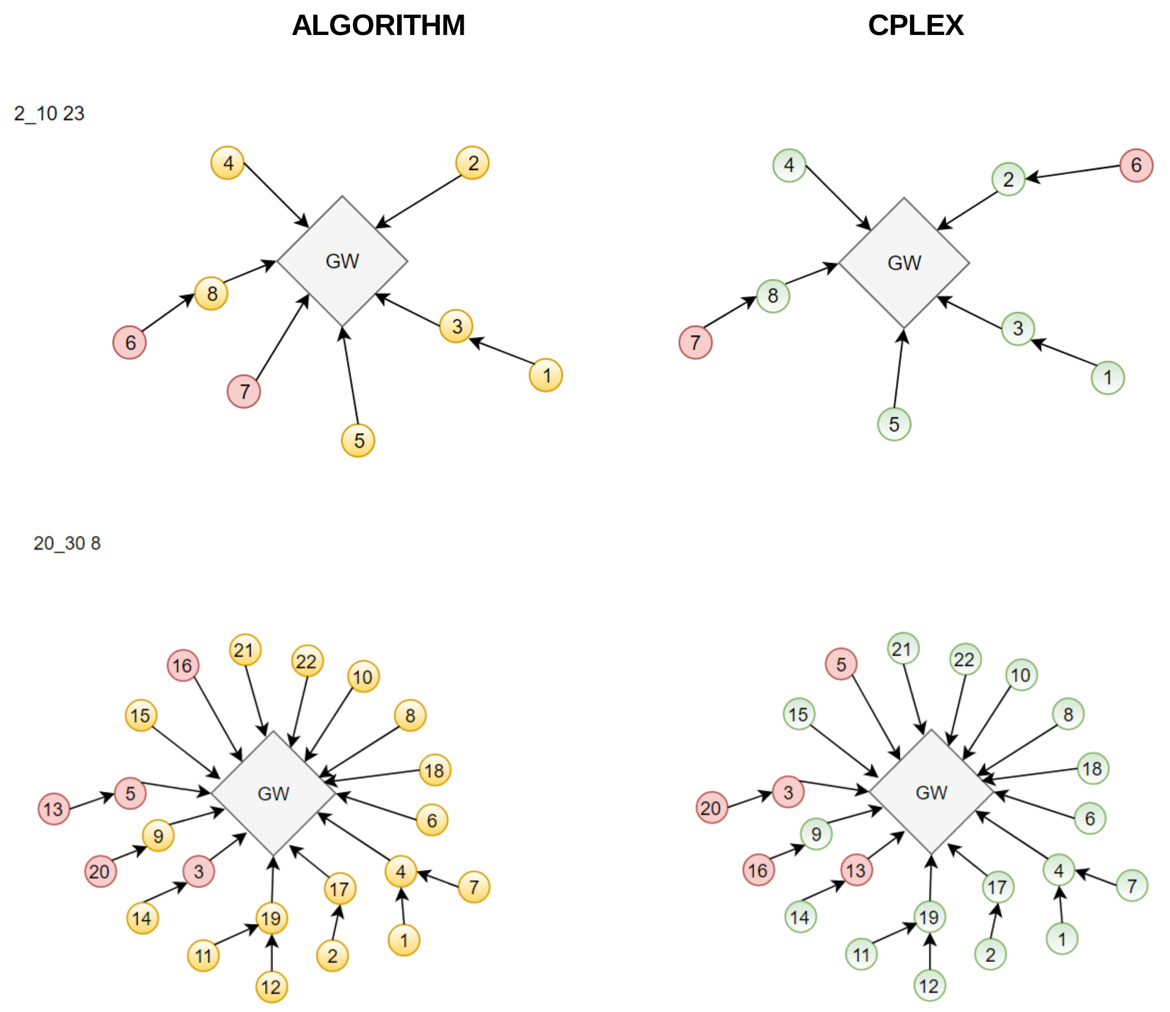


Figure 27 Two examples of the differences in topology between the CPLEX solution and the algorithm
Source: Own

The main differences found between both solutions, CPLEX vs Algorithm, are detailed just below. All cases mentioned can be found pictured in Annex 3.

- Same number of chains of 2 nodes, but a slight change between nodes (Cases: 20_30 8, 20_30 35)
- Reducing the number of chains of 2 nodes but creating one of 3 (Cases: 10_20 13, 10_20 18, 20_30 4, 20_30 17)
- Going from all chains of 2 nodes and one chain of 3 in Algorithm, to all chains of 2 nodes in CPLEX (Cases: 20_30 2, 20_30 3)
- Adding a new chain of 2 (Cases: 2_10 23, 10_20 4, 10_20 6, 10_20 8, 10_20 31)

In this sense, it cannot be assured that with bigger N’s topologies of the Algorithm and the CPLEX will be more plausible.

7. Environmental impact

It is known that sensors can do the job that humans cannot. With sensors, we can reach inaccessible, polluted or inhabitable spaces without putting in risk people’s lives. Programming sensors to optimize rubbish collection, reduce CO_2 emissions, control pollution, oil recycling, measure air and water quality, and other innumerous activities can help contribute the evolution of the environment.

There are plenty of use cases where the use of the IoT technology has helped improve human’s environmental footprint. One example is the reduction of 70% of water consumption in blueberry farms through an IoT based approach that a university in Chile put in practice. Another example is when the Boston Consulting Group announced that ICT-enables climate mitigation strategies could reduce global climate change a 16,5% by 2020 compared to today’s current efforts to fight against the climate change.

On the other hand, IoT also brings huge challenges for the environment, mainly referred to the e-waste, which is the waste of electrical and electronic equipment. As years goes by, more and more devices are manufactured and connected to the Internet, but, the hardware upgrades are also very frequently and so the number of e-waste is considerably important. In 2013, an amount of 53 million metric tons of e-waste were disposed worldwide and this number will do nothing but increase in the next years.

Since discarded electronic components out of IoT objects are the main source of e-waste these days, manufacturers of IoT equipment must consider the arising dangers that they cause after their useful life and start including smart manufacturing components, so to reduce their dangerous impact for the environment.

Another challenge IoT is facing is energy consumption as IoT networks require enormous data centers to store big amounts of data. Therefore, the energy consumption is massive and the resources needed to produce this amount of energy are a huge load for the environment and is affecting the whole energy sector. Furthermore, the resources and energy to manufacture all these new devices the industry of IoT will require is another source of energy consumption that is already having an important impact.

Last but not least, there is no doubt that the routing optimization of the data network in the IoT technology will contribute the industry to diminish its energy consumption.

8. Planning and costs

Next, there is a table (Figure 28) which explains the different costs related to the realization of the thesis:

Tasks	**Hours**
Literature review	50
Algorithm in Python	475
Programming	450
Getting results	25
CPLEX	85
Programming	45
Getting results	40
Analyses	70
Writing the thesis	250
Total	**930**

Figure 28 Tasks and hours per task

Source: Own

According to the average rate for a junior programmer, a cost of 30 $/h is stablished, and the next table (Figure 29) shows the total cost.

Tasks	**Amount**
Programmer hours (930 h * 30 $/h)	$27.900
Equipment	$1.000
Total	$28.900
TAX (20%)	$5.780
Total + TAX	**$34.680**

Figure 29 Total cost of the project

Source: Own

Conclusions

This Master Thesis was realized in École de Technologie Supérieure (ETS, Montréal) with the supervision of Professor Kim Nguyen, and reflects the last work of the Double Master's Degree in Industrial Engineering and Management Engineering at the Universitat Politècnica de Catalunya (UPC, Barelona). The thesis studies the routing optimization in the Internet of Things (IoT) network between the sensors and the gateway, which is not optimized in today's applications but will need to be in a near future due to the enormous amount of devices that will be connected to the Internet and the huge amount of energy consumption that fact will be carrying.

