COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

INDUSTRIAL ENGINEERING DEGREE

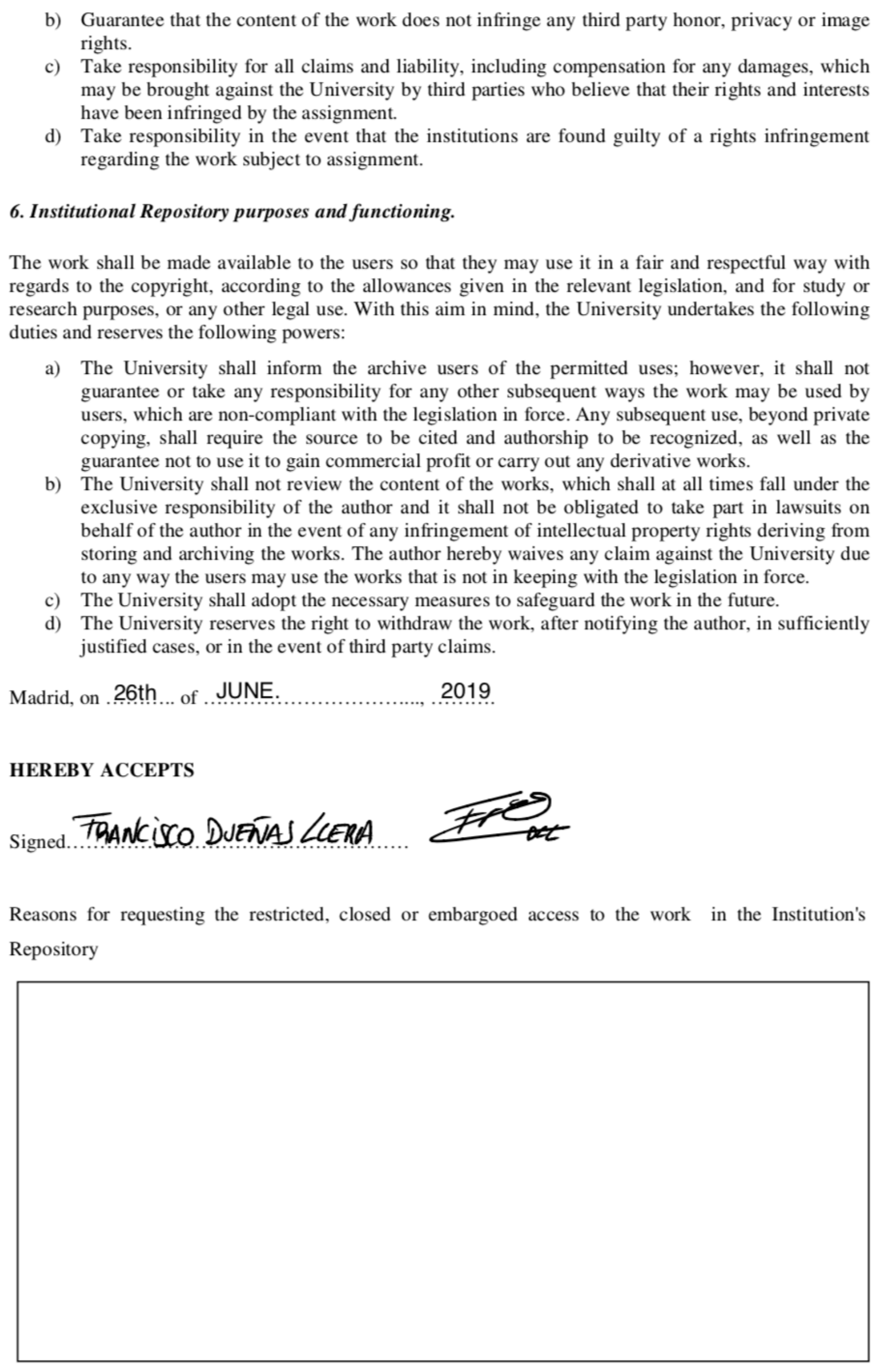
DEGREE'S FINAL PROJECT

"DULLE - FACILITY LAYOUT OPTIMIZATION AND EVALUATION ENGINE"

Author: Francisco Dueñas Llera

Director: Jeffrey Hermann

Madrid
June 2019



Madrid, on 26th of JUNE, 2019

HEREBY ACCEPTS

Signed FRANCISCO DUEÑAS LLERA [illegible]

Reasons for requesting the restricted, closed or embargoed access to the work in the Institution's Repository

I, hereby, declare that I am the only author of the project report with title:

DULLE - FACILITY LAYOUT OPTIMIZATION AND EVALUATION ENGINE

which has been submitted to ICAI School of Engineering of Comillas Pontifical University in the academic year 2018/19. This project is original, has not been submitted before for any other purpose and has not been copied from any other source either fully or partially. All information sources used have been rightly acknowledged.

[illegible]

Fdo.: Francisco Dueñas Llera Date: 25/06/2019

I authorize the submission of this project

PROJECT SUPERVISOR

Jeffrey W. Herrmann

Fdo.: Jeffrey W. HERRMANN Date: 26 June 2019

INDUSTRIAL ENGINEERING DEGREE

DEGREE'S FINAL PROJECT

"DULLE - FACILITY LAYOUT OPTIMIZATION AND EVALUATION ENGINE"

Author: Francisco Dueñas Llera

Director: Jeffrey Hermann

Madrid
June 2019

DULLE – FACILITY LAYOUT OPTIMIZATION AND EVALUATION ENGINE

Autor: Dueñas Llera, Francisco

Director: Hermann, Jeffrey

Entidad Colaboradora: ICAI – Universidad Pontificia de Comillas
University of Maryland – College Park

Optimizar el layout de una planta es una tarea necesaria y requerida en el día a día de cualquier fábrica. No importa tanto si la fábrica se esta construyendo desde cero o si la misma ya esta en pleno funcionamiento, en cualquier momento puede surgir una mejora o ampliación en la que haya que optimizar el espacio de trabajo. Para la mejor optimización en estas situaciones ha sido creado el programa DULLE. Programado en Python, DULLE es un programa interactivo y sencillo que resuelve problemas de optimización de planta a distintos niveles. Recibe del usuario dos archivos en formato csv en los que se incluye toda la información necesaria para resolver el problema. Esta información comprende las dimensiones de los departamentos y de la planta y los costes de manutención y transporte de los productos entre departamentos. Al ser un programa interactivo, toda la información se muestra en la diversas paginas del programa. Estas páginas están distribuidas como: "Basic Information", "Problem Statement", "Technical Results" y "Graphical Results" y la información incluye los datos recibidos del usario y los resultados del problema además de el gráfico de layout final.

Para obtener una evaluación coherente y exigente de la capacidad del programa para resolver los problemas se han resulto cuatro problemas distintos. Estos problemas incluyen diferentes escenarios tanto de distinto números de departamentos como de departamentos con distintos tamaños. Estos cuatro problemas han sido previamente resueltos por otros investigadores por lo que se podrán comparar resultados y se establecerá una correcta medida en cuanto a la capacidad de DULLE de resolver los distintos tipos de problemas.

DULLE – FACILITY LAYOUT OPTIMIZATION AND EVALUATION ENGINE

Author: Dueñas Llera, Francisco

Director: Hermann, Jeffrey

Collaborating Entities: ICAI – Universidad Pontificia de Comillas
University of Maryland – College Park

Facility layout optimization is present in every factory planning and construction. It does not matter if the factory is being built from scratch or if it is already built. Every day in any factory, layout optimization may be needed when installing new machines or when changing the supply chain process. For these situations, DULLE was created. It is a python-based program that solves optimization problems fast and easy. It receives from the user two different files in csv format including all the information needed for the problem such as department dimensions, factory dimensions and handling cost of products between departments. As the program is coded in python, it has a user interface easy to use and understand where all the information and results are displayed. All the information is displayed through four tabs arranged as: Basic Information, Problem Statement, Technical Results and Graphical Results.

For the evaluation of the program's performance four problems were solved by DULLE. These four problems represent different scenarios from low to high number of departments and fixed or variable widths and breaths for each department. Each problem results are displayed, explained and summarized in a table so that conclusions arise easily and are common for every reader. These problems have been already solved by other researches and so results are compared to measure the performance of DULLE's solving power.

INDEX

1- Abstract

Facility layout problems studied until now are mainly solved with tricky algorithms difficult to understand by an everyday user's knowledge. In this paper, "DULLE" is presented to solve both unequal and equal size departments problems to optimize any facility layout. Proven to be efficient and easy to use, it handles any type of problem from 2 departments to 20 or even more. "DULLE" has an easy to use user interface ready to be understood by everyone. It is built in python language including native modules such as gurobi and tkinter and gathers all the information from the needed information by two csv files submitted by the user. Alongside with the code used to run "DULLE", this paper presents all the information needed to understand how it was made and what are the basic principles to solve facility layout problems. Once again, results proof that "DULLE" works perfectly and that is recommended to use by any factory designer to achieve a first quick solution.

2- Introduction

From the Latin words *"Facility"*; space specifically built or used to improve the performance of any job and *"Layout"*; the way in which parts of something are arranged, arises a very well-known industrial management problem. The Facility Layout Problem (FLO) involves optimizing in a certain working space (*Facility*) the best possible arrangement of departments or facilities so that several constraints are satisfied, and one or more objectives are achieved. Having an optimal layout is one of the most important goals any company should look for when design its factory. Furthermore, some studies have come up that with an optimal layout, overall handling expenses could be reduced to 50% **[KOOP57].** Optimizing a layout is then proven to be very complex but crucial. It is an integration of many both dependent and independent factors such as worker's needs, pace and abilities, and it also integrates material raw and machinery among other parts of a production plant. The most important fact to demonstrate how complex FLP is, it is by realizing that it has been studied for decades now and there isn't a final solution yet.

"DULLE" is not meant to change the past seven decades of researching but it is considered as a very strong first and simple approach to such a complex problem. Every engineer and statistic specialist know how important is to have a good first optimal solution in order to achieve an even better one. "DULLE" is a python-based program built with the objective of solving any type of facility layout problem giving a lot of different input characteristics and information about the plant from the working pace or speed of the workers to the number of parts produced in the plant and the departments that involve the production of those parts. Making things simple while keeping a high efficiency and being as coherent as possible is the main philosophy throughout this project.

In order to present this project, "DULLE – FACILITY LAYOUT OPTIMIZATION AND EVALUATION ENGINE", I have organized the work in some sections that might help to understand deeper each and every part and function of this program. First, **Section 3**, provides a little background and past information about what has been studied in this topic as an introduction to the reader. As all the background I found was too big to be completely covered, the most important researches in recent history of the management industry alongside with the more influential papers for the creation of "DULLE" will be exposed here. **Section 4** introduces the optimization model formulation, explaining from beginning to end how DULLE mathematically solves and iterates between the possible problem solutions until the optimized layout is found. Then, **Section 5** introduces the software used to build DULLE as it is much more than a simple python-based program. Every package and software utility used will be widely and precisely explained so that even readers without programming knowledge will be able to follow the project design path. In **Section 6**, the mathematical method already mentioned in Section 4 will be coded and introduced to the python script, allowing the program to understand the optimization objectives and constraints. Here, the user will experience the difference on how we understand and compute math compared to how computers do. Later on, in **Section 7**, the program's appearance and how the user interacts with it will be explained. Both how it was made and how it works for the user will be widely studied and covered. Next, the overall results alongside with the studies of different scenarios will be covered in **Section 8**. Four different problems will be solved with DULLE and compared with other researches. This is the most important sections as DULLE comes to action and proves how efficient it is compared to other well-known historic algorithms and programs. In **Section 9**, summarized conclusions will be shown. There will be also an Appendix in **Section 10** at the bottom of the paper where the full code used to run DULLE will be shared so that anyone can check it and learn from it if interested.

3- Past Researches and Background

There is no doubt how long Facility Layout Problems have been around management-engineer's minds. One of the oldest papers I have read about FLP is **[KOOP57]** *and* it's interesting how back in the 50's they were already starting to think about how to optimize the layout of certain industries, facilities, or even more common spaces such as university's campus. But even more interesting is to analyze how more than six decades later the same problem is not solved yet. Furthermore, the problem has raised several levels of complexity. With the creation of the modern big companies and their corresponding massive factories and plants around the world, the facility layout problem seems now impossible to solve correctly. There are lots of factors, by the way, difficult to measure and counter, that may be crucial to change a facility layout for. Some "old fashioned" company's directors with who I have the opportunity to talk with feel like for them the best layout solution is the one that makes them see their factory more productive thanks to their experience. In reality, each one of them have their own optimized layout, but none of them really achieved the maximum optimization.

During these past centuries of researching, several mathematical methods to solve the Facility Layout Problem have been proposed. Some of the most important and popular methods will be explained below. This part of the section will be interesting to later on compare these methods with the simpler one proposed in this project.

- **Quadratic Assignment Problem (QAP):**

QAP was first proposed by **[KOOP57]** in the same paper cited above. In this paper, they considered QAP as a simplified version of a facility layout problem as they took for granted the number of locations and the departments cost flow and size were known a priori as it is done in this project.

It is though, one of the most popular methods for its simplicity but also for the many years that has been around achieving good results. It is based on a number *n* of locations and facilities. The objective is to assign *n* facilities to *n* potential locations in order to optimize the distance travelled between all the locations taking also into account the cost flow between each facility.

Each pair of facilities and locations have certain material flow and distance between them. It is also important to remember that this method only allows to work with fixed location sizes unlike DULLE which solves problems with un-equal area sizes.

The mathematical formulation for this method would be:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} c(i,j,k,l) * x(i,k) * x(j,l) \quad (3.1)$$

$$subject\ to:$$

$$\sum_{i=1}^{n} x(i,j) = 1 \quad for\ j = 1, \ldots, n \quad (3.2)$$

$$\sum_{j=1}^{n} x(i,j) = 1 \quad for\ i = 1, .., n \quad (3.3)$$

$$where:$$

$$c(i,j,k,l): cost\ for\ assigning\ facilities\ i\ and\ k\ to\ locations\ k\ and\ l\ respectively$$

$$x(i,j): 1\ if\ facility\ i\ is\ asgined\ to\ location\ k, 0\ otherwise$$

As the method only works assuming fixed locations or departments sizes, it optimizes all the facilities between a given grid configuration. It is considered to be a frequently usual requirement for any factory or plant to have different sizes of departments, so this method was not considered for this project. Later on, some other researches tried to improve the QAP to solve problems of unequal departments sizes. Some of those researches are ***[GILM63]*** and **[KUSI87].**

This last one proposed back in 1992 a new modified QAP where the first method's known locations were divided into smaller mini-locations of equal area. Then, each facility was assigned to a certain number of those mini-locations resulting in different facilities sizes and shapes. The feasibility of this method has been recently questioned for all the versions excluding the first proposed in **[KOOP57].**

- **Mixed Integer Problem (MIP):**

MIP method was first introduced in ***[MONT91]*** At first, it was a simple method that could solve a problem with up to 7 departments. The author of ***[MONT91]*** improved his previous method later on with several researches and modified versions. This proposed mixed-integer method was strongly influenced by the QAP method, especially in how the objective function was thought as it also minimizes the total material handling cost alongside with the total distance between departments. This type of objective function is also used in this project. He also proposed several linear and non-linear integer constraints to keep the departments within the facility range and to prevent overlapping between them.