Firstly, the thesis introduces the problematic with the problem statement together with the objectives, research questions and hypotheses considered. Also, an introduction to the Internet of Things is done englobing its first days, today's presence in everyone's lives and the future prospects and challenges it will bring.

After the introduction, the literature review was needed in order to understand in which point is the problem and if there is new research and discoveries to solve it. In particular, a solid solution has been found and this is the DRINA [9] which is based on a cluster approach where nodes are divided into clusters and some nodes (cluster-heads) are elected to aggregate the data coming from others and send it to the sink node. In other words, there are different categories of nodes, also different types of data and the it considers a dynamic network. Three things that our algorithm did not take into account.

It is important to say that the problem is tackled in a high-level way, in which the network is considered to be a graph where the sensors are the nodes and their connections the links. Also, one of the basis of the project, is the use of the main idea of the Clarke & Wright's Savings Algorithm [8] which is used in the transportation and logistics field and the thesis takes its principal equation (the savings concept) and adapts it to the present case and so the equation is slightly different and so are the conditions to set the links between nodes (because the constraints to accomplish are different). The main constraints are related to the nodes' characteristics which include: data they gather, the total capacity of data they can gather, the lifetime of its battery, the costs of sending the data and the maximum distance it can be connected to other nodes.

The coding of the algorithm has been done with the Python language and a Linux system, and to test its feasibility, a comparison with three other methods has been made. These three are: a base line which reflects the absence of application of an algorithm (each node sending data directly to the gateway), a second base line which is similar to the actual algorithm but limits to a maximum of 2 nodes connected before reaching the gateway, and also the exact solution with the CPLEX program [18].

These comparisons were carried out in 500 data sets established randomly (of N's that went from 2 to 100 nodes). The user only had to decide an N (total number of nodes), this would then be set randomly between the number the user typed and 10 units less. Also, the user needed to decide how many times he wanted the algorithm to run for that group of N's. These questions were asked to the user through an *exe file* which contained the algorithm, but could be run in any operative system. Note that at worst the algorithm's solution was like the BL or the BL2 in each case. At the end, out of the 500 data sets, the algorithm was 475 times better than the BL with an average percentage of improvement of 36,54% out of the 475, and 329 times better than the BL2 with the same percentage this time of 0,91% out of the 391.

Another approach was made to compare the algorithm with the CPLEX solution, due to its large execution time for a given data set. The CPLEX program solves problems of linear programming (LP) and our problem was integer linear programming (ILP), so it could be programmed with the CPLEX software. However, the CPLEX returns the exact solution and that means that it needs to calculate the solution in all the iterations and then choose the one with the minimum cost in our case. For that, only those data sets with less than 30 nodes were capable of being solved. A total of 52 cases were solved and from these, just 13 of them the CPLEX solution was better than the algorithm with an average percentage of improvement of 5,21% out of the 13. It is worth to mention that differences in topology where not very vast when comparing both solutions, and only little changes were perceptible. This could change with larger N's, which would be feasible if execution time in CPLEX was lower.

For the approach being done, the algorithm is proved to be reliable enough and would be applicable to an infinite different IoT applications, which was the objective at the beginning. However, the author is conscious that this thesis has a lot of further steps, like integrating different types of data or different types of sensors like the algorithm DRINA does. This would make the project even more reliable as it would be nearer of what it can be found in the real world. Another thing to take into account is the algorithm's complexity which is susceptible of being reduced by improving the algorithm and putting it at the level of DRINA.

The personal conclusions the author extracts go from the realization of a project of these characteristics to the overcoming of all the difficulties presented during the way which made her fight for reaching the final objective and end being proud of all the work made. All the concepts learned through the realization of the thesis have been vital to understand the context and be able to design an algorithm of these characteristics, even if it is done in a high-level approach, but still applicable to a wide range of fields as mentioned.

To finish, the opportunity that has been given to the author to write a paper about the algorithm designed will be a remarkable experience that will determine for sure her professional career.

"I am already eager to see where the IoT industry will finally lead us".

Of gratitude for…

I would like to give special thanks to my tutor Mr. Kim NGUYEN, professor in Electrical Engineering Department (GE) at ETS, for his full predisposition in helping the author overcome all difficulties, for his guidance in every step of the way and his availability every time needed. Thank you Kim.

Also, I would like to thank professor Mr. Mohammed CHERIET, director of Synchromedia Laboratory and professor in Automated Production Engineering Department (GPA) at ETS, for his wise words in every conversation he and the author exchanged and his predisposition in helping her.

I would like to thank student Tuan Duong NGUYEN for his enthusiasm in helping the author solve the problems found during the realization of the thesis and his positive state of mind.

Last but not least, a big thank you to my whole family and Bernat Serra Deola for all the support given during the way.

Thank you all.

Bibliography

[1] Jesús CARRETERO, J. Daniel GARCÍA. *The internet of Things: connecting the world*. 2014

[2] IERC, Internet of Things European Research Cluster. *The Internet of Things, New Horizons*. 2012

[3] Journal of Network and Computer Applications. *Internet of Things: Smart things network and communication.* 2014

[4] Gilles PRIVAT. *Extending the Internet of Things.* 2012

[5] Katsuhiro NAITO. *A survey on the Internet of Things: Standards, Challenges and Future Prospects*. 2016

[6] Ovidiu VERMESAN, Peter FRIESS. *Digitising the industry. Internet of Things: Connecting the Physical, Digital and Virtual Worlds.* 2016

[7] Thomas ZACHARIAH et al. *The Internet of Things Has a Gateway Problem*. 2015

[8] Clarke & Wright. *Clarke & Wright's Savings Algorithm.* 1964

[9] Leandro APARECIDO VILLAS et al. *DRINA: A Lightweight and Reliable Routing Approach for In-Network Aggregation in Wireless Sensor Networks*. 2013

[10] Alain LOUCHEZ, Valerie THOMAS. *E-waste and the Internet of Things*. 2015 <https://itunews.itu.int/En/4850-E-waste-and-the-Internet-of-Things.note.aspx>

[11] *Environmental Impact of IoT*. <http://www.advancedmp.com/environmental-impact-of-iot/>

[12] *5 ways the IoT is helping the Environment*. 2016 https://iot.telefonica.com/blog/5-ways-the-iot-is-helping-the-environment

[13] Jong-Moon CHUNG. Yonsei University. *IoT Architecture*. 2017 <https://www.coursera.org/learn/iot-augmented-reality-technologies/lecture/8ZlnC/iot-architecture>

[14] *Internet of Things History*. <https://www.postscapes.com/internet-of-things-history/>

[15] Keith D. FOOTE. *A brief History of the Internet of Things*. 2016. <http://www.dataversity.net/brief-history-internet-things/>

[16] *IoT: Where does the data go?* https://www.wired.com/insights/2015/03/internet-things-data-go/

[17] Peter JONSSON et al. *Ericsson Mobility Report*. 2015

[18] IBM Corporation. *IBM ILOG CPLEX Optimization Studio Getting Started with CPLEX*