Nowadays, there are hundreds of different mixed integer methods proposed to solve FLP and even different kind of location problems. Mixed integer methods are known as a mathematical solving method instead of a layout problem solving method. Furthermore, integer notation and basics will be used throughout the formulation of this project.

- **Genetic Algorithm (GA):**

It was around 1970 when John Henry Holland introduced genetic algorithms. Genetic algorithm is a metaheuristic inspired by human reproduction and development such us mutation, crossover and selection. Thanks to this humanistic approach, this method can give a good quality optimized value of any problem. Although its difficulty is much higher than any other method explained until now, the solutions given are expected to be better.

Explaining briefly the process, it consists of five main steps. In the first one, the initial population is declared. Each string of the initial population is a solution of the problem and it is called chromosome. To create a chromosome, each value is called gene. For instance, each chromosome is a solution to the problem, and it can be composed of either binary or continuous numbers.

In the second step, the fitness function is created. This function declares what would be the optimal solution and lets the algorithm evaluate how good is each chromosome compared to it. Thanks to this comparison, an evaluation of each chromosome is performed. This evaluation value is called fitness score and will be used in the next step.

Third, the selection process consists in selecting the two fittest chromosomes after evaluating the fitness function. These selected chromosomes pass their genes to the next generation. Both chosen chromosomes, called parents, exchange genes between themselves before reproducing. This means that from a random crossover point, genes of both chromosomes are exchanged, and a new offspring is created.

Lastly, the process terminates whenever the new offspring created is similar enough compared to the previous generation.

Genetic Algorithms are then for sure more accurate and precise than other methods, but they are also much more time-consuming and difficult. Extensive study is needed before deciding between genetic algorithms or any other simpler method.

Introducing now how DULLE works, it needs to be mentioned that the method or algorithm that I decided to use was a Mixed Integer Algorithm mainly due to its simplicity and easy-understanding. The main goal when creating DULLE was to achieve an efficient and easy for the user program and so this is the main reason to use a MIP algorithm. Furthermore, most of the optimization modules available in python such as gurobi, which is the one used for DULLE, handle better MIP problems as they are much simpler than other more complex methods.

4- Optimization Model Formulation

After having a first glance of what methods have been created and used during the past in Section 3, we can now talk deeper about DULLE's mathematical formulation. It is neither a pure QAP or MIP, but it is although much nearer to being a MIP problem. As it will be explained later on, I like to call it a "common sense method". This means that every mathematical operation has a reasonable explanation. The method was elaborated using both continuous and integer variables, and all the constraints are linear. We can then assume that the method is a linear and mixed integer problem in the background, but it wasn't built to be any of those. Instead, it just ended being a kind of mixed integer method thanks to the already mentioned "common sense" decisions. Every step taken when formulating the model had a reason or objective. Firstly, the objective function was stablished and from there on I started to create the variables needed for every constraint. All the constraints are ordered such that it represents the evolution and the steps made when creating the model.

The Optimization Model Formulation used in DULLE is shown below:

$$\boldsymbol{MIN} \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \sum_{\boldsymbol{j=1}}^{\boldsymbol{N}} \boldsymbol{C[i,j]} * \boldsymbol{Dij[i,j]} \quad \textbf{(4.1)}$$

$$subject\ to$$

$$\sum_{i=1}^{N} x[i] + \frac{W[i]}{2} \leq xUpper \quad \textbf{(4.2)}$$

$$\sum_{i=1}^{N} y[i] + \frac{B[i]}{2} \leq yUpper \quad \textbf{(4.3)}$$

$$\sum_{i=1}^{N} x[i] - \frac{W[i]}{2} \geq 0 \qquad (4.4)$$

$$\sum_{i=1}^{N} y[i] - \frac{B[i]}{2} \geq 0 \qquad (4.5)$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} Dij[i,j] = |x[i] - x[j]| + |y[i] - y[j]| \qquad (4.6)$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} s1[i,j] = 1 \quad only\ if \quad x[i] + \frac{W[i]}{2} \leq x[j] - \frac{W[j]}{2} \qquad (4.7)$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} s2[i,j] = 1 \quad only\ if \quad x[j] + \frac{W[j]}{2} \leq x[i] - \frac{W[i]}{2} \qquad (4.8)$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} s1[i,j] = 1 \quad only\ if \quad y[i] + \frac{B[i]}{2} \leq y[j] - \frac{B[j]}{2} \qquad (4.9)$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} s3[i,j] = 1 \quad only\ if \quad y[j] + \frac{B[j]}{2} \leq y[i] - \frac{B[i]}{2} \qquad (4.10)$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} s1[i,j] + s2[i,j] + s3[i,j] + s4[i,j] \geq 1 \qquad (4.11)$$

where:

$$x[i] \in \mathbb{R} \equiv X\ axis\ position\ of\ the\ center\ point\ of\ department\ i$$

$$y[i] \in \mathbb{R} \equiv Y\ axis\ position\ of\ the\ center\ point\ of\ department\ i$$

$$Dij[i] \in \mathbb{R} \equiv total\ \text{Manhattan}\ distance\ between\ department\ i\ and\ j$$

$$C[i,j] \in \mathbb{R} \equiv calculated\ handling\ cost\ between\ departments\ i\ and\ j$$

$$W[i] \in \mathbb{R} \equiv department's\ i\ width\ dimension$$

$$B[i] \in \mathbb{R} \equiv department's\ i\ breath\ dimension$$

$$xUpper \in \mathbb{R} \equiv X\ axis\ facility\ dimension$$

$$yUpper \in \mathbb{R} \equiv Y\ axis\ facility\ dimension$$

$$s1[i,j]\ \ ,s2[i,j]\ ,s3[i,j]\ ,s4[i,j]\ \equiv\ \mathbf{1}\ if\ each\ constraint\ is\ satisfied,\mathbf{0}\ otherwise$$

For a further understanding of all the constraints and how they were made, what thoughts and intentions were being held in the process of creating DULLE are exposed below:

- **(2)** & **(3)** & **(4)** & **(5)** ≡ **Facility's Plant Dimensions**

DULLE's software was created to perform in between a certain given facility's plant dimensions. These dimensions are given by the user through two csv files later on explained and mentioned as "User Input". Two of the variables that are saved from the User Input are "xUpper" and "yUpper". As their name say, they are the dimensions of the facility's plant in the "x" and "y" axis respectively.

Those four constrains force each department limit side to be within the plant region stablished by the user input. Both lateral sides as well as top and bottom limits have to be between the dimensions of the plant.

It is important to know the notation of both "x[i]" and "y[i]" variables. As it will be explained later on in the Section 7, the center of coordinates of the plant is the top left corner. Any movement to the right increases "x[i]" whereas any downwards movement increases "y[i]". As we can see both "x[i]" and "y[i]" can only be positive continuous numbers.

- **(6)** ≡ **Distance between Departments**

Measuring the distance between every department is one of the most important parts of the formulation. Here, in order to make the method as simplest as possible I decided to use the "Manhattan Distance" method which was first proposed in the XIX century in ***[MD].*** The

Manhattan Distance is calculated by summing the absolute of the differences between the x and y axis positions of each of the two points involved. More graphically it is calculated by:

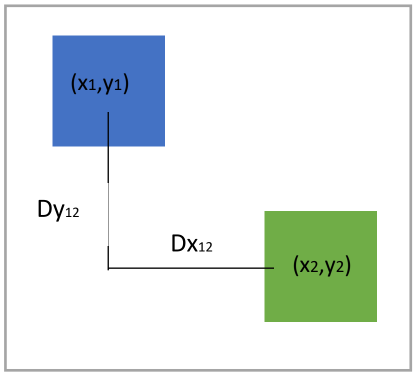


Figure 1: Manhattan Distance

$$Dij[1,2] = |x[1] - x[2]| + |y[1] - y[2]| \quad (4.12)$$

CONSTRAINT 5

The main reason to have chosen this distance calculation instead of the also known Euclidean Distance is to keep the model as simply as possible. Computing Euclidean distances involve nonlinear equations meanwhile the Manhattan Distance is a linear calculation.

- **(7), (8), (9), (10), (11)** ≡ **Non-overlapping Departments**

Without any doubt this is the trickier and more flexible constraint. The main objective is to make every department not overlap any of the others. Each "pixel" of the facility's plant can only be occupied by one department and by no more than one. If we don't incorporate this constraint in our program the solution is always going to lead us to set all the departments in the same point, as the distance calculated would be zero and the objective would be then minimized.

There are several options to formulate this constraint but many of the already proposed in other papers and researchers involve a nonlinear operation. Here in DULLE's formulation is presented one of the most effective and also simplest ways to satisfy this constraint. The background theory of this implementation is to think about what possibilities every pair of departments have to place one apart from each other. Below are presented each of the 4 possibilities and their relation to the constraints.

- **CASE A**

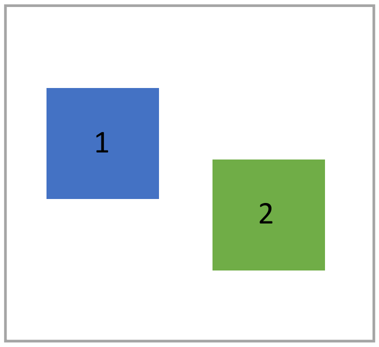


Figure 2: Departments layout for Case A

In this case, the right side of department 1 has to be in the left of the left side of the department 2

$$x[1] + \frac{1}{2}W[1] \leq x[2] - \frac{1}{2}W[2] \qquad (4.13)$$

CONSTRAINT 6

- **CASE B**

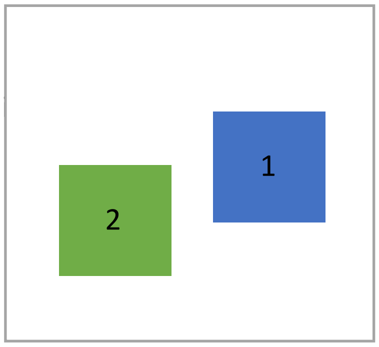


Figure 3: Departments layout for Case B

In this case, the right side of department 2 has to be in the left of the left side of the department 1

$$x[2] + \frac{1}{2}W[2] \leq x[1] - \frac{1}{2}W[1] \qquad (4.14)$$

CONSTRAINT 7

- **CASE C**

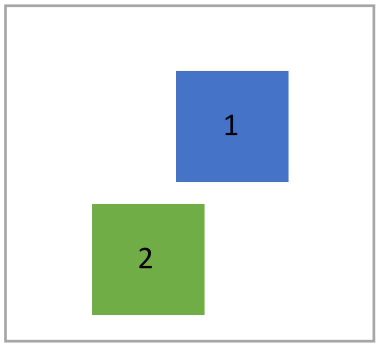


Figure 4: Departments layout for Case C

In this case, the bottom side of <u>department 1</u> has to be above the top side of <u>department 2</u>

$$y[1]+\frac{1}{2}B[1]\leq y[2]-\frac{1}{2}B[2] \qquad (4.15)$$

CONSTRAINT 8

(Remember "y" increases when we move downwards)

- **CASE D**

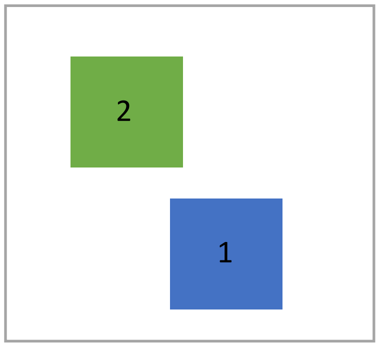


Figure 5: Departments layout for Case D

In this case, the bottom side of <u>department 2</u> has to be above the top side of <u>department 1</u>

$$y[2]+\frac{1}{2}B[2]\leq y[1]-\frac{1}{2}B[1] \qquad (4.16)$$

CONSTRAINT 9

(Remember "y" increases when we move downwards)

Another important part of the calculation process with an incredible influence into the overall method efficiency is the handling cost between departments. As said before, the user input is given by two csv files. From them, DULLE stores five variables to calculate the handling cost; number of trips needed between each pair of departments per year, cost per hour of workers associated with each department, department's handling inverse speed in hour per meter. After storing all these values for every pair of departments, DULLE calculates the handling cost between each department through the below equation:

$$\boldsymbol{Cost} = \frac{\#\ of\ trips\ needed\ between\ departments\ i\ and\ j}{year} * \frac{\frac{\$\ workers}{hour} * \frac{hour\ of\ workers}{meter}}{trip} \qquad (4.17)$$

Like in every research and design project, some assumptions had to be made in order to keep the problem simple and the calculations not too complicated. Although while creating DULLE there weren't a lot of assumptions needed, there are a couple worth mentioning.

When calculating the handling cost, theoretically, we would have to multiply it by the distance between departments "i" and "j". As DULLE's main objective is to place each department in a certain position so that the distance between departments alongside their cost makes the total cost minimum. When initializing the program at first (calculating the cost and input values) we don't have any department position and so then it is not possible to introduce the distance in any equation.

As one of the objectives of the project was the simplicity, it is assumed that every department has a fixed rotation. This means that the width and the breadth cannot be interchanged. This assumption wasn't found important as a lot of departments are mainly squares of different sizes, having their width and breath pretty similar. It also helps the user to have more influence in the program as he can choose from the first moment the rotation of each department and if after having an optimal solution, he decides to change the orientation, he would only need to change two parameters of the user input files and then run the program again.

5- Programing Structure and Techniques

DULLE's programming is based off of four main software or languages. The first and most important is python, known programming language among any software engineer. Gurobi and Tkinter are the most important modules installed inside the python code. They are used to solve the optimization model and create the graphical user interface of the program respectively. The last program used for DULLE's project is Excel, although it has nothing in common with python or any other programing language DULLE uses it to receive all the user input information needed to solve the problem. In this section these four programs used will be analyzed furtherly and the basics on how they were implemented to DULLE will be explained.

Starting with the most important part of DULLE, python, which is the roots, the basics of this project. Before getting into more complex programming material… What is python and how

does it work? Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Python is known for offering an endless list of additional libraries with which any type of task could be satisfied. In this category we have for example the two libraries used in DULLE, Gurobi and Tkinter. The most common uses for python are both data analysis and web development, but many other implementations such as optimization models like DULLE are also seen throughout the python community. Python was declared the most desirable language by the Stack's Overflow 2018 Developers Survey for its second year in a row.

As it will be explained in the next section with given examples of the code, the main structure of the python code in DULLE has four parts. These are parts are:

1- Creation and initialization of libraries, functions and graphical user interface.
2- Storing and displaying all the information inputted by the user.

3- Implementation of the Optimization Model.

4- Storing the results and displaying them alongside the final layout drawing.

Now each of these four main parts of DULLE's programming structure will be analyzed briefly. In the first part of the code, we need to declare and import all the libraries needed for our program. Libraries such as Tkinter and Gurobi have to be imported here, but also other libraries like csv file management have to be imported in order to be able to read and store files in .csv format, most commonly known as excel files. Later on, several functions are declared in order to create classes and store all the information inputted by the user into their correspondent variables. But, what is a class in the python world? As python is an object-oriented programming language almost everything in the code is oriented to a certain construct called classes. This construct lets us keep everything related to the object that belongs to. For example, in DULLE one class is the dimension of the departments and in it we can find the width and the breath of every department. This values of width and breath are not stored everywhere and cannot be called from anywhere in the program. They are stored inside the object "facility dimensions" and they have to be called from there.

In the second part, those functions declared before to store any information found in the csv files are called. After storing these values, everything is displayed in the graphical user interface (also known as GUI) so that the user can check that everything was received from the csv files and stored to the program properly. At this moment, if any value is wrong or the user wants to change any of the information sent, any of the two csv files can be updated. One everything is changed the user only needs to close the program and run it again.

The third part, which is where the optimization model is created will be fully explained in the next section, specially to note the difference and changes that had to be made to the optimization model proposed in Section 4 to convert it in both python and gurobi notation. It is important to assure this part is done properly as the computer needs everything to be really obvious and well-declared to be able to understand it. This part will be treated in the next section because first it is needed to know the basics of gurobi, the optimization model module solver, which will be explained in this section.

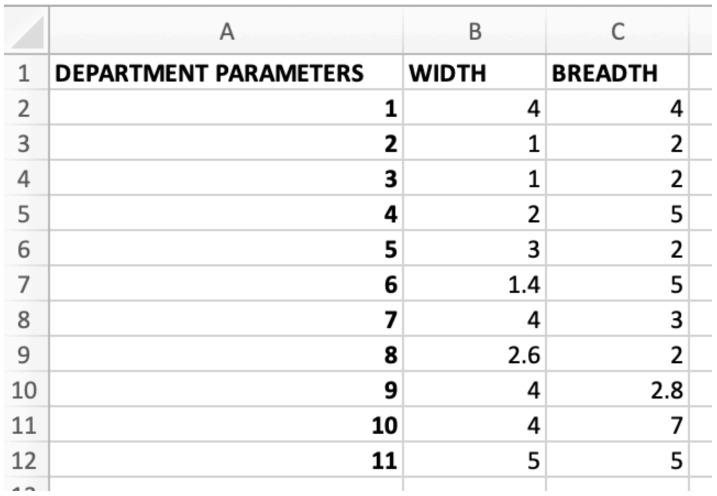
Last, the fourth part of the code could be separated into two subparts. In the first subpart, we would find the process of storing all the results to their corresponding variables. As gurobi does not return all the values of the variables straight away, we need to extract and store them from the model. After getting and storing all the results from gurobi we can now move on to the second subpart which is displaying the results. The results displayed in the GUI are the location of the departments, the distance between the departments, the final layout drawing and the total minimized cost of the layout.

Moving forward to the two modules used in DULLE, both will be explained and detailed in order to know how tkinter and gurobi were used to build up DULLE. It will also help readers have a better knowledge on how these modules work.

First, as it is used first, let's talk about tkinter. It is the module that creates the canvas and all the interface in the program thanks to the code later explained in Section 6 and provided in the *Appendix A*. Thanks to it, the introduction page was created, and a simple step-by-step guide was incorporated into this first page. It also allows to manage the results pages, creating the layout drawing in the last page and displaying the final distances between departments alongside the final total cost of the layout. Every tkinter implementation will be also explained in Section 6.

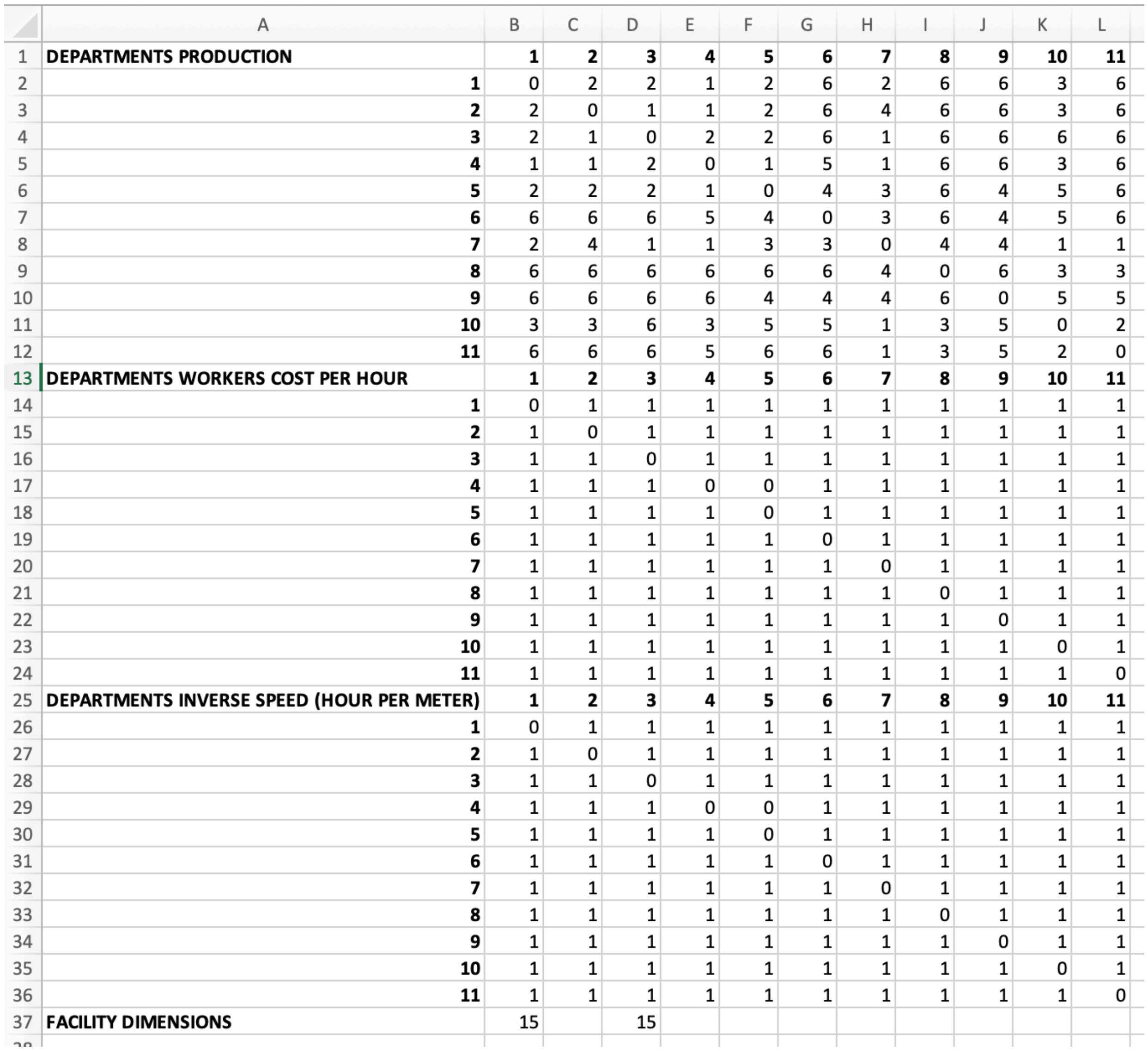
As well as tkinter, gurobi is a python module used in DULLE programming techniques. It is the mathematical formulation module that DULLE uses to solve the problem. As it will be shown in Section 6, DULLE has specific code lines created to let gurobi perform the optimization problem and then store in DULLE's variables the results that gurobi achieves.

Another important part of DULLE's problem solving sequence is the process of importing the user information. As it was mentioned a couple times by now, DULLE receives two csv files from the user with all the information. To let the user know how this information has to be placed in the csv so that DULLE works as expected there will be screenshots explaining how to arrange all the user information below. It is important to know that there has to be two different files with specific information in each. The first one is called "Department Parameters.csv" and includes the width and the breadth of every department in the problem. The second file is called "Facility Information.csv" and includes the number of trips between departments, the cost of the workers moving between departments, the inverse speed of these workers moving between departments and the facility dimensions. Below are showed both files so that the user can arrange his information just like in these examples.



	A	B	C
1	**DEPARTMENT PARAMETERS**	**WIDTH**	**BREADTH**
2	**1**	4	4
3	**2**	1	2
4	**3**	1	2
5	**4**	2	5
6	**5**	3	2
7	**6**	1.4	5
8	**7**	4	3
9	**8**	2.6	2
10	**9**	4	2.8
11	**10**	4	7
12	**11**	5	5

Figure 6: Department Parameters



	A	B	C	D	E	F	G	H	I	J	K	L
1	**DEPARTMENTS PRODUCTION**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
2	**1**	0	2	2	1	2	6	2	6	6	3	6
3	**2**	2	0	1	1	2	6	4	6	6	3	6
4	**3**	2	1	0	2	2	6	1	6	6	6	6
5	**4**	1	1	2	0	1	5	1	6	6	3	6
6	**5**	2	2	2	1	0	4	3	6	4	5	6
7	**6**	6	6	6	5	4	0	3	6	4	5	6
8	**7**	2	4	1	1	3	3	0	4	4	1	1
9	**8**	6	6	6	6	6	6	4	0	6	3	3
10	**9**	6	6	6	6	4	4	4	6	0	5	5
11	**10**	3	3	6	3	5	5	1	3	5	0	2
12	**11**	6	6	6	5	6	6	1	3	5	2	0
13	**DEPARTMENTS WORKERS COST PER HOUR**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
14	**1**	0	1	1	1	1	1	1	1	1	1	1
15	**2**	1	0	1	1	1	1	1	1	1	1	1
16	**3**	1	1	0	1	1	1	1	1	1	1	1
17	**4**	1	1	1	0	0	1	1	1	1	1	1
18	**5**	1	1	1	1	0	1	1	1	1	1	1
19	**6**	1	1	1	1	1	0	1	1	1	1	1
20	**7**	1	1	1	1	1	1	0	1	1	1	1
21	**8**	1	1	1	1	1	1	1	0	1	1	1
22	**9**	1	1	1	1	1	1	1	1	0	1	1
23	**10**	1	1	1	1	1	1	1	1	1	0	1
24	**11**	1	1	1	1	1	1	1	1	1	1	0
25	**DEPARTMENTS INVERSE SPEED (HOUR PER METER)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
26	**1**	0	1	1	1	1	1	1	1	1	1	1
27	**2**	1	0	1	1	1	1	1	1	1	1	1
28	**3**	1	1	0	1	1	1	1	1	1	1	1
29	**4**	1	1	1	0	0	1	1	1	1	1	1
30	**5**	1	1	1	1	0	1	1	1	1	1	1
31	**6**	1	1	1	1	1	0	1	1	1	1	1
32	**7**	1	1	1	1	1	1	0	1	1	1	1
33	**8**	1	1	1	1	1	1	1	0	1	1	1
34	**9**	1	1	1	1	1	1	1	1	0	1	1
35	**10**	1	1	1	1	1	1	1	1	1	0	1
36	**11**	1	1	1	1	1	1	1	1	1	1	0
37	**FACILITY DIMENSIONS**	15		15								

Figure 7: Facility Dimensions

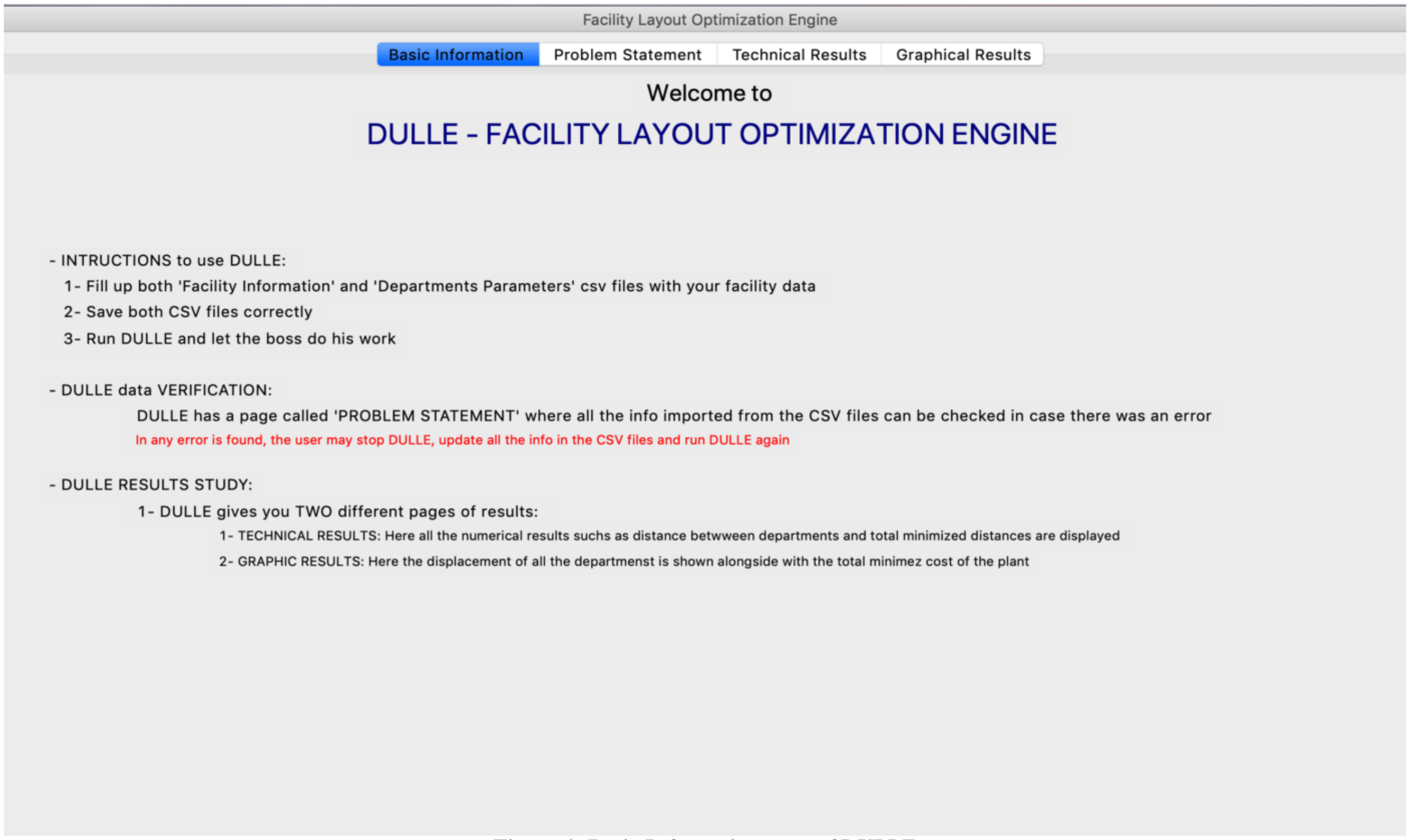
6- Program's Algorithm Implementation

For a better user experience and understating about how DULLE's is supposed to be used, every feature implemented to the software and the way it was implemented will be explained in this section. From the first to the last page of DULLE, every information and useful tips will be covered. Later on, every python language code needed to create the program and the user interface will be also explained. Furthermore, every modification to the original mathematical formulation showed in Section 4 needed in order to adequate it to the python and gurobi language will be also stated and explained.

6.1- Program's User Interface

The user experience and reviews are the most valuable feedback for any software engineer and it's the best way of measuring how good the product is and what parts need to improve. With the willing to achieve the best possible feedback and opinions from the users it is desirable to let and inform the user how to control and use the product at its full potential so that this feedback is as realistic as possible. Therefore, every implementation in DULLE from the first page will be explained, including any minor noticeable detail.