Annex 1: Python files

- **Ejec.py**

```
# -*- coding: utf-8 -*-
import networkx as nx
import matplotlib.pyplot as plt
#plt.use('Agg')
import json
import numpy as np
import os
import sys
import time
import random
import commands

import algorithm_new
import algorithm_new_2

print "We are running the algorithm X times: "
X=raw_input('')
print "We are running the algorithm for N: "
size=raw_input('')

fdata=open("AllData  N  "  +  str(max(2,int(size)-10))  +  "_"  +  str(int(size))  +
".txt",'w')
fsol=open("AllSolutions  N  "  +  str(max(2,int(size)-10))  +  "_"  +  str(int(size))  +
".txt",'w')
y=0
while y<int(X): #Everytime the loop is executed, I would like all random variables
to update
       initialtime=time.time()
       print "Archive " + str(y+1) + " \n"
       N=np.random.randint(max(2,int(size)-10),int(size))
       N2=N
       C=[[0,1000000000,1000000000,0,1000000000]]
       jj=1
       while jj<N:
              C0=np.random.randint(1,100)
              C1=C0+np.random.randint(1,100)
              C2=np.random.randint(1,10000)
              C3=np.random.random()*0.00001
              C4=np.random.randint(1,500)
              C+=[[C0,C1,C2,C3,C4]]
              jj+=1
       CC=C

       Info_left=[]
       for elemx in range(N):
              Info_left+=[C[elemx][0]]
```

```
        D=np.zeros((N,N))
        ii=0
        while ii<N:
                js=ii+1
                while js<N and js>ii:
                        D[ii][js]=np.random.randint(1,500)
                        D[js][ii]=D[ii][js]
                        js+=1
                ii+=1
        D=D.tolist()
        DD=D

        Cost_i=[]
        i=0
        while i<N:
                Cost_i+=[C[i][3]*D[i][0]*C[i][0]]
                i+=1

        S=[]
        i=0
        while i<N:
                j=0
                SS=[]
                while j<N:
                        sav=round(Cost_i[i]-
(C[i][3]*D[i][j]*C[i][0]+C[j][3]*D[j][0]*C[i][0]),4)
                        SS.append(sav)
                        c=i
                        p=j
                        j+=1
                S+=[SS]
                i+=1
        ix=0
        jt=0
        while ix<N and jt<N:
                S[ix][jt]=0
                ix+=1
                jt+=1
        iu=0
        ju=0
        while ju<N:
                S[iu][ju]=0
                ju+=1

        #With the variables defined, I run the algorithm
        #Sm=algorithm_new.max_savings(S,N)
        Cost,Links=algorithm_new.algor(D,C,N,S,Info_left,Cost_i)

        Info_left2=[]
        for elemx in range(N2):
                Info_left2+=[CC[elemx][0]]
        Cost_i2=[]
        i=0
        while i<N2:
                Cost_i2+=[CC[i][3]*DD[i][0]*CC[i][0]]
                i+=1

        S2=[]
        i=0
        while i<N2:
                j=0
                SS2=[]
                while j<N2:
                        sav=round(Cost_i2[i]-
(CC[i][3]*DD[i][j]*CC[i][0]+CC[j][3]*DD[j][0]*CC[i][0]),4)
                        SS2.append(sav)
                        c=i
                        p=j
                        j+=1
                S2+=[SS2]
                i+=1
        ix=0
        jt=0
        while ix<N2 and jt<N2:
                S2[ix][jt]=0
                ix+=1
                jt+=1
        iu=0
        ju=0
        while ju<N2:
                S2[iu][ju]=0
                ju+=1

        #Sm2=algorithm_new_2.max_savings2(S2,N2)
        Cost_2,Links_2=algorithm_new_2.algor_2(DD,CC,N2,S2,Info_left2,Cost_i2)

        if Cost_2<Cost:
                Cost=Cost_2
                Links=Links_2

        #I write in a file how the dataset looks like
        fo=open("DataSet" + str(y+1) + ".txt",'w')
        line="Data  set:  "  +  str(y+1)  +  "\n"  +  str(N)  +  "\n"  +  str(5)  +  "\n"  +
"{'C':" + str(C[1:]) + "}\n" + "{'D':" + str(D) + "}\n"
        fo.writelines(line)
        fo.close()
        finaltime=time.time()

        fdata.write(line)

        print "Execution time: " + str(finaltime-initialtime) + " seconds" + "\n"
        #Cost base line
        CostBL=0
        i=0
        while i<N:
                CostBL+=C[i][3]*D[i][0]*C[i][0]
                i+=1
        print "Cost BL: " + str(CostBL) + " $" + "\n"

        #Cost base line_2
        print "Cost BL2: " + str(Cost_2) + " $" + "\n"
        print "Cost Algorithm: " + str(Cost) + " $" + "\n"
        print "Improv BL2 vs Alg: " + str(((Cost-Cost_2)/Cost_2)*100) + " %" + "\n"

        #I write in a file the solution
        outfile=open("Solution" + str(y+1) + ".txt",'w')
        outline="Data set: " + str(y+1) + "\n" + "Execution time: " + str(finaltime-
initialtime) + " seconds" + "\n" + "Cost BL: " + str(CostBL) + " $" + "\n" + "Cost
BL2: " + str(Cost_2) + " $" + "\n" + "Cost Algorithm: " + str(Cost) + " $" + "\n" +
"Links: " + str(Links) + "\n" + "Links_2: " + str(Links_2) + "\n"
        outfile.writelines(outline)
        outfile.close()

        fsol.write(outline)

#       G=nx.DiGraph()
#       i=0
#       while i<N:
#               G.add_node(i)
#               i+=1
#       t=0
#       while t<len(Links):
#               G.add_edge(Links[t][0],Links[t][1])
#               t+=1
#       nx.draw_circular(G,    node_color='green',    node_size=800,    alpha=0.65,
with_labels=True)
#       plt.show()

        N=0
        C0=0
        C1=0
        C2=0
        C3=0
        C4=0
        Cost=0
        Cost_2=0
        Links=[]
        Links_2=[]

        y+=1

fdata.close()
fsol.close()
print "Execution finished"
```