The first page of every book, paper or report is always a simple introduction and so DULLE needs one as well. For DULLE, it was thought to include an easy step-by-step introduction so that the user would be comfortable and would feel that he knows what DULLE is doing and how. Below is attached a screenshot of the mentioned first-page's introduction:



Welcome to

DULLE - FACILITY LAYOUT OPTIMIZATION ENGINE

- INTRUCTIONS to use DULLE:
  1- Fill up both 'Facility Information' and 'Departments Parameters' csv files with your facility data
  2- Save both CSV files correctly
  3- Run DULLE and let the boss do his work

- DULLE data VERIFICATION:
  DULLE has a page called 'PROBLEM STATEMENT' where all the info imported from the CSV files can be checked in case there was an error
  In any error is found, the user may stop DULLE, update all the info in the CSV files and run DULLE again

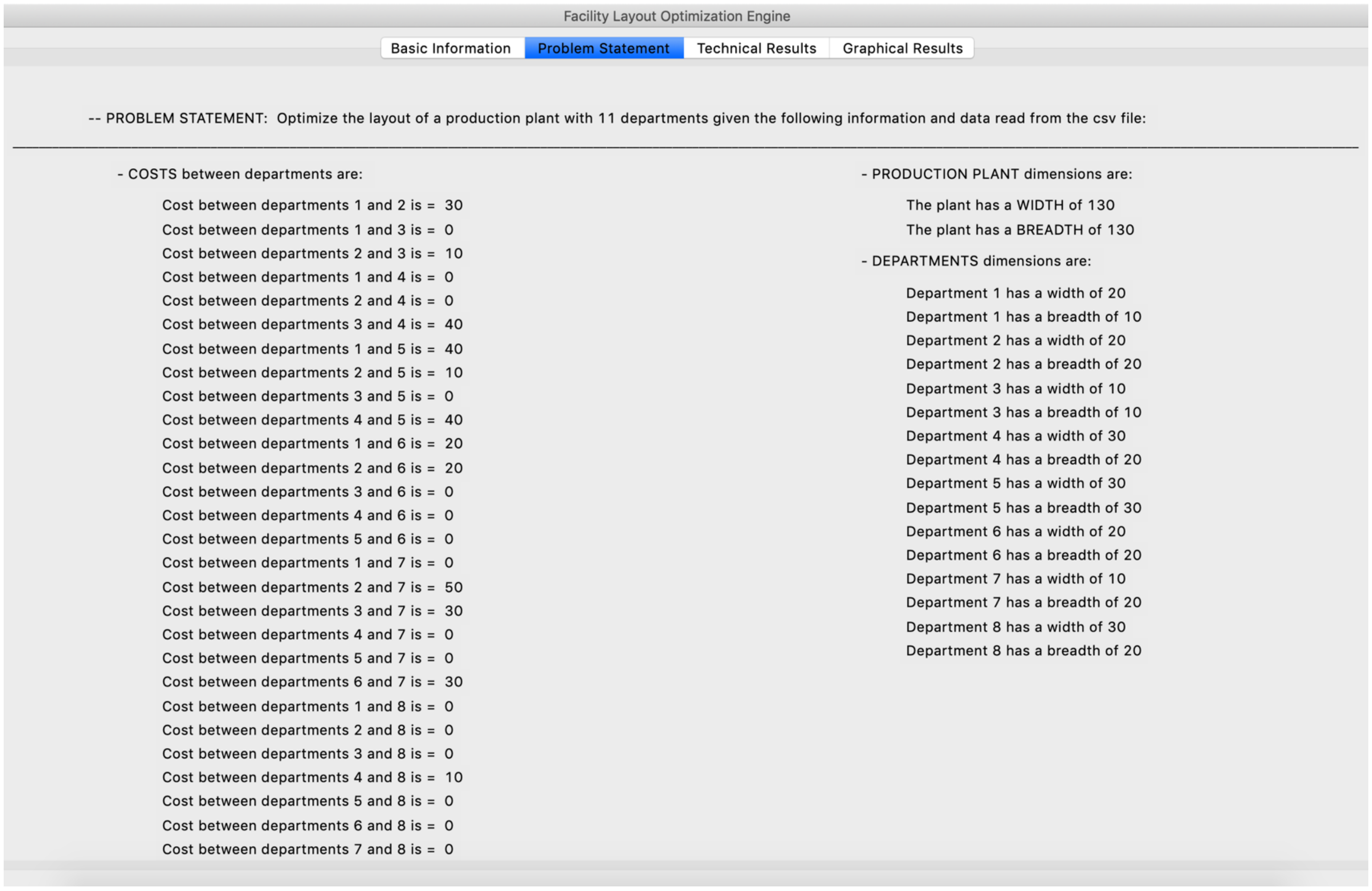
- DULLE RESULTS STUDY:
  1- DULLE gives you TWO different pages of results:
    1- TECHNICAL RESULTS: Here all the numerical results suchs as distance betwween departments and total minimized distances are displayed
    2- GRAPHIC RESULTS: Here the displacement of all the departmenst is shown alongside with the total minimez cost of the plant

Figure 6: Basic Information page of DULLE

As it can be seen in the above screenshot, there is a simple explanation in the first page about what DULLE does and what it solves, just in case the user is a little bit lost about what he needs to solve. There is also a simple step-by-step guide on what the user needs to do in order to make DULLE perform great and achieve good solutions. These steps include three different parts that relate with the next three pages of the program. Firstly, it explains everything about the handling and interaction of the user’s input information. The procedure to manage the csv files within the excel platform and how to save them in order to let DULLE read and store them correctly is also explained here. Later, it is recommended to give a look at all the information imported from the user to check that everything is correct. This could be done thanks to the displayed information in the second page which will be showed later. Last, the two most important DULLE's pages are explained. These pages show all the results and there is a quick description of each of them to know what the user may find in each of them.

The screenshot below shows the second page of DULLE's program, which is simple and easy to understand. It quickly shows all the information the user previously sent to the program through the two csv files explained in Section 5. The information displayed includes three main types of variables. These variables are: dimensions of the departments (width and breadth), dimensions of the facility (width and breadth) and costs between departments. As it should be known by now the handling cost between departments is calculated thanks to the *Equation 4.17* already stated in Section 4 of this paper. The total handling cost defined by this equation depends in other three variables that the user is expected to provide to the program. These variables are: number of parts or trips between department "i" and department "j", cost of workers moving those parts or making those trips from department "i" to department "j" and the inverse speed (hours per meter) of those workers moving from department "i" to department "j". Combining these three variables and solving *Equation 4.17,* the handling total cost between each pair of departments can be computed. DULLE makes all the math straightway and shows the user the total handling cost in this page so that he can check if every value stored is correct.

Apart from the total handling cost between each pair of departments, it has a brief explanation of the problem statement up top. It also displays the other two needed variables on the right side of the page that are facility's dimensions and departments dimensions.

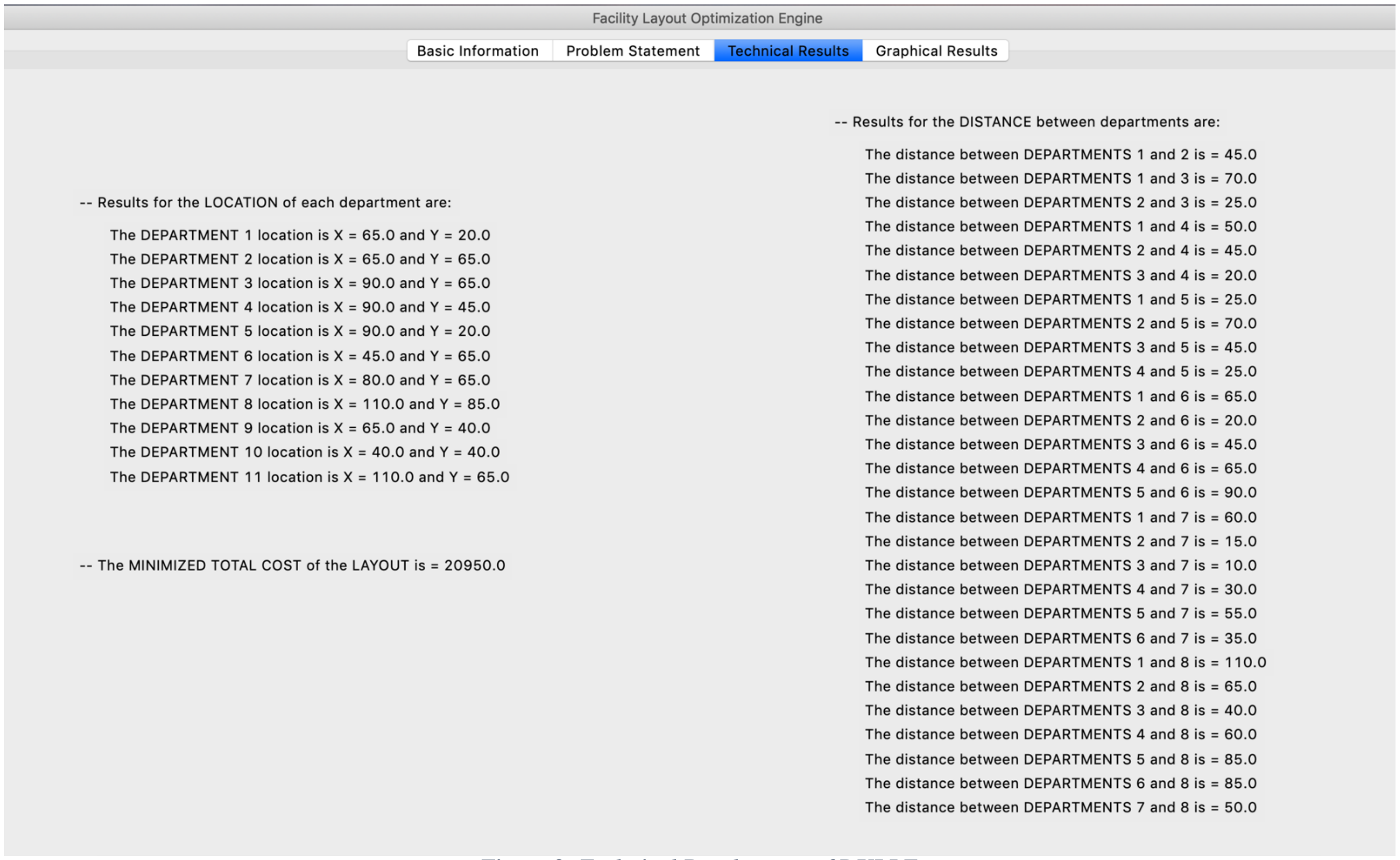


```
-- PROBLEM STATEMENT:  Optimize the layout of a production plant with 11 departments given the following information and data read from the csv file:
_______________________________________________________________________________________________

 - COSTS between departments are:                         - PRODUCTION PLANT dimensions are:

     Cost between departments 1 and 2 is =  30                 The plant has a WIDTH of 130
     Cost between departments 1 and 3 is =  0                  The plant has a BREADTH of 130
     Cost between departments 2 and 3 is =  10
     Cost between departments 1 and 4 is =  0             - DEPARTMENTS dimensions are:
     Cost between departments 2 and 4 is =  0
     Cost between departments 3 and 4 is =  40                 Department 1 has a width of 20
     Cost between departments 1 and 5 is =  40                 Department 1 has a breadth of 10
     Cost between departments 2 and 5 is =  10                 Department 2 has a width of 20
     Cost between departments 3 and 5 is =  0                  Department 2 has a breadth of 20
     Cost between departments 4 and 5 is =  40                 Department 3 has a width of 10
     Cost between departments 1 and 6 is =  20                 Department 3 has a breadth of 10
     Cost between departments 2 and 6 is =  20                 Department 4 has a width of 30
     Cost between departments 3 and 6 is =  0                  Department 4 has a breadth of 20
     Cost between departments 4 and 6 is =  0                  Department 5 has a width of 30
     Cost between departments 5 and 6 is =  0                  Department 5 has a breadth of 30
     Cost between departments 1 and 7 is =  0                  Department 6 has a width of 20
     Cost between departments 2 and 7 is =  50                 Department 6 has a breadth of 20
     Cost between departments 3 and 7 is =  30                 Department 7 has a width of 10
     Cost between departments 4 and 7 is =  0                  Department 7 has a breadth of 20
     Cost between departments 5 and 7 is =  0                  Department 8 has a width of 30
     Cost between departments 6 and 7 is =  30                 Department 8 has a breadth of 20
     Cost between departments 1 and 8 is =  0
     Cost between departments 2 and 8 is =  0
     Cost between departments 3 and 8 is =  0
     Cost between departments 4 and 8 is =  10
     Cost between departments 5 and 8 is =  0
     Cost between departments 6 and 8 is =  0
     Cost between departments 7 and 8 is =  0
```

Figure 7: Problem Statement page of DULLE

Moving into the mentioned last two pages of the program, which are result-focused pages, first a clear explanation on what each page stands for will be given. As it can be seen in the last screenshot of the program and as it was stated in the introduction of the section, the titles for these third and fourth pages are “Technical Results” and “Graphical Results” respectively. Taking a further look in each one of them, in the third page all the mathematical results are displayed. These mathematical results include the position of each department, the distance between each pair of departments and the total cost that was minimized. It has a similar layout as the second page and displays all the information in the same way the second page does.

We can say then that the only difference between the second and third page is the information it displays. Below we can see a screenshot showing the layout of the just described third page named "Technical Results":



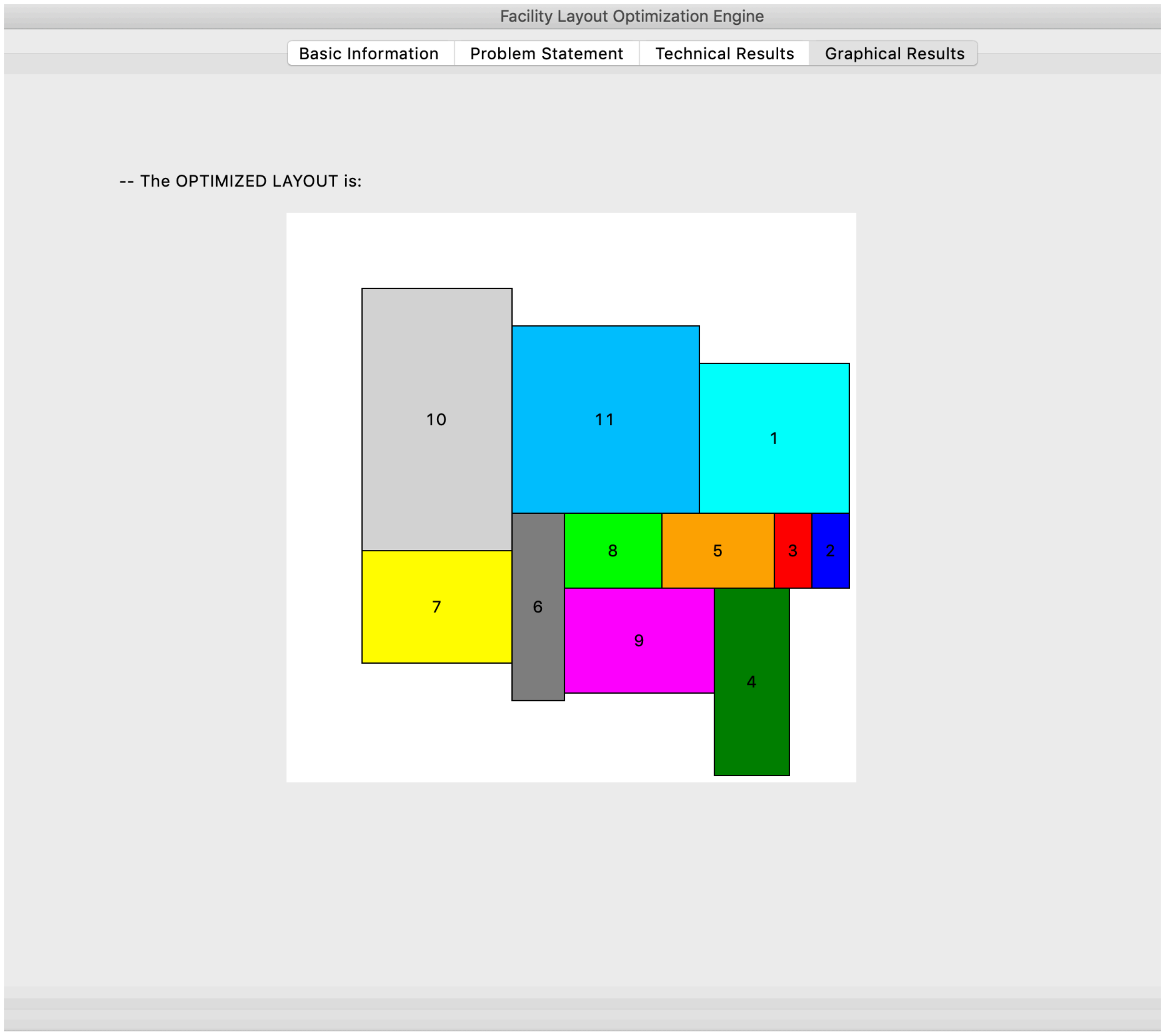
```
-- Results for the LOCATION of each department are:

     The DEPARTMENT 1 location is X = 65.0 and Y = 20.0
     The DEPARTMENT 2 location is X = 65.0 and Y = 65.0
     The DEPARTMENT 3 location is X = 90.0 and Y = 65.0
     The DEPARTMENT 4 location is X = 90.0 and Y = 45.0
     The DEPARTMENT 5 location is X = 90.0 and Y = 20.0
     The DEPARTMENT 6 location is X = 45.0 and Y = 65.0
     The DEPARTMENT 7 location is X = 80.0 and Y = 65.0
     The DEPARTMENT 8 location is X = 110.0 and Y = 85.0
     The DEPARTMENT 9 location is X = 65.0 and Y = 40.0
     The DEPARTMENT 10 location is X = 40.0 and Y = 40.0
     The DEPARTMENT 11 location is X = 110.0 and Y = 65.0

-- The MINIMIZED TOTAL COST of the LAYOUT is = 20950.0
```

```
-- Results for the DISTANCE between departments are:

     The distance between DEPARTMENTS 1 and 2 is = 45.0
     The distance between DEPARTMENTS 1 and 3 is = 70.0
     The distance between DEPARTMENTS 2 and 3 is = 25.0
     The distance between DEPARTMENTS 1 and 4 is = 50.0
     The distance between DEPARTMENTS 2 and 4 is = 45.0
     The distance between DEPARTMENTS 3 and 4 is = 20.0
     The distance between DEPARTMENTS 1 and 5 is = 25.0
     The distance between DEPARTMENTS 2 and 5 is = 70.0
     The distance between DEPARTMENTS 3 and 5 is = 45.0
     The distance between DEPARTMENTS 4 and 5 is = 25.0
     The distance between DEPARTMENTS 1 and 6 is = 65.0
     The distance between DEPARTMENTS 2 and 6 is = 20.0
     The distance between DEPARTMENTS 3 and 6 is = 45.0
     The distance between DEPARTMENTS 4 and 6 is = 65.0
     The distance between DEPARTMENTS 5 and 6 is = 90.0
     The distance between DEPARTMENTS 1 and 7 is = 60.0
     The distance between DEPARTMENTS 2 and 7 is = 15.0
     The distance between DEPARTMENTS 3 and 7 is = 10.0
     The distance between DEPARTMENTS 4 and 7 is = 30.0
     The distance between DEPARTMENTS 5 and 7 is = 55.0
     The distance between DEPARTMENTS 6 and 7 is = 35.0
     The distance between DEPARTMENTS 1 and 8 is = 110.0
     The distance between DEPARTMENTS 2 and 8 is = 65.0
     The distance between DEPARTMENTS 3 and 8 is = 40.0
     The distance between DEPARTMENTS 4 and 8 is = 60.0
     The distance between DEPARTMENTS 5 and 8 is = 85.0
     The distance between DEPARTMENTS 6 and 8 is = 85.0
     The distance between DEPARTMENTS 7 and 8 is = 50.0
```

Figure 8: Technical Results page of DULLE

Last but not least is located the most important page of the program. In this last page, the final layout solution of the facility problem is displayed. Every department is associated with a color, and each one of them has also displayed its department number in the center. It is the most valuable and important page of the program because it gives the user the feel and the knowledge that the problem was solved efficiently, and that the final solution at least seems

optimal. Although it is assured and compared afterwards in Section 8 that the results are really good and that the level of optimization is really high, thanks to this page the user feels he is in good hands with DULLE. Making the user trust the program is one of the most important points to any project, start-up or even company, but it is even more important for DULLE as it needs to demonstrate how clever, fast and precise it is. Below there is a screenshot showing the described fourth page of the program during one of the experimental studies made:



-- The OPTIMIZED LAYOUT is:

Figure 9: Graphical Results page of DULLE

6.2- Program’s User Interface Implementation to Python

Understanding how DULLE was created and coded is as important as understanding how to use it. For that reason, all the code implementations needed to create the user interface and the program itself are going to be explained and introduced. As it is very likely that the reader doesn’t handle python codes quite often, explain every part of the code will be explained as simplest as possible.

The most important part when reading and understanding any code is to clearly know and understand each part of it and how these parts are arranged. Therefore, the main structure of DULLE’s code is the following:

1. Package importing
2. Functions declaration
3. Variables declaration
4. Function calling to store user information
5. Creation of canvas and pages
6. Displaying information of first and second pages
7. Optimization Model
8. Storing optimized values
9. Printing third and fourth pages results

I will now explain briefly each one of the parts mentioned above and how each part was written in the program. For further specifications the full code of the program will be attached in the appendix of this paper.

1. **Package importing**

Python is a language that gives the option to work and build different modules and packages to enhance the experience of the program. For DULLE, as it was mentioned before,

several modules and packages such as gurobi and tkinter were used. In order to be able to call them in our code, we need to import it from the library by using these commands:

```
"from tkinter import *
from gurobipy import *"
```

2. Functions declaration

As in every code, no matter what language its being used, functions are needed to make the program simpler and more straight forward. For DULLE three kind of functions were declared in order to be able to store all the information imported from the user's csv files. These three functions let the program read the three variables needed to calculate the total handling cost between departments, the facility's dimensions and the departments dimensions. One example of these functions is the following:

```
"class ReadFileProduction():
        def __init__ (self, filename):
                with open (filename, 'r', encoding='utf-8') as f_input:
                        csv_input = csv.reader(f_input)
                        self.production = list(csv_input)

        def GetProduction (self, row , column):
                return self.production[row-1][column-1]"
```

3. Variables declaration

Declaring variables is also a usual step in every code. For DULLE, it was needed to declare a total of 17 array variables which each one of them could include an infinite number of values. These array variables are for example the number of trips between departments, the inverse speed of the workers within these departments and the rest of the variables needed to calculate the handling cost. These variables also include all the variables needed for the optimization model, such as the position *"x"* and *"y"* of each department or the distance between departments.

4. Function calling to store user information

In order to store the information imported from the user it is required to call the functions previously declared. When calling them, each value of the user information is stored in one position of the declared array variables. After storing the information, DULLE computes the handling cost between departments thanks to *[Equation 4.17]* and stores the new final cost values in its corresponding array variable. Below it is showed how each value is stored in its corresponding variable.

```
"DepartmentsInfo = ReadFileDimensions('Department Parameters.csv')
for row in range(1,NumDepts+1):
    W[row] = float(DepartmentsInfo.dimensions[row][1])
    B[row] = float(DepartmentsInfo.dimensions[row][2])"
```

5. Creation of canvas and pages

This is the first time that the module tkinter is used. It is used to create the global user interface, also known as canvas. Inside this canvas tkinter can create any type of interface like texts boxes, figures, pages and graphs. This time four pages were created including all the information explained before. Below are some code examples on how to create each page with tkinter.

```
"nb = ttk.Notebook(roota)

first = ttk.Frame(roota)
```

nb.add(first, text='Basic Information')"

6. Displaying information of the first and second pages

Moving forward with the module tkinter, all the information included in the first and second page is written now. This is an example of how to print text on any page of the program:

"d_label1= Label(first , text = "- INTRUCTIONS to use DULLE: " , bg="gray92").place(relx=0.10,rely=0.20)"

Reading the code from left to right it can be seen that the name of the label where the text is printed is *"d_label1"*, and then the label is created in the first page. The text printed in this label is *"-INTRUCTIONS to use DULLE:"* and the background of the label is in "gray92" color. Finally, it is seen that the label is located between the full page in a relative "x" of 0.10 out of 1 and in a relative "y" of 0.20 out of 1. These positions mean that if the full wide dimension of the page is 100 units, the label will be located in the unit 10 for the x axis and 20 for the y axis starting from the center of coordinates, which is in the top left corner of the page.

7. Optimization Model

Adapting a real-world mathematical model to perform in a computer can be really tricky mainly because the model has to be well broken-down to let the software run fast and easy.

Several modules such as gurobi have several limitations in this topic. For example, gurobi struggles with quadratic equality constraints so that if we want to have a good and optimized software it is recommended to avoid quadratic constraints or quadratic variables. In order to achieve a good result, some changes were performed to parts of the formulation proposed in Section 4 and adapted to gurobi's formulation basics. The most noticeable change made to the formulation relates to the calculation of the distance between departments. This distance, as explained in Section 4, is calculated through the "*Manhattan distance"* method just to avoid having a quadratic distance equation. The other option would be using the "*Euclidean distance"*

method but it would create an undesirable quadratic constraint. Furthermore, gurobi struggles when calling the absolute value function for more than one variable at once, so for coding it into DULLE more variables had to be declared and the distance equation had to be broken into several declarations. Below is showed the code lines used to declare Equation 4.6 stated in Section 4:

```
"model.addConstr (Dx[i,j] == (x[i] - x[j]))
model.addConstr (Dy[i,j] == (y[i] - y[j]))
model.addConstr (Dxabs[i,j] == abs_(Dx[i,j]))
model.addConstr (Dyabs[I,,j] == abs_(Dy[i,j]))
model.addConstr (Dij[i,j] == Dxabs[i,j] + Dyabs[i,j])"
```

Taking a look at the code showed above it can be checked that the first two lines declare the "x" and the "y" axis distance respectively. As they can be either positive or negative and the program doesn't know if "x[i]" is greater, equal or less than "x[j]", their absolute values are needed. Later, thanks to the third and fourth lines, the absolute values of the previously computed distances are saved into new variables called *"Dxabs"* and *"Dyabs"*. Finally, the absolute values just calculated are summed up to get the total "*Manhattan Distance"* between departments *"i"* and *"j".*

Due to the fact that quadratic constraints are not recommended to be used in the program, some other assumptions had to be made. At first, it would be easier to use a quadratic equation to solve the non-overlapping constraint as it would be more straight forward, and it would only take one line to declare. As it was recommended by the program performance abilities to not declare quadratic constraints another way to handle the non-overlapping constraint was developed. In this case, it resulted that the new linear formulation proposed to solve the constraint [Equation 4.7 to 4.11] ended up being very effective and maybe even being better than the quadratic constraint. Resulting with a better result after changing some parameters or equations of a model to make it simpler does not happen in every time and it could

be different in other cases. Furthermore, it could make the design process slow down and stuck the project development.

8. Storing optimized values

After computing the optimization, gurobi does not store by itself the results in the desired variables. It is required to get the results of the optimization model and store each value in the corresponding created variable after solving the optimization. The command used in python to get the value of each gurobi variable and store it in a python variable is the following and must be repeated for every variable that is needed to later on create the results pages:

"x_1 = model.getVarByName("x_1")"
"x_1 = round (x_1.X , 4)"

As it can be seen, the variable that we want to get and store in the above example is "*x_1*"*.* Although gurobi has this value stored as *"x_1"* already, it isn't related by any means to th python variable "x_1". Thanks to the command *"getVarByName"* the value from gurobi is passed and stored in the python variable so that later on it can be printed without any problem in any of the program's pages.

9. Printing third and fourth pages results

Just as it was done in the Part 6 of the code with the user information, it is required now to display the results information. This time, in the third page all the variables such as the distance between departments or the total cost of the layout have to be shown. The same command as in the one used in Part 6 of the code has to be used here to print these variables in the third page. In this case, the final layout of the facility given the positions of the departments and their dimensions is displayed. Thanks to these values and the following command each department within the grid of the layout is printed:

“Dept1_rectangle = canvas3.create_rectangle(3(x_1 - (W[1]/2)), 3*(y_1 + (B[1]/2)), 3*(x_1 + (W[1]/2)), 3*(y_1 - (B[1]/2)), fill="cyan")”*

*“Dept1_lable = canvas3.create_text(3*x_1, 3*y_1 , text = "1" , fill = "black")”*

In the first line it is seen how the rectangle is created whereas in the second row the number of the department is printed on top of the center of itself. To print the rectangle for each department the top left and bottom right coordinates of each department are calculated. These coordinates are then given to the command “*create_rectangle* “ alongside with the color which in this case is cyan. The coordinates are multiplied by three to scale the drawing and make it easier to see in the page.

7- Problems Statements

In order to achieve the maximum possible conclusions and useful results there will be four different problems resolved by DULLE. They are based off of the same problem, with the main difference that each of them will have a different number of departments and departments sizes to optimize. Some of the scenarios or problems have been resolved previously in other researches such as ***[HASD17]*** *(Problems 1,3 and 4) ,* ***[WONG15]*** *(Problems 1, 3 and 4),* ***[IMAM93]*** *(Problem 2),* ***[KHAR88]*** *(Problem 2) and* ***[IMAM01]*** *(Problem 2).* Every problem will be solved with DULLE's software to later on compare all the result with the already published and mentioned papers. It is expected to have similar results compared to the other researches done but differing sometimes in the result found or in the time spent to solve it. For smaller problems DULLE is expected to perform better in both aspects whereas when the problem gets more complicated DULLE's performance should decrease but would still compete with the other researches.

As it is already known, DULLE's software needs to receive two csv files regarding the facility and departments information. In the following tables all this information is included for each problem.