- **Algorithm_new.py**

```
# -*- coding: utf-8 -*-

import networkx as nx
import matplotlib.pyplot as plt
import json
import numpy as np
import os
import sys
import time
import copy

def algor(D,C,N,S,Info_left,Cost_i):
       Space=[]
       bat=[]
       IC=[]
       i=0
       while i<N:
              Space+=[C[i][1]-C[i][0]]
              bat+=[C[i][2]]
              IC+=[C[i][0]]
              i+=1

       YY=XY(N)
       Links=[]
       lenLinks=0
       lenLinksAnt=0
       Depth=depth(N)
       Sons=parents_sons(N)
       Parents=parents_sons(N)
       Sod=S #Em servira per anar canviar la matriu destalvis (sense ordenar)
       #print S
       Sm=max_savings(S,N)
       #print Sm
       Smod=Sm  #Copio  la  matriu  destalvis  ordenada  perque  es  la  que  anire
modificant
       Smant=Sm #Em servira per saber quina era lanterior. En un inici poso que es
igual que Sm
       #print Smant
       z=0
       finish=False
       while not finish and lenLinks<N and z<len(Smant):
              i=Smant[z][0]
              j=Smant[z][1]
              if   D[i][j]<=C[i][4]   and   YY[i][j]==0   and   Info_left[i]>0   and
Space[j]>0:
                     if bat[j]>Depth[i]+1 or C[j][2]>Depth[i]+1:
                            #if
Cost_i[i]>=Cost_i[j]+(D[i][j]*C[i][3]*Info_left[i]):  #preguntarme si val la pena
en termes de cost, ajuntar
```

```
                            #if
Cost_i[i]>=(C[j][3]*D[j][0]*(Info_left[i]+Info_left[j]))+(D[i][j]*C[i][3]*Info_left
[i]):
                            if Info_left[i]<=Space[j] and len(Parents[i])<1:
                                   feas=0
                                   for elem in Parents[j]:
                                          if      Space[elem]>=Info_left[i]      and
bat[elem]>Depth[i]+1:
                                                 feas+=1

                                   if feas==len(Parents[j]):
                                          #NOU
                                          cu=0
                                          low_cost=1000000
                                          ex_i=[]
                                          Found=False

                                          for elem in Parents[j]:
                                                 if Space[elem]>=Info_left[i]:
                                                        ex_i+=[elem]

                                          for elem3 in ex_i:
                                                 if C[elem3][3]<low_cost:
                                                        low_cost=C[elem3][3]
                                                        k=elem3
                                                        Found=True

                                          if Found==True:
                                                 e=0
                                                 while e<len(Links):
                                                        if Links[e][0]==k:#**

       Links[e][2]+=Info_left[i] #**
                                                        if     Links[e][0]==j     and
Links[e][1]==k:

       Links[e][2]+=Info_left[i]


                                                        e+=1

                                          Links+=[[i,j,Info_left[i]]]
                                          lenLinks+=1
                                          YY[i][j]=1
                                          YY[j][i]=1
                                          if i not in Sons[j]:
                                                 Sons[j]+=[i] #Afegeixo I com a fill
de J
                                          if j not in Parents[i]:
                                                 Parents[i]+=[j]
                                          for elem in Sons[i]: #Els fills de I tambe
son els fills de J
```

```
                                                if elem not in Sons[j]:
                                                       Sons[j]+=[elem]
                                                if j not in Parents[elem]:
                                                       Parents[elem]+=[j]

                                         for elem in Parents[j]:
                                                if i not in Sons[elem]:
                                                       Sons[elem]+=[i]
                                                if elem not in Parents[i]:
                                                       Parents[i]+=[elem]       #Els
pares de J tambe son els pares de I

                                         for elem in Sons[i]:
                                                for elem2 in Parents[j]:
                                                       if     elem2     not     in
Parents[elem]:

       Parents[elem]+=[elem2]
                                                       if elem not in Sons[elem2]:
                                                              Sons[elem2]+=[elem]

                                         for elem in Parents[i]:
                                                for elem2 in Sons[j]:
                                                       YY[elem][elem2]=1

                                         for elem in Parents[j]:
                                                Info_left[elem]+=Info_left[i]
                                                Space[elem]=Space[elem]-
Info_left[i] #Actualitzo el espai de tots els pares

       IC[elem]=max(IC[elem],Info_left[elem])   #***

                                         #if len(Parents[j])==0:    #Si    ja    te
pares, ja esta afegida la info left al link amb els pares
                                         if Found==False:
                                                Info_left[j]+=Info_left[i]
                                                Space[j]=Space[j]-Info_left[i]
                                                IC[j]=max(IC[j],Info_left[j])

       #***


                                         Info_left[i]=0

                                         bat[i]=C[i][2]-(Depth[i]+1)    #Actualitzo
bateria
                                         if Depth[j]<Depth[i]+1: #En cas que amb la
unio de I, augmenti la prof de J, actualitzo la prof
                                                Depth[j]=Depth[i]+1
                                                bat[j]-=Depth[i]+1
                                                for elem in Parents[j]:
                                                       Depth[elem]=Depth[i]+1
                                                       bat[elem]-=Depth[i]+1
#Actualitzo bateria Parents[i]
```

```
                                         #Actualitzo cost de i per arribar al gw

       Cost_i[i]=(C[j][3]*D[j][0]*IC[i])+(D[i][j]*C[i][3]*IC[i]) #***
                                         #Si val la pena per i tambe ho valdra pels
seus fills
                                         for elem in Sons[i]:

       Cost_i[elem]=Cost_i[i]+(C[elem][3]*D[elem][i]*IC[elem]) #***

                                         #Recalculo savings del node i amb tota la
resta menys j i 0
                                         for elem2 in range(N):
                                                Sod[i][elem2]=-1
                                                if   elem2!=0   and   elem2!=j   and
elem2!=i and Sod[elem2][i]!=-1: #***
                                                       if len(Parents[i])==1:

       Sod[elem2][i]=round(Cost_i[elem2]-
((C[i][3]*D[i][j]*IC[elem2])+(C[j][3]*D[j][0]*IC[elem2])+(C[elem2][3]*D[elem2][i]*I
C[elem2])),4)
                                                       else:
                                                              ll=[]
                                                              for      elex      in
Parents[i]:
                                                                     if

len(Parents[elex])>0:

       ll+=[[elex,len(Parents[elex])]]
                                                              #print ll
                                                              order=[]
                                                              mini=10000000000
                                                              acabat=False
                                                              ind=-1
                                                              eles=0
                                                              while  not  acabat  and
eles<len(ll):
                                                                     if

ll[eles][1]<mini:

       mini=ll[eles][1]

       ind=ll[eles][0]
                                                                            pp=eles
                                                                     eles+=1
                                                                     if ind!=-1:

       order+=[ind]

       ll[pp]=10000000000
                                                                     if len(ll)==0:

       acabat=True
```

```
                                                                          #print order
                                                                          io=1

      SA=C[order[0]][3]*D[order[0]][0]*IC[elem2]
                                                                          while
io<(len(order)):

      SA+=C[order[io]][3]*D[order[io]][order[io-1]]*IC[elem2]
                                                                                io+=1


      Sod[elem2][i]=round(Cost_i[elem2]-
((C[i][3]*D[i][j]*IC[elem2])+(SA)+(C[elem2][3]*D[elem2][i]*IC[elem2])),4)

                                                       for elem3 in range(N):

                                                             if Sod[elem3][i]==0:
                                                                   Sod[elem3][i]=-1
                                                             if   elem3!=0   and   elem3!=i   and
elem3!=j and Sod[j][elem3]!=0 and Sod[j][elem3]!=-1:

      Sod[j][elem3]=round((C[j][3]*D[j][0]*IC[j])-
((C[elem3][3]*D[elem3][0]*IC[j])+(C[j][3]*D[j][elem3]*IC[j])),4)

            Smod=max_savings(Sod,N)
            #print Smod
            if Smod==Smant:
                  z+=1
                  if z==len(Smant):
                        finish=True

            else:
                  z=0
                  Smant=Smod
                  #print Smant
                  lenLinks+=1
                  #Sod=S
                  #print Sod

      for elem in range(len(Info_left)):
            if len(Parents[elem])>0 and Info_left[elem]>0:
                  Info_left[elem]=0
            if Info_left[elem]>0:
                  Links+=[[elem,0,Info_left[elem]]]
                  if len(Parents[elem])==0:
                        bat[elem]-=1

      i=1
      while i<N:
            Sons[0]+=[i]
            Parents[i]+=[0]
            i+=1
```

```
        Cost=0
        for elem in range(len(Links)):
                ig=Links[elem][0]
                j=Links[elem][1]
                t=Links[elem][2]
                Cost+=C[ig][3]*D[ig][j]*t

        return [Cost, Links]

def max_savings(SS,N):
        max_sav=0
        SX=copy.deepcopy(SS)
        Smm=[]
        while len(Smm)<N*N:
                i=0
                while i<len(SX):
                        j=0
                        while j<len(SX):
                                if SX[i][j]>max_sav:
                                        max_sav=SX[i][j]
                                        c=i
                                        p=j

                                j+=1
                        i+=1

                if max_sav==0:
                        break

                Smm+=[[c,p,max_sav]]
                SX[c][p]=-1
                #print "SS: " + str(SS)
                #print "SX: " + str(SX)
                max_sav=0
        return Smm

def depth(N):
        Depth=[]
        i=0
        while i<N:
                Depth+=[0]
                i+=1
        return Depth

def parents_sons(N):
        Sons=[]
        i=0
        while i<N:
                Sons+=[[]]
                i+=1
        return Sons
```

```
def XY(N): #Matriu de zeros NxN
       XY=[]
       XX=[]
       i=0
       while i<N:
              j=0
              while j<N:
                     XX+=[0]
                     j+=1
              XY+=[XX]
              XX=[]
              i+=1
       return XY
```