- **<u>PROBLEM 1</u> - (8 DEPARTMENTS)**

DEPARTMENTS PRODUCTION	1	2	3	4	5	6	7	8
1	0	0.5	0.5	0	0	2	0	3
2	0.5	0	4	1.5	2	0	0	1
3	0.5	0.4	0	2	0	0.6	1	0
4	0	1.5	2	0	1	1	0	2
5	0	2	0	1	0	0	0	2
6	2	0	0.6	1	0	0	4	0
7	0	0	1	0	0	4	0	1
8	3	1	0	2	2	0	1	0

Table 1: Departments production in Problem 1

DEPARTMENTS WORKERS COST PER HOUR	1	2	3	4	5	6	7	8
1	0	1	2	4	1	2	1	1
2	1	0	0.5	1	2	1	1	1
3	2	0.5	0	1	1	5	2	1
4	4	1	1	0	5	2	1	1
5	1	2	1	5	0	1	1	1
6	2	1	5	2	1	0	1	1
7	1	1	2	1	1	1	0	1
8	1	1	1	1	1	1	1	0

Table 2: Departments workers cost per hour in Problem 1

DEPARTMENTS INVERSE SPEED (HR/METER)	1	2	3	4	5	6	7	8
1	0	2	2	0.5	1	0.5	1	1
2	2	0	2	2	1.5	1	1	2
3	2	2	0	1	1	1	0.5	1
4	0.5	2	1	0	1	1	1	1
5	1	1.5	1	1	0	1	1	2
6	0.5	1	1	1	1	0	1	1
7	1	1	0.5	1	1	1	0	1
8	1	2	1	1	2	1	1	0

Table 3: Departments inverse speed in Problem 1

DEPARTMENTS TOTAL PRODUCTION	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	1	2	0	0	0	2	0
2	1	0	4	3	6	0	0	2
3	2	4	0	2	0	3	1	0
4	0	3	2	0	5	2	0	2
5	0	6	0	5	0	0	0	4
6	0	0	3	2	0	0	4	0
7	2	0	1	0	0	4	0	1
8	0	2	0	2	4	0	1	0

Table 4: Departments total production in Problem 1

DEPARTMENT PARAMETERS	**WIDTH**	**BREADTH**
1	3	2
2	4	5
3	2	2
4	3	3
5	2	4
6	4	4
7	4	4
8	3	4

Table 5: Departments parameters in Problem 1

- **FACILITY DIMENSIONS:**
  - **X AXIS** = 12
  - **Y AXIS** = 12

- **PROBLEM 2 – (12 DEPARTMENTS)**

DEPARTMENTS TOTAL PRODUCTION	1	2	3	4	5	6	7	8	9	10	11	12
1	0	2	4	1	3	5	2	4	7	3	5	7
2	2	0	2	3	1	3	4	2	4	3	3	5
3	4	2	0	5	3	1	6	4	3	7	5	3
4	1	3	5	0	2	4	1	3	5	2	4	6
5	3	1	3	2	0	2	3	1	3	4	2	4
6	5	3	1	4	2	0	5	3	1	6	4	3
7	2	4	6	1	3	5	0	2	4	1	3	5
8	4	2	4	3	1	3	2	0	3	5	3	6
9	7	4	3	5	3	1	4	3	0	1	3	5
10	3	3	7	2	4	6	1	5	1	0	5	3
11	5	3	5	4	2	4	3	3	3	5	0	5
12	7	5	3	6	4	3	5	6	5	3	5	0

Table 6: Departments total production in Problem 2

DEPARTMENT PARAMETERS	WIDTH	BREADTH
1	1	1
2	1	1
3	1	1
4	1	1
5	1	1
6	1	1
7	1	1
8	1	1
9	1	1
10	1	1
11	1	1
12	1	1

Table 7: Departments parameters in Problem 2

- **FACILITY DIMENSIONS:**
    - **X AXIS** = 15

- **Y AXIS** = 15

- **<u>PROBLEM 3</u> – (11 DEPARTMENTS)**

DEPARTMENTS PRODUCTION	1	2	3	4	5	6	7	8	9	10	11
1	0	0.5	0.5	1	1	3	2	1	1	3	3
2	5	0	1	0.5	2	2	2	1	3	1.5	6
3	5	1	0	2	1	6	1	3	6	3	6
4	1	0.5	2	0	1	5	0.5	6	6	1.5	6
5	1	2	1	1	0	4	3	6	2	2.5	3
6	3	2	6	5	4	0	1.5	3	4	5	6
7	2	2	1	0.5	3	1.5	0	2	4	1	1
8	1	1	3	6	6	3	2	0	6	1.5	3
9	1	3	6	6	2	4	4	6	0	5	5
10	3	1.5	3	1.5	2.5	5	1	1.5	5	0	2
11	3	6	6	5	3	6	1	3	5	2	0

Table 8: Departments production in Problem 3

DEPARTMENTS WORKERS COST PER HOUR	1	2	3	4	5	6	7	8	9	10	11
1	0	4	8	1	2	0.5	1	6	3	1	1
2	4	0	1	2	1	3	1	3	2	2	1
3	8	1	0	1	1	2	1	1	2	4	1
4	1	2	1	0	1	1	2	1	1	2	1
5	2	1	1	1	0	1	1	1	2	2	2
6	0.5	3	2	1	1	0	1	1	1	1	1
7	1	1	1	2	1	1	0	2	1	1	1
8	6	3	1	1	1	1	2	0	1	2	1
9	3	2	2	1	2	1	1	1	0	1	1
10	1	2	4	2	2	1	1	2	1	0	1
11	1	1	1	1	2	1	1	1	1	1	0

Table 9: Departments workers cost per hour in Problem 3

DEPARTMENTS INVERSE SPEED (HR/METER)	1	2	3	4	5	6	7	8	9	10	11
1	0	1	0.5	1	1	4	1	1	2	1	2
2	1	0	1	1	1	1	2	2	1	1	1
3	0.5	1	0	1	2	0.5	1	2	0.5	0.5	1
4	1	1	1	0	1	1	1	1	1	1	1
5	1	1	2	1	0	1	1	1	1	1	1
6	4	1	0.5	1	1	0	2	2	1	1	1
7	1	2	1	1	1	2	0	1	1	1	1
8	1	2	2	1	1	2	1	0	1	1	1
9	2	1	0.5	1	1	1	1	1	0	1	1
10	1	1	0.5	1	1	1	1	1	1	0	1
11	2	1	1	1	1	1	1	1	1	1	0

Table 10: Departments inverse speed in Problem 3

DEPARTMENTS TOTAL PRODUCTION	1	2	3	4	5	6	7	8	9	10	11
1	0	2	2	1	2	6	2	6	6	3	6
2	2	0	1	1	2	6	4	6	6	3	6
3	2	1	0	2	2	6	1	6	6	6	6
4	1	1	2	0	1	5	1	6	6	3	6
5	2	2	2	1	0	4	3	6	4	5	6
6	6	6	6	5	4	0	3	6	4	5	6
7	2	4	1	1	3	3	0	4	4	1	1
8	6	6	6	6	6	6	4	0	6	3	3
9	6	6	6	6	4	4	4	6	0	5	5
10	3	3	6	3	5	5	1	3	5	0	2
11	6	6	6	5	6	6	1	3	5	2	0

Table 11: Departments Total Production in Problem 3

DEPARTMENT PARAMETERS	WIDTH	BREADTH
1	4	4
2	1	2
3	1	2
4	2	5
5	3	2
6	1.4	5
7	4	3
8	2.6	2
9	4	2.8
10	4	7
11	5	5

Table 12: Departments parameters in Problem 3

- **FACILITY DIMENSIONS:**
    - **X AXIS** = 15
    - **Y AXIS** = 15

- **PROBLEM 4 – (20 DEPARTMENTS)**

DEPARTMENTS PRODUCTION	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	30	0	0	20	10	0	0	10	0	0	25	30	0	10	0	0	5	0	0
2	30	0	10	0	10	10	25	0	15	0	0	0	20	0	15	0	10	10	5	10
3	0	10	0	40	0	0	15	0	0	0	10	0	0	0	0	0	10	0	20	15
4	0	0	40	0	20	0	0	10	25	10	0	20	0	0	20	10	0	10	0	0
5	20	10	0	40	0	0	0	0	10	10	10	10	0	0	15	10	0	10	0	40
6	10	20	0	0	0	0	15	0	0	25	0	0	10	0	0	0	10	0	0	0
7	0	50	30	0	0	30	0	0	0	0	0	0	20	0	10	0	30	20	0	5
8	0	0	0	10	0	0	0	0	0	0	20	0	0	25	0	20	0	10	0	0
9	10	30	0	50	10	0	0	0	0	15	0	10	0	0	0	10	0	0	0	0
10	0	0	0	30	40	50	0	0	30	0	0	25	0	10	10	10	0	15	20	0
11	0	0	10	0	10	0	0	20	0	0	0	0	0	50	25	20	0	40	15	10
12	25	0	0	20	50	0	0	0	50	50	0	0	50	0	20	0	0	10	0	0
13	30	20	0	0	0	40	40	0	0	0	0	50	0	0	30	0	20	0	0	0
14	0	0	0	0	0	0	0	50	0	10	50	0	0	0	0	25	0	50	10	0
15	10	30	0	40	30	0	20	0	0	20	50	20	30	0	0	0	10	20	15	30
16	0	0	0	50	20	0	0	40	20	40	40	0	0	50	0	0	40	25	0	0
17	0	30	50	0	0	20	30	0	0	0	0	0	20	0	10	40	0	0	10	50
18	5	10	0	10	50	0	20	10	0	30	40	10	0	50	40	50	0	0	40	10
19	0	20	20	0	0	0	0	0	0	40	30	0	0	10	30	0	10	40	0	25
20	0	30	30	0	40	0	10	0	0	0	10	0	0	0	30	0	50	10	50	0

Table 13: Departments production in Problem 4

WORKERS COST PER HOUR	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	2	1	2	2	1	6	1	1	1	2	1	2	5	2	1	2	3	1
2	1	0	1	2	1	2	2	1	2	1	1	0	1	0	2	1	3	1	4	1
3	2	1	0	1	0	0	2	0	1	0	1	0	0	0	0	0	5	0	1	2
4	1	2	1	0	1	0	0	1	1	3	0	1	0	0	2	5	0	1	0	1
5	2	1	0	1	0	0	0	0	1	4	1	5	0	0	2	2	0	5	0	1
6	2	2	0	0	0	0	2	0	0	2	0	0	4	0	0	0	2	0	0	0
7	1	2	2	0	0	2	0	0	0	0	0	0	2	0	2	0	1	1	0	1
8	6	1	0	1	0	0	0	0	0	0	1	0	0	2	0	2	0	1	0	0
9	1	2	1	1	1	0	0	0	0	2	0	5	0	0	0	2	0	0	0	0
10	1	1	0	3	4	2	0	0	2	0	0	2	0	1	2	4	0	2	2	0
11	1	1	1	0	1	0	0	1	0	0	0	0	0	1	2	2	0	1	2	1
12	2	0	0	1	5	0	0	0	5	2	0	0	1	0	1	0	0	1	0	0
13	1	1	0	0	0	4	2	0	0	0	0	1	0	0	1	0	1	0	0	0
14	2	0	0	0	0	0	0	2	0	1	1	0	0	0	0	2	0	1	1	0
15	5	2	0	2	2	0	2	0	0	2	2	1	1	0	0	0	1	2	2	1
16	2	1	0	5	2	0	0	2	2	4	2	0	0	2	0	0	1	2	0	0
17	1	3	5	0	0	2	1	0	0	0	0	0	1	0	1	1	0	0	1	1
18	2	1	0	1	5	0	1	1	0	2	1	1	0	1	2	2	0	0	0	1
19	3	4	1	0	0	0	0	0	0	2	2	0	0	1	2	0	1	0	0	2
20	1	1	2	1	1	0	1	0	0	0	1	0	0	0	1	0	1	1	2	0

Table 14: Workers cost per hour in Problem 4

DEPARTMENTS INVERSE SPEED	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	1	1	1	1	1	1	4	1	2	1	1	2	1	2	1	1	2	2
2	1	0	1	1	1	1	1	1	1	2	1	0	1	0	1	0	1	1	1	3
3	1	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	1	0	1	1
4	1	1	1	0	2	0	0	1	2	1	0	1	0	0	1	1	0	1	0	0
5	1	1	0	2	0	1	0	0	1	1	1	1	0	0	1	1	0	1	0	1
6	1	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	1	0	0	0
7	1	1	1	0	0	1	0	0	0	0	0	0	1	1	1	0	1	1	0	2
8	1	1	0	1	0	0	0	0	0	0	1	1	0	1	1	1	0	1	0	0
9	4	1	1	2	1	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0
10	1	2	0	1	1	1	0	0	1	0	0	1	0	1	1	1	0	1	1	0
11	2	1	1	0	1	0	0	1	0	0	0	0	0	1	1	1	0	1	1	1
12	1	0	0	1	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0	0
13	1	1	0	0	0	1	1	0	0	0	0	1	0	0	1	0	1	0	0	0
14	2	0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	0	1	1	0
15	1	1	0	1	1	0	1	1	0	1	1	1	1	0	0	0	1	1	1	1
16	2	0	0	1	1	1	0	1	1	1	1	0	0	1	0	0	1	1	0	0
17	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	0	1	1
18	1	1	0	1	1	0	1	1	0	1	1	1	0	1	1	1	0	0	1	1
19	2	1	1	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	0	1
20	2	3	1	0	1	0	2	0	0	0	1	0	0	0	1	0	1	1	1	0

Table 15: Departments inverse speed in Problem 4

DEPARTMENTS TOTAL HANDLING COST	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	30	0	0	40	20	0	0	40	0	0	50	30	0	50	0	0	10	0	0
2	30	0	10	0	10	20	50	0	30	0	0	0	20	0	30	0	30	10	20	30
3	0	10	0	40	0	0	30	0	0	0	10	0	0	0	0	0	50	0	20	30
4	0	0	40	0	40	0	0	10	50	30	0	20	0	0	40	50	0	10	0	0
5	40	10	0	40	0	0	0	0	10	40	10	50	0	0	30	20	0	50	0	40
6	40	20	0	0	0	0	30	0	0	50	0	0	40	0	0	0	20	0	0	0
7	0	50	30	0	0	30	0	0	0	0	0	0	40	0	20	0	30	20	0	10
8	0	0	0	10	0	0	0	0	0	0	20	0	0	50	0	40	0	10	0	0
9	40	30	0	50	10	0	0	0	0	30	0	50	0	0	0	20	0	0	0	0
10	0	0	0	30	40	50	0	0	30	0	0	50	0	10	20	40	0	30	40	0
11	0	0	10	0	10	0	0	20	0	0	0	0	0	50	50	40	0	40	30	10
12	50	0	0	20	50	0	0	0	50	50	0	0	50	0	20	0	0	10	0	0
13	30	20	0	0	0	40	40	0	0	0	0	50	0	0	30	0	20	0	0	0
14	0	0	0	0	0	0	0	50	0	10	50	0	0	0	0	50	0	50	10	0
15	50	30	0	40	30	0	20	0	0	20	50	20	30	0	0	0	10	40	30	30
16	0	0	0	50	20	0	0	40	20	40	40	0	0	50	0	0	40	50	0	0
17	0	30	50	0	0	20	30	0	0	0	0	0	20	0	10	40	0	0	10	50
18	10	10	0	10	50	0	20	10	0	30	40	10	0	50	40	50	0	0	40	10
19	0	20	20	0	0	0	0	0	0	40	30	0	0	10	30	0	10	40	0	50
20	0	30	30	0	40	0	10	0	0	0	10	0	0	0	30	0	50	10	50	0

Table 16: Departments total handling cost in Problem 4

DEPARTMENTS DIMENSIONS	WIDTH	BREADTH
1	2	1
2	2	2
3	1	1
4	3	2
5	3	3
6	2	2
7	1	2
8	3	2
9	2	3
10	3	3
11	3	2
12	2	1
13	2	3
14	3	3
15	3	3
16	2	2
17	2	3
18	2	2
19	2	2
20	1	2

Table 17: Departments dimensions in Problem 4

- **FACILITY DIMENSIONS:**
    - **X AXIS** = 13
    - **Y AXIS** = 13

8- Experimental Results

This section will study all the results and it will be divided into three major parts. Firstly each problem's result will be showed by tables including the position of each department within the plant, the distance between each pair of departments once every department is located in their optimized position and the total handling cost between each department. With all this information the user can compare the results with the input sent easily and faster. Secondly, statistics of each problem will be showed in the report with information such as solving time, iterations made and number of constraints and variables. Lastly, one of the most interesting parts of this report, there will be a table comparing DULLE's results with some of the papers already published and mentioned before. It is then the most challenging but also exciting part of the report as good results are expected from DULLE's performance.

Moving forward to the first part of this section, below are showed all the technical results for the problem with 8 departments. As stated several times throughout this paper it is important to remember how DULLE draws and interacts with the plant layout. The center of coordinates is always in the upper left corner of the grid, and the "x" axis increases when we move to the right whereas the "y" axis increases when we move downwards. Here is a quick and visual drawing of the plant to keep it in mind:

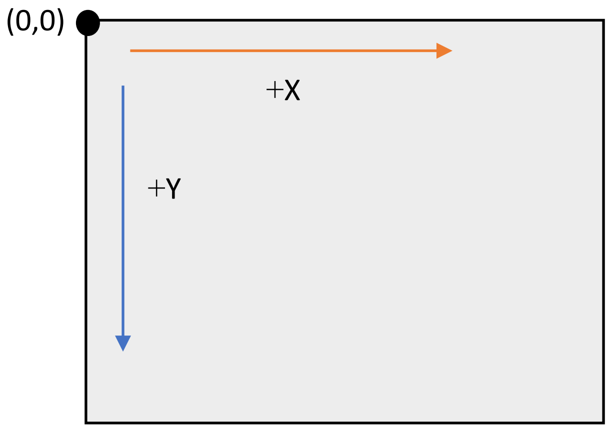


Figure 10: Center of coordinates explanation

As it is an important factor when solving any problem, the computer used to run DULLE to solve all these problems is a 2015 Macbook Pro with a 2.7 GHz Intel Core i5. As it is a four-year-old laptop, better results could be achieved with a more powerful computer.

- **<u>PROBLEM 1</u> - (8 DEPARTMENTS)**

First, solving the Problem 1 which has eight departments to optimize, DULLE obtained the following results:

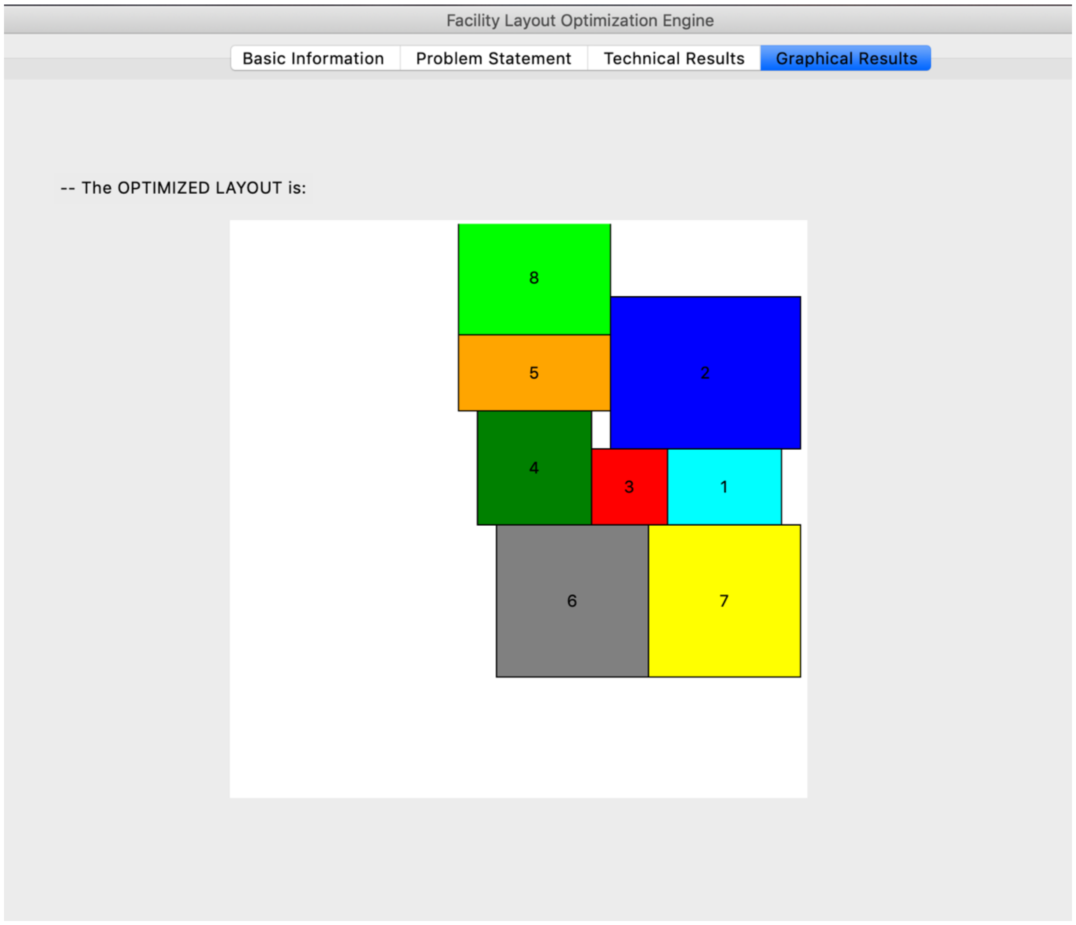
DEPARTMENT LOCATIONS	X AXIS	Y AXIS
1	13	7
2	12.5	4
3	10.5	7
4	8	6.5
5	8	4
6	9	10
7	13	10
8	8	1.5

Table 18: Department locations in Problem 1

DISTANCE BETWEEN DEPARTMENTS	1	2	3	4	5	6	7	8
1	0	3.5	2.5	5.5	8	7	3	10.5
2	3.5	0	5	7	4.5	9.5	6.5	7
3	2.5	5	0	3	5.5	4.5	5.5	8
4	5.5	7	3	0	2.5	4.5	8.5	5
5	8	4.5	5.5	2.5	0	7	11	2.5
6	7	9.5	4.5	4.5	7	0	4	9.5
7	3	6.5	5.5	8.5	11	4	0	13.5
8	10.5	7	8	5	2.5	9.5	13.5	0

Table 19: Distance between departments in Problem 1

The



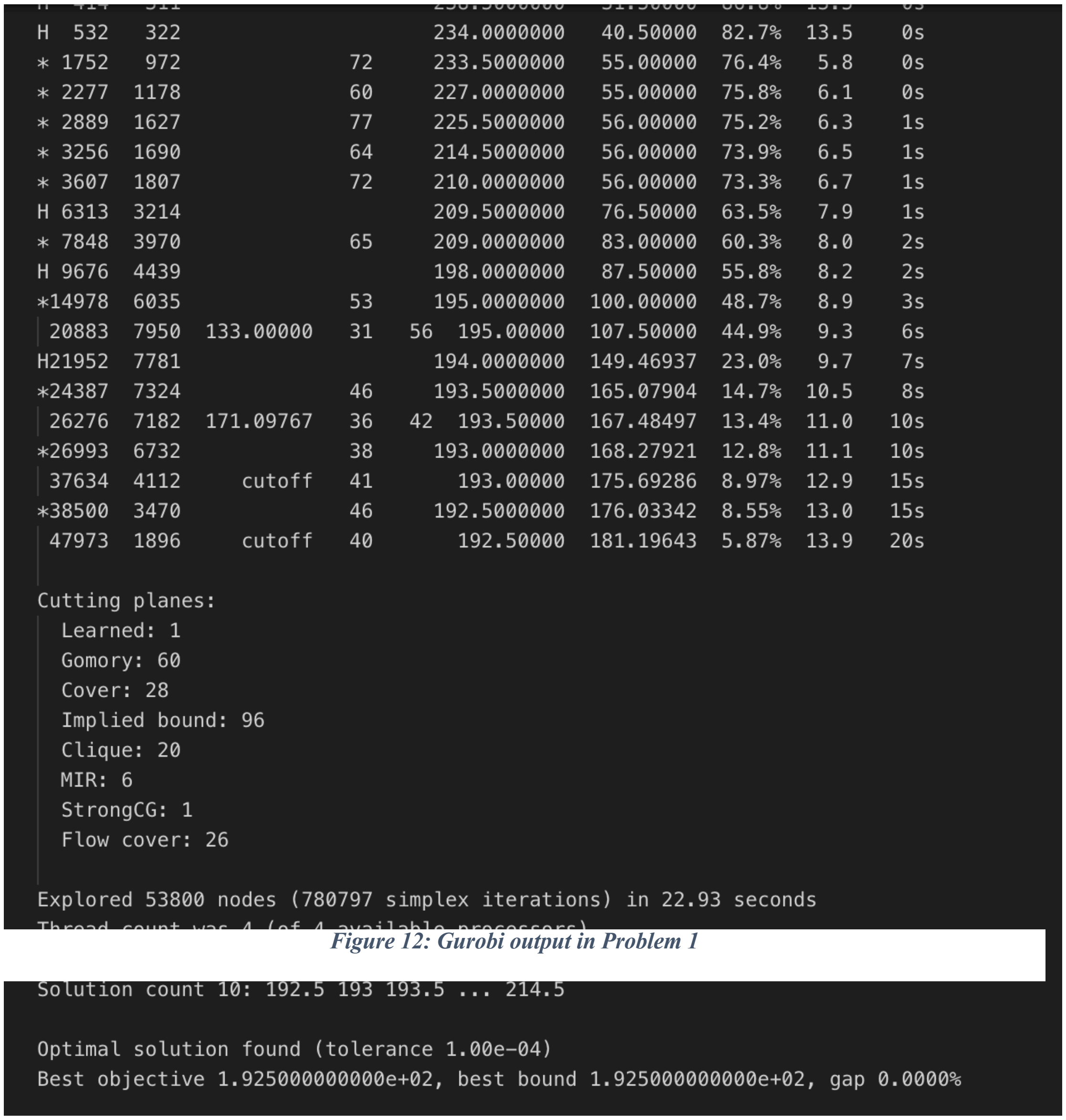
-- The OPTIMIZED LAYOUT is:

resulting **LAYOUT** of the **FACILITY** is:

Figure 11: Final layout in Problem 1

The resulting **TOTAL COST** of the **LAYOUT** is = **192.5**

The console output of the gurobi module can also be saved to have more information on how the DULLE handled the solving procedure.



```
H  532   322                     234.0000000   40.50000  82.7%  13.5    0s
* 1752   972             72      233.5000000   55.00000  76.4%   5.8    0s
* 2277  1178             60      227.0000000   55.00000  75.8%   6.1    0s
* 2889  1627             77      225.5000000   56.00000  75.2%   6.3    1s
* 3256  1690             64      214.5000000   56.00000  73.9%   6.5    1s
* 3607  1807             72      210.0000000   56.00000  73.3%   6.7    1s
H 6313  3214                     209.5000000   76.50000  63.5%   7.9    1s
* 7848  3970             65      209.0000000   83.00000  60.3%   8.0    2s
H 9676  4439                     198.0000000   87.50000  55.8%   8.2    2s
*14978  6035             53      195.0000000  100.00000  48.7%   8.9    3s
 20883  7950  133.00000  31   56  195.00000  107.50000  44.9%   9.3    6s
H21952  7781                     194.0000000  149.46937  23.0%   9.7    7s
*24387  7324             46      193.5000000  165.07904  14.7%  10.5    8s
 26276  7182  171.09767  36   42  193.50000  167.48497  13.4%  11.0   10s
*26993  6732             38      193.0000000  168.27921  12.8%  11.1   10s
 37634  4112     cutoff  41        193.00000  175.69286  8.97%  12.9   15s
*38500  3470             46      192.5000000  176.03342  8.55%  13.0   15s
 47973  1896     cutoff  40        192.50000  181.19643  5.87%  13.9   20s

Cutting planes:
  Learned: 1
  Gomory: 60
  Cover: 28
  Implied bound: 96
  Clique: 20
  MIR: 6
  StrongCG: 1
  Flow cover: 26

Explored 53800 nodes (780797 simplex iterations) in 22.93 seconds
Thread count was 4 (of 4 available processors)

Solution count 10: 192.5 193 193.5 ... 214.5

Optimal solution found (tolerance 1.00e-04)
Best objective 1.925000000000e+02, best bound 1.925000000000e+02, gap 0.0000%
```

Figure 12: Gurobi output in Problem 1

As shown in the figure above, the first objective function result was 214.5 in the first iteration which later on ended up being 192.5 after 780.797 iterations. The simulation took approximately 20 seconds, so DULLE solved Problem 1 really fast. It is also proven that the solution is optimal as the gap of the final value is 0 %, although this value will be later on discussed in other problems.