- **Algorithm_new_2.py**

```
# -*- coding: utf-8 -*-

import networkx as nx
import matplotlib.pyplot as plt
import json
import numpy as np
import os
import sys
import time

def algor_2(DD,CC,N2,S2,Info_left2,Cost_i2):
       Space2=[]
       bat2=[]
       i=0
       while i<N2:
              Space2+=[CC[i][1]-CC[i][0]]
              bat2+=[CC[i][2]]
              i+=1

       YY2=XY2(N2)
       Links_2=[]
       Depth2=depth2(N2)
       Sons2=parents_sons2(N2)
       Parents2=parents_sons2(N2)
       Sm2=max_savings2(S2,N2)
       z=0
       while z<len(Sm2):
              i=Sm2[z][0]
              j=Sm2[z][1]
              if  DD[i][j]<=CC[i][4]  and  YY2[i][j]==0  and  Info_left2[i]>0  and
Space2[j]>0 and len(Parents2[j])<1 and len(Sons2[i])<1:
                     if bat2[j]>Depth2[i]+1 or CC[j][2]>Depth2[i]+1:
                            if Info_left2[i]<=Space2[j] and len(Parents2[i])<1:
                                   feas2=0
                                   #for elem in Parents[j]:
                                   #      if      Space[elem]>=Info_left[i]      and
bat[elem]>Depth[i]+1:
                                   #             feas+=1

                                   if feas2==0: #len(Parents[j]):
#                                         #NOU
#                                         cu=0
#                                         low_cost=1000000
#                                         ex_i=[]
#                                         Found=False
#
#                                         for elem in Parents[j]:
#                                                if Space[elem]>=Info_left[i]:
#                                                       ex_i+=[elem]
#
```

```
#                                        for elem3 in ex_i:
#                                               if C[elem3][3]<low_cost:
#                                                      low_cost=C[elem3][3]
#                                                      k=elem3
#                                                      Found=True
#
#                                        if Found==True:
#                                               e=0
#                                               while e<len(Links_2):
#                                                      if Links_2[e][0]==k:#**
#
       Links_2[e][2]+=Info_left[i] #**
#                                                      if    Links_2[e][0]==j    and
Links_2[e][1]==k:
#
       Links_2[e][2]+=Info_left[i]
#
#                                                      e+=1

                                         Links_2+=[[i,j,Info_left2[i]]]
                                         YY2[i][j]=1
                                         YY2[j][i]=1
                                         if i not in Sons2[j]:
                                                Sons2[j]+=[i]  #Afegeixo  I  com  a
fill de J
                                         if j not in Parents2[i]:
                                                Parents2[i]+=[j]
                                         for  elem  in  Sons2[i]:  #Els  fills  de  I
tambe son els fills de J
                                                if elem not in Sons2[j]:
                                                       Sons2[j]+=[elem]
                                                if j not in Parents2[elem]:
                                                       Parents2[elem]+=[j]

                                         #for elem in Parents2[j]:
                                         #      if i not in Sons2[elem]:
                                         #             Sons2[elem]+=[i]
                                         #      if elem not in Parents[2i]:
                                         #             Parents2[i]+=[elem]      #Els
pares de J tambe son els pares de I

                                         for elem in Sons2[i]:
                                                for elem2 in Parents2[j]:
                                                       if      elem2      not      in
Parents2[elem]:

       Parents2[elem]+=[elem2]
                                                       if elem not in Sons2[elem2]:
                                                              Sons2[elem2]+=[elem]

                                         for elem in Parents2[i]:
                                                for elem2 in Sons2[j]:
                                                       YY2[elem][elem2]=1

                                         #for elem in Parents[j]:
                                         #      Info_left[elem]+=Info_left[i]
                                         #      Space[elem]=Space[elem]-
Info_left[i] #Actualitzo el espai de tots els pares

                                         #if len(Parents[j])==0:    #Si    ja    te
pares, ja esta afegida la info left al link amb els pares
#                                        if Found==False:
                                         Info_left2[j]+=Info_left2[i]
                                         Space2[j]=Space2[j]-Info_left2[i]

                                         Info_left2[i]=0

                                         bat2[i]=CC[i][2]-(Depth2[i]+1)  #Actualitzo
bateria
                                         if  Depth2[j]<Depth2[i]+1: #En  cas  que  amb
la unio de I, augmenti la prof de J, actualitzo la prof
                                                Depth2[j]=Depth2[i]+1
                                                bat2[j]-=Depth2[i]+1
                                                #for elem in Parents[j]:
                                                #      Depth[elem]=Depth[i]+1
                                                #      bat[elem]-=Depth[i]+1
#Actualitzo bateria Parents[i]

              z+=1

       for elem in range(len(Info_left2)):
              if len(Parents2[elem])>0 and Info_left2[elem]>0:
                     Info_left2[elem]=0
              if Info_left2[elem]>0:
                     Links_2+=[[elem,0,Info_left2[elem]]]
                     if len(Parents2[elem])==0:
                            bat2[elem]-=1

       i=1
       while i<N2:
              Sons2[0]+=[i]
              Parents2[i]+=[0]
              i+=1

       Cost_2=0
       for elem in range(len(Links_2)):
              ig=Links_2[elem][0]
              j=Links_2[elem][1]
              t=Links_2[elem][2]
              Cost_2+=CC[ig][3]*DD[ig][j]*t

       return [Cost_2,Links_2]
```

```
def max_savings2(SS2,N2):
       max_sav2=0
       SX2=SS2
       Smm2=[]
       while len(Smm2)<N2*N2:
              i=0
              while i<len(SX2):
                     j=0
                     while j<len(SX2):
                            if SX2[i][j]>max_sav2:
                                   max_sav2=SX2[i][j]
                                   c=i
                                   p=j
                            j+=1
                     i+=1

              if max_sav2==0:
                     break
              Smm2+=[[c,p,max_sav2]]
              SX2[c][p]=0
              max_sav2=0
       return Smm2

def depth2(N2):
       Depth2=[]
       i=0
       while i<N2:
              Depth2+=[0]
              i+=1
       return Depth2

def parents_sons2(N2):
       Sons2=[]
       i=0
       while i<N2:
              Sons2+=[[]]
              i+=1
       return Sons2

def XY2(N2): #Matriu de zeros NxN
       XY2=[]
       XX2=[]
       i=0
       while i<N2:
              j=0
              while j<N2:
                     XX2+=[0]
                     j+=1
              XY2+=[XX2]
              XX2=[]
              i+=1
       return XY2
```