- **<u>PROBLEM 2</u> - (12 DEPARTMENTS OF EQUAL AREA)**

After solving Problem 2 which has 12 departments of equal are size but different handling cost to optimize DULLE obtained the following results:

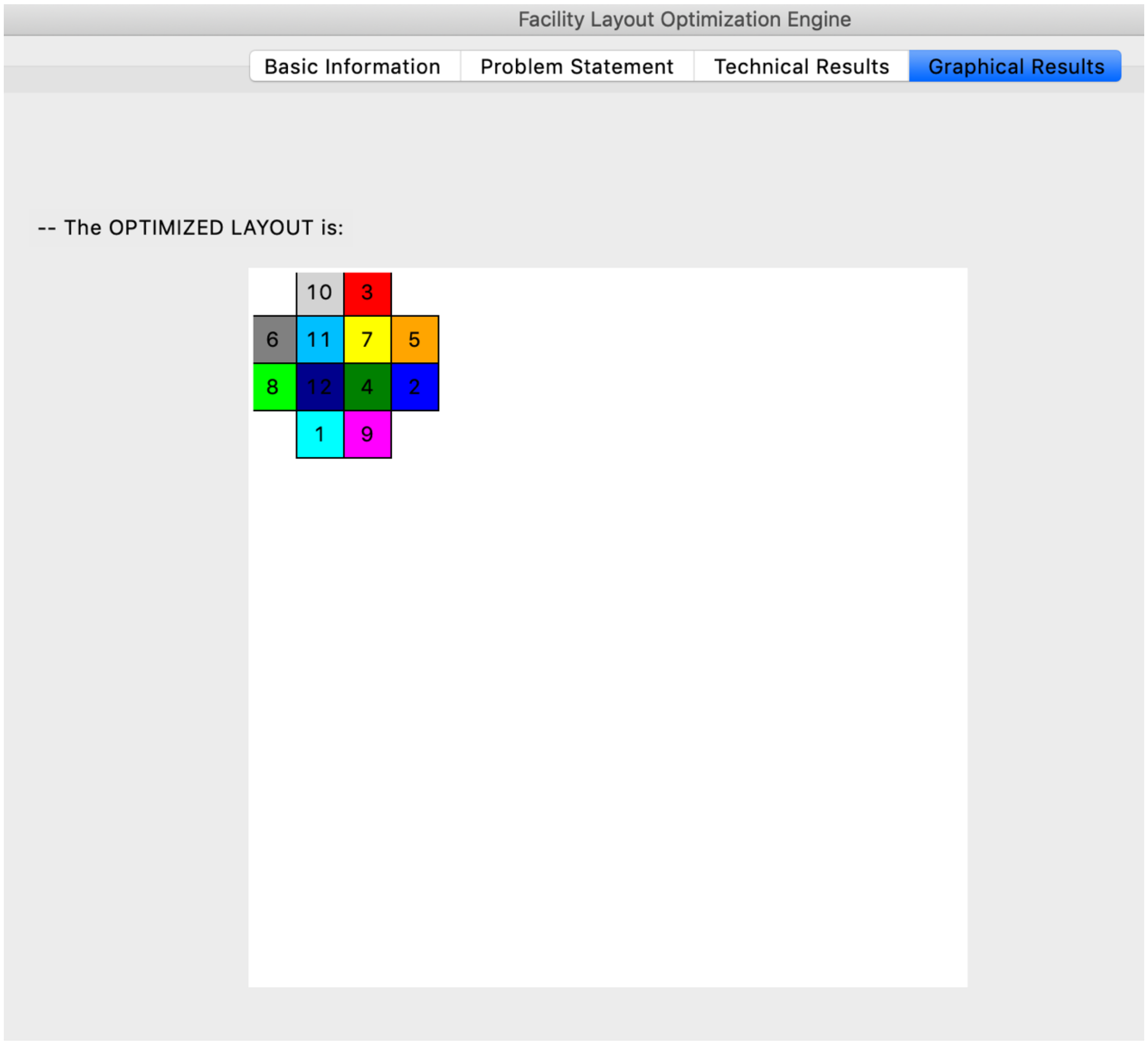
DEPARTMENT LOCATIONS	X AXIS	Y AXIS
1	1.5	3.5
2	3.5	2.5
3	2.5.5	0.5
4	2.5	2.5
5	3.5	1.5
6	0.5	1.5
7	2.5	1.5
8	0.5	2.5
9	2.5	3.5
10	1.5	0.5
11	1.5	1.5
12	1.5	2.5

Table 20: Department locations in Problem 2

DISTANCE BETWEEN DEPARTMENTS	1	2	3	4	5	6	7	8	9	10	11	12
1	0	3	4	2	4	3	3	2	1	3	2	1
2	3	0	3	1	1	4	2	3	2	4	3	2
3	4	3	0	2	2	3	1	4	3	1	2	3
4	2	1	2	0	2	3	1	2	1	3	2	1
5	4	1	2	2	0	3	1	4	3	3	2	3
6	3	4	3	3	3	0	2	1	4	2	1	2
7	3	2	1	1	1	2	0	3	2	2	1	2
8	2	3	4	2	4	1	3	0	3	3	2	1
9	1	2	3	1	3	4	2	3	0	4	3	2
10	3	4	1	3	3	2	2	3	4	0	1	2
11	2	3	2	2	2	1	1	2	3	1	0	1
12	1	2	3	1	3	2	2	1	2	2	1	0

Table 21: Distance between departments in Problem 2

The resulting **LAYOUT** of the **FACILITY** is:



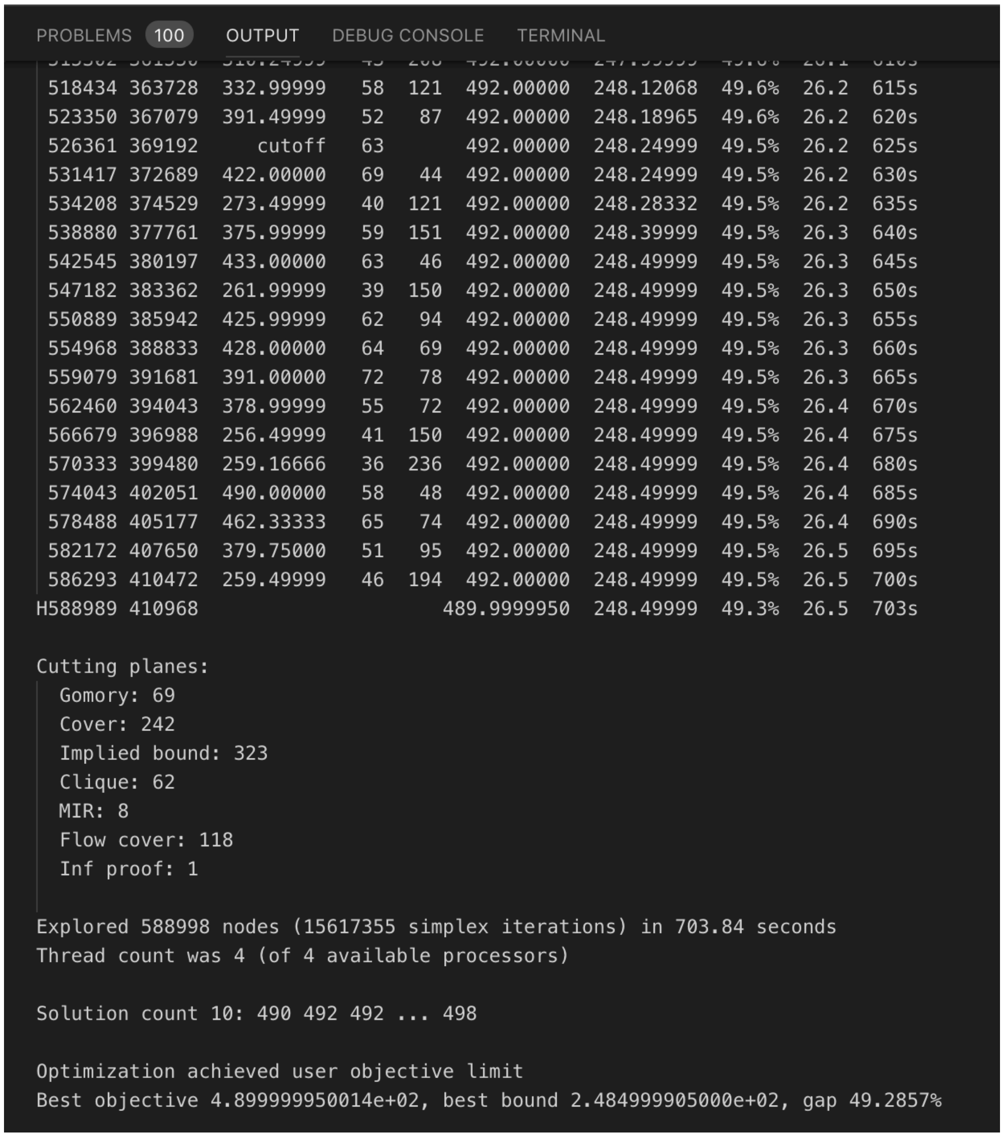
-- The OPTIMIZED LAYOUT is:

	10	3	
6	11	7	5
8	12	4	2
	1	9	

Table 22: Final layout in Problem 2

The resulting **TOTAL COST** of the **LAYOUT** is = **490**

The output stored from the gurobi console was:



```
 518434 363728  332.99999   58  121  492.00000  248.12068  49.6%  26.2  615s
 523350 367079  391.49999   52   87  492.00000  248.18965  49.6%  26.2  620s
 526361 369192     cutoff   63       492.00000  248.24999  49.5%  26.2  625s
 531417 372689  422.00000   69   44  492.00000  248.24999  49.5%  26.2  630s
 534208 374529  273.49999   40  121  492.00000  248.28332  49.5%  26.2  635s
 538880 377761  375.99999   59  151  492.00000  248.39999  49.5%  26.3  640s
 542545 380197  433.00000   63   46  492.00000  248.49999  49.5%  26.3  645s
 547182 383362  261.99999   39  150  492.00000  248.49999  49.5%  26.3  650s
 550889 385942  425.99999   62   94  492.00000  248.49999  49.5%  26.3  655s
 554968 388833  428.00000   64   69  492.00000  248.49999  49.5%  26.3  660s
 559079 391681  391.00000   72   78  492.00000  248.49999  49.5%  26.3  665s
 562460 394043  378.99999   55   72  492.00000  248.49999  49.5%  26.4  670s
 566679 396988  256.49999   41  150  492.00000  248.49999  49.5%  26.4  675s
 570333 399480  259.16666   36  236  492.00000  248.49999  49.5%  26.4  680s
 574043 402051  490.00000   58   48  492.00000  248.49999  49.5%  26.4  685s
 578488 405177  462.33333   65   74  492.00000  248.49999  49.5%  26.4  690s
 582172 407650  379.75000   51   95  492.00000  248.49999  49.5%  26.5  695s
 586293 410472  259.49999   46  194  492.00000  248.49999  49.5%  26.5  700s
H588989 410968                     489.9999950  248.49999  49.3%  26.5  703s

Cutting planes:
  Gomory: 69
  Cover: 242
  Implied bound: 323
  Clique: 62
  MIR: 8
  Flow cover: 118
  Inf proof: 1

Explored 588998 nodes (15617355 simplex iterations) in 703.84 seconds
Thread count was 4 (of 4 available processors)

Solution count 10: 490 492 492 ... 498

Optimization achieved user objective limit
Best objective 4.899999950014e+02, best bound 2.484999905000e+02, gap 49.2857%
```

Figure 13: Gurobi output in Problem 2

It can be seen that both the time and the gap for the solution are high, but it does not match the high-quality result. One again, the objective value was optimized correctly, and the result is even better than those of the other researches. For that reason, it is understood that his solution is a local optimal value and that it could be improved with a better and more powerful computer.

- **PROBLEM 3 - (11 DEPARTMENTS)**

Stepping up the difficulty of the problems solved, DULLE solved Problem 3 which has 11 departments of unequal area sizes. The results obtained by DULLE are the following:

DEPARTMENT LOCATIONS	X AXIS	Y AXIS
1	8	12.4
2	8	8.5
3	8	7.5
4	4.5	5
5	6	7.5
6	8	9.7
7	13.3	7.5
8	8	5.7
9	10.4	7
10	8	2
11	2.5	8.5

Table 23: Department locations in Problem 3

DISTANCE BETWEEN DEPARTMENTS	1	2	3	4	5	6	7	8	9	10	11
1	0	3.9	4.9	10.9	6.9	2.7	10.2	6.7	7.8	10.4	9.4
2	3.9	0	1	7	3	1.2	6.3	2.8	3.9	6.5	5.5
3	4.9	1	0	6	2	2.2	5.3	1.8	2.9	5.5	6.5
4	10.9	7	6	0	4	8.2	11.3	4.2	7.9	6.5	5.5
5	6.9	3	2	4	0	4.2	7.3	3.8	4.9	7.5	4.5
6	2.7	1.2	2.2	8.2	4.2	0	7.5	4	5.1	7.7	6.7
7	10.2	6.3	5.3	11.3	7.3	7.5	0	7.1	3.4	10.8	11.8
8	6.7	2.8	1.8	4.2	3.8	4	7.1	0	3.7	3.7	8.3
9	7.8	3.9	2.9	7.9	4.9	5.1	3.4	3.7	0	7.4	9.4
10	10.4	6.5	5.5	6.5	7.5	7.7	10.8	3.7	7.4	0	12
11	9.4	5.5	6.5	5.5	4.5	6.7	11.8	8.3	9.4	12	0

Table 24: Distance between departments in Problem 3

The resulting **LAYOUT** of the **FACILITY** is:

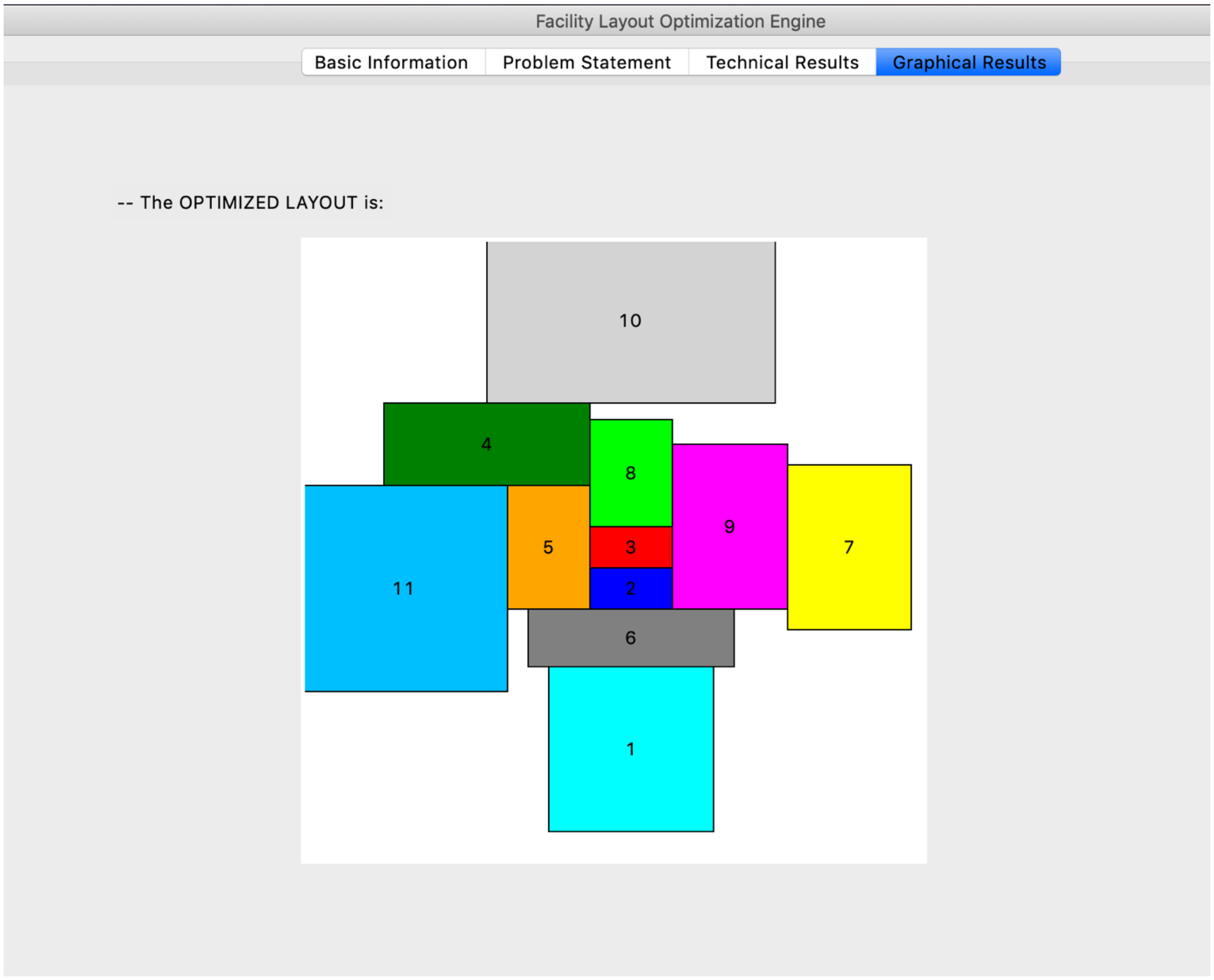