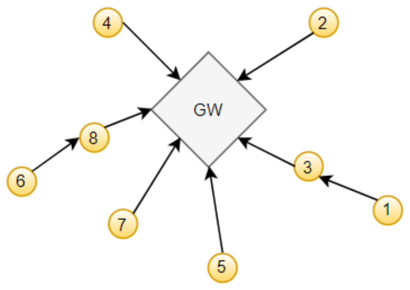
Annex 2: CPLEX program

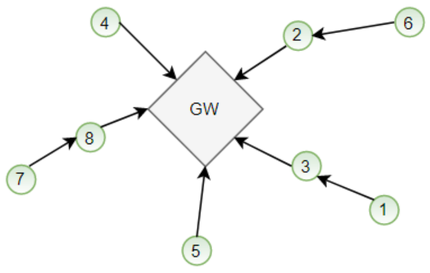
```
int N=...;
int K=...;
float C[1..N][1..K]=...; //characteristics
int D[1..N][1..N]=...; //distances

dvar int X[1..N][1..N]; //data sent from i to j
dvar boolean Y[1..N][1..N]; //1 if i sends to j
dvar boolean Z[1..N][1..N];
dvar float fo;
dvar float BL;
dvar int M[1..N][1..N];

minimize
  fo;

subject to
{
       fo==sum(i in 2..N, j in 1..N)(C[i][4]*X[i][j]*D[i][j]);
       BL==sum(i in 2..N)(C[i][4]*C[i][1]*D[i][1]);

       forall(i in 1..N, j in 1..N, l in 1..N)
         {
             X[i][j]>=0;

                Y[1][j]==0;
                X[1][j]==0;
                Z[1][j]==0;

                C[i][3]>=1+sum(l in 1..N)(Y[l][i]);
                X[i][i]==0;
                Y[i][i]==0;
                Z[i][i]==0;

                if (i!=j && l!=j && l!=i)
                 {
                      if (i!=1)
                      {
                             sum(i in 1..N)(C[i][1]*Z[i][j])<=C[j][2]-C[j][1];
                             sum(j in 1..N)(Y[i][j])==1;
                             X[i][j]>=(C[i][1]+sum(l in 2..N)(C[l][1]*Z[l][i]))*Y[i][j];
                             Y[i][j]<=Z[i][j];
                             Y[i][j]+Y[j][i]<=1;
                             Z[l][i]+Y[i][j]<=Z[l][j]+1;

                             if (j!=1)
                              {
                                     D[i][j]*Y[i][j]<=C[i][5];
                              }
                      }

                 }
         }

}
```

Annex 3: Comparing results: Algorithm vs CPLEX

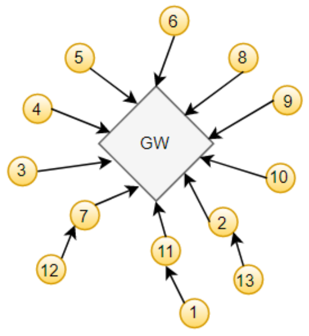
ALGORITHM	CPLEX

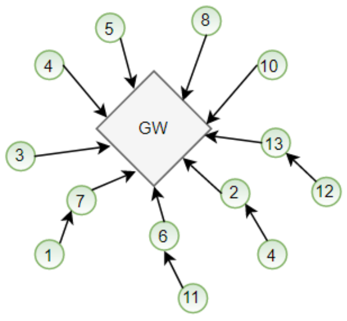
2_10 23



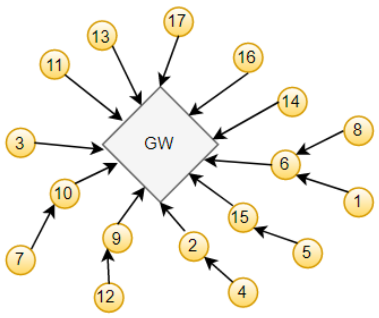


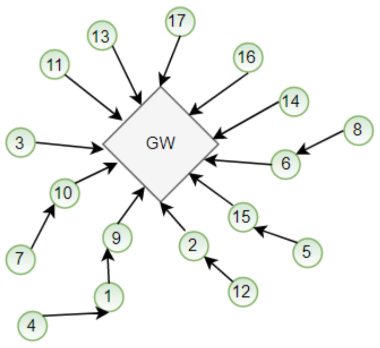
10_20 6



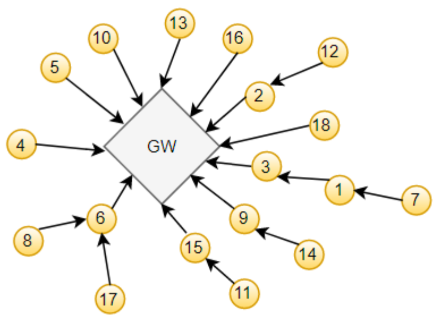


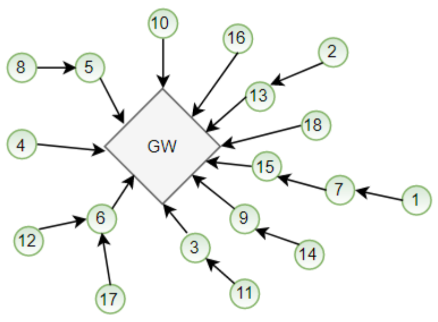
10_20 18



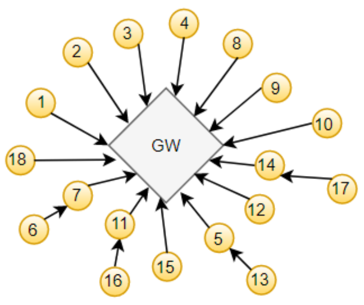


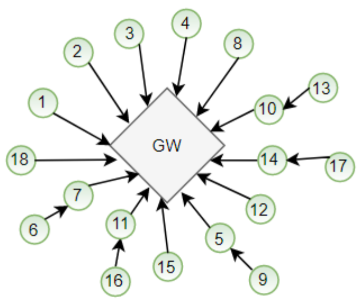
10_20 4



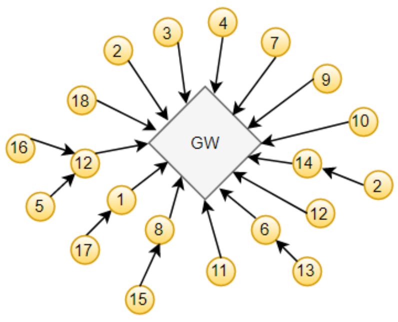


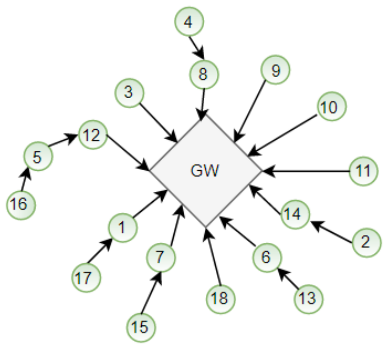
10_20 8



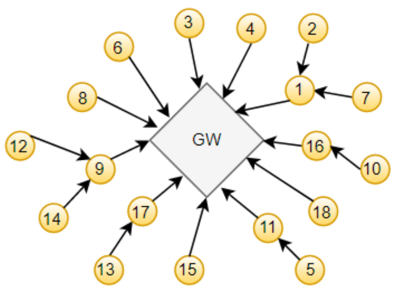


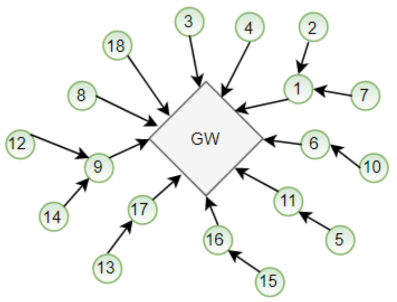
10_20 13



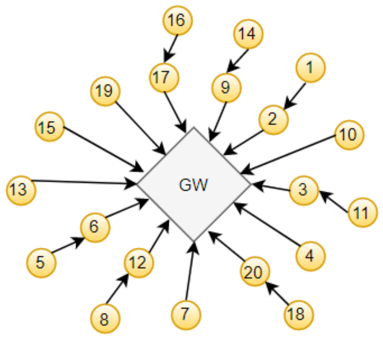


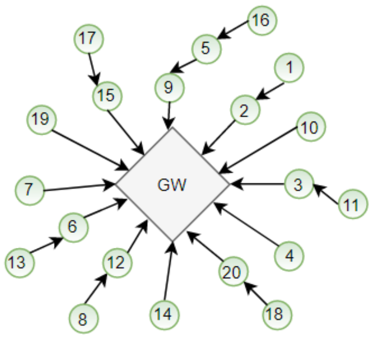
10_20 31



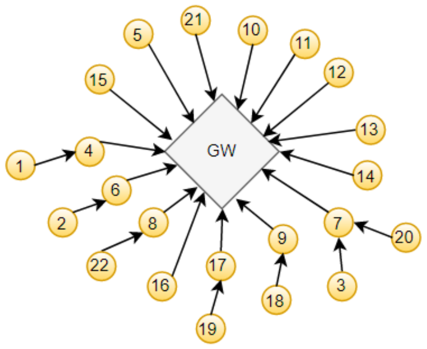


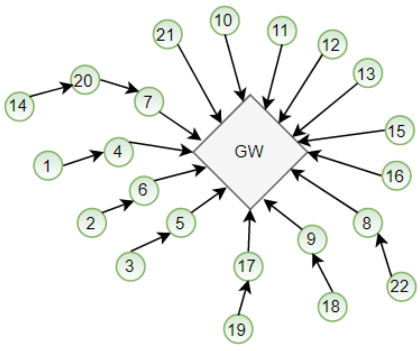
20_30 17



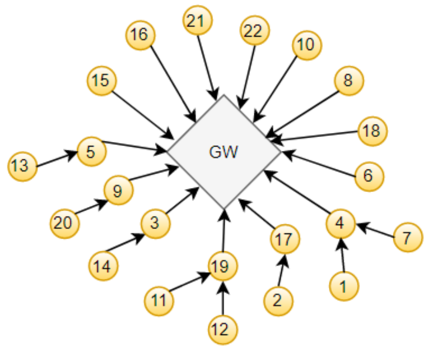


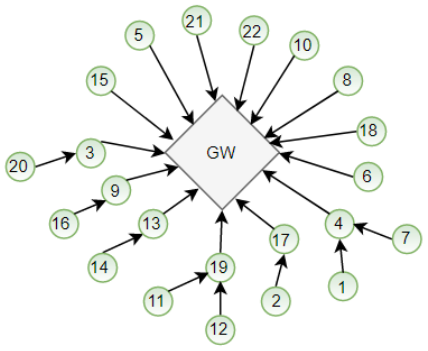
20_30 4



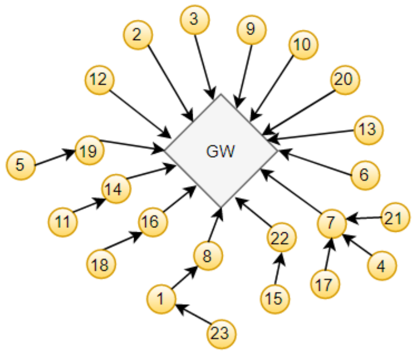


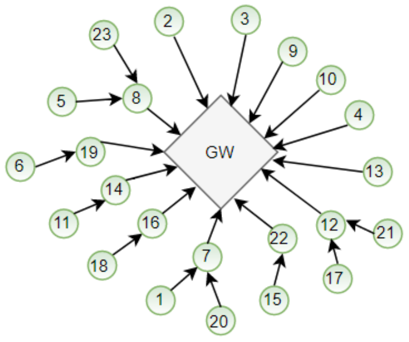
20_30 8



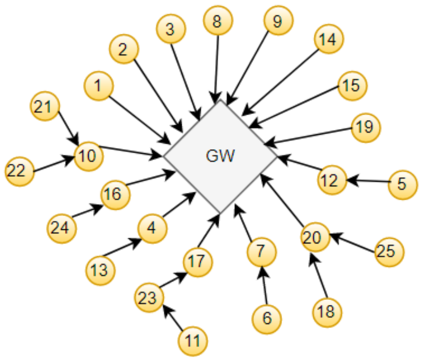


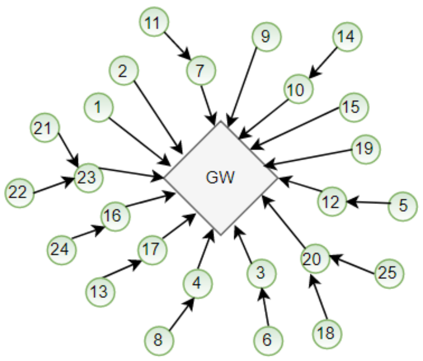
20_30 2



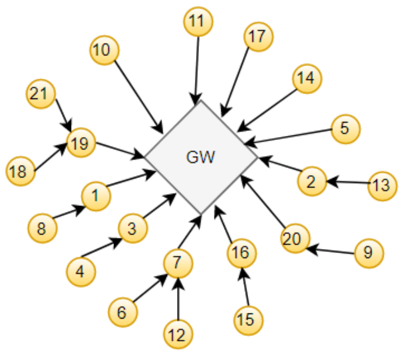


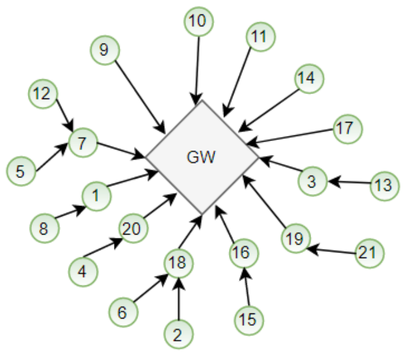
20_30 3





20_30 35


GW
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21


GW
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

Annex 4: Complexity analysis

Showing the WORST case

SUMMARY:

- *max_savings* function could be N^2 (we can use 1-dimensional array where each element is a set of the value and indices, so we only do N^2 iterations to find the max one).
- Then, total complexity of *algo* function would be N^3, since:

$$O\{(N-1)*[8*(N-2)+(N-1)+N+(N-2)^2+N*3*(N-2)+N^2]\}=O(N^3)$$

```
def algor(D,C,N,S,Info_left,Cost_i):
    Space=[]
    bat=[]
    IC=[]
    i=0
    while i<N:
        Space+=[C[i][1]-C[i][0]]
        bat+=[C[i][2]]
        IC+=[C[i][0]]
        i+=1
    YY=XY(N)
    Links=[]
    lenLinks=0
    lenLinksAnt=0
    Depth=depth(N)
    Sons=parents_sons(N)
    Parents=parents_sons(N)
    Sod=S #Em servira per anar canviar la matriu destalvis (sense ordenar)
    #print S
    Sm=max_savings(S,N)
    #print Sm
    Smod=Sm #Copio la matriu destalvis ordenada perque es la que anire modificant
    Smant=Sm #Em servira per saber quina era lanterior. En un inici poso que es igual que Sm
    #p
    z=0
    finish=False
    while not finish and lenLinks<N and z<len(Smant):                    N-1
        i=Smant[z][0]
        j=Smant[z][1]
        if D[i][j]<=C[i][4] and YY[i][j]==0 and Info_left[i]>0 and Space[j]>0:
            if bat[j]>Depth[i]+1 or C[j][2]>Depth[i]+1:
                if Info_left[i]<=Space[j] and len(Parents[i])<1:
                    feas=0
                    for elem in Parents[j]:
  N-2                   if Space[elem]>=Info_left[i] and bat[elem]>Depth[i]+1:
                            feas+=1

                    if feas==len(Parents[j]):
                        #NOU
                        cu=0
                        low_cost=1000000
                        ex_i=[]
                        Found=False
                        for elem in Parents[j]:
                            if Space[elem]>=Info_left[i]:
                                ex_i+=[elem]
  N-2in both            for elem3 in ex_i:
  cases                     if C[elem3][3]<low_cost:
                                low_cost=C[elem3][3]
                                k=elem3
                                Found=True
                if Found==True:
                    e=0
                    while e<len(Links):
N-1                     if Links[e][0]==k:
                            Links[e][2]+=Info_left[i]
                        if Links[e][0]==j and Links[e][1]==k:
                            Links[e][2]+=Info_left[i]
                        e+=1
                Links+=[[i,j,Info_left[i]]]
                lenLinks+=1
                YY[i][j]=1
                YY[j][i]=1
                if i not in Sons[j]:
                    Sons[j]+=[i]
                if j not in Parents[i]:
                    Parents[i]+=[j]
                for elem in Sons[i]:
                    if elem not in Sons[j]:
N-2                     Sons[j]+=[elem]
                    if j not in Parents[elem]:
                        Parents[elem]+=[j]

                for elem in Parents[j]:
                    if i not in Sons[elem]:
N-2                     Sons[elem]+=[i]
                    if elem not in Parents[i]:
                        Parents[i]+=[elem]

                for elem in Sons[i]:
                    for elem2 in Parents[j]:
                        if elem2 not in Parents[elem]:
(N-2) * (N-2)               Parents[elem]+=[elem2]
                        if elem not in Sons[elem2]:
                            Sons[elem2]+=[elem]

                for elem in Parents[i]:
N-2                 for elem2 in Sons[j]:
                        YY[elem][elem2]=1

                for elem in Parents[j]:
N-2                 Info_left[elem]+=Info_left[i]
                    Space[elem]=Space[elem]-Info_left[i]
                    IC[elem]=max(IC[elem],Info_left[elem])

                if Found==False:
                    Info_left[j]+=Info_left[i]
                    Space[j]=Space[j]-Info_left[i]
                    IC[j]=max(IC[j],Info_left[j])
                Info_left[i]=0
                bat[i]=C[i][2]-(Depth[i]+1)
                if Depth[j]<Depth[i]+1:
                    Depth[j]=Depth[i]+1
                    bat[j]-=Depth[i]+1
                    for elem in Parents[j]:
                        Depth[elem]=Depth[i]+1
                        bat[elem]-=Depth[i]+1
```

```
                                        #Actualitzo cost de i per arribar al gw
                                        Cost_i[i]=(C[j][3]*D[j][0]*IC[i])+(D[i][j]*C[i][3]*IC[i])
                                        #Si val la pena per i tambe ho valdra pels seus fills
                                        for elem in Sons[i]:
                N-2
          Cost_i[elem]=Cost_i[i]+(C[elem][3]*D[elem][i]*IC[elem])

                                        for elem2 in range(N):
                                              Sod[i][elem2]=-1
                        N
                                                    if elem2!=0 and elem2!=j and elem2!=i
and Sod[elem2][i]!=-1:
                                                    if len(Parents[i])==1:

          Sod[elem2][i]=round(Cost_i[elem2]-
((C[i][3]*D[i][j]*IC[elem2])+(C[j][3]*D[j][0]*IC[elem2])+(C[elem2][3]*D[elem2][i]*IC[elem2])),4)
                                                    else:
                                                          ll=[]
                                                          for elex in Parents[i]:
                                                                if
len(Parents[elex])>0:                N-2

          ll+=[[elex,len(Parents[elex])]]
                                                          #print ll
                                                          order=[]
                                                          mini=10000000000
                                                          acabat=False
                                                          ind=-1
                                                          eles=0
                                                          while not acabat and
eles<len(ll):
                                                                if ll[eles][1]<mini:
                                     N-2
          mini=ll[eles][1]

          ind=ll[eles][0]

                                                                      pp=eles
                                                                eles+=1
                                                                if ind!=-1:

          order+=[ind]
          ll[pp]=10000000000
                                                                if len(ll)==0:

          acabat=True
                                                          io=1
                                        SA=C[order[0]][3]*D[order[0]][0]*IC[elem2]
                                                          while io<(len(order)):
     N-2
          SA+=C[order[io]][3]*D[order[io]][order[io-1]]*IC[elem2]
                                                                io+=1
          Sod[elem2][i]=round(Cost_i[elem2]-((C[i][3]*D[i][j]*IC[elem2])+(SA)+(C[elem2][3]*D[elem2][i]*IC[elem2])),4)
```

```
                    for elem3 in range(N):
                        if Sod[elem3][i]==0:
                            Sod[elem3][i]=-1
N                       if elem3!=0 and elem3!=i and elem3!=j and
Sod[j][elem3]!=0 and Sod[j][elem3]!=-1:

        Sod[j][elem3]=round((C[j][3]*D[j][0]*IC[j])-((C[elem3][3]*D[elem3][0]*IC[j])+(C[j][3]*D[j][elem3]*IC[j])),4)

        Smod=max_savings(Sod,N)                Calling a function

        if Smod==Smant:
            z+=1
            if z==len(Smant):
                finish=True

        else:
            z=0
            Smant=Smod
            lenLinks+=1

    for elem in range(len(Info_left)):
        if len(Parents[elem])>0 and Info_left[elem]>0:
            Info_left[elem]=0
        if Info_left[elem]>0:
            Links+=[[elem,0,Info_left[elem]]]
            if len(Parents[elem])==0:
                bat[elem]-=1

    i=1
    while i<N:
        Sons[0]+=[i]
        Parents[i]+=[0]
        i+=1

    Cost=0
    for elem in range(len(Links)):
        ig=Links[elem][0]
        j=Links[elem][1]
        t=Links[elem][2]
        Cost+=C[ig][3]*D[ig][j]*t

    return [Cost, Links]
```

```
def max_savings(SS,N):
    max_sav=0
    SX=copy.deepcopy(SS)
    Smm=[]
    while len(Smm)<N*N:
        i=0
        while i<len(SX):
            j=0
            while j<len(SX):
                if SX[i][j]>max_sav:
                    max_sav=SX[i][j]
                    c=i
                    p=j
                j+=1
            i+=1
        if max_sav==0:
            break
        Smm+=[[c,p,max_sav]]
        SX[c][p]=-1
        #print "SS: " + str(SS)
        #print "SX: " + str(SX)
        max_sav=0

    return Smm
```

(N*N) * N * N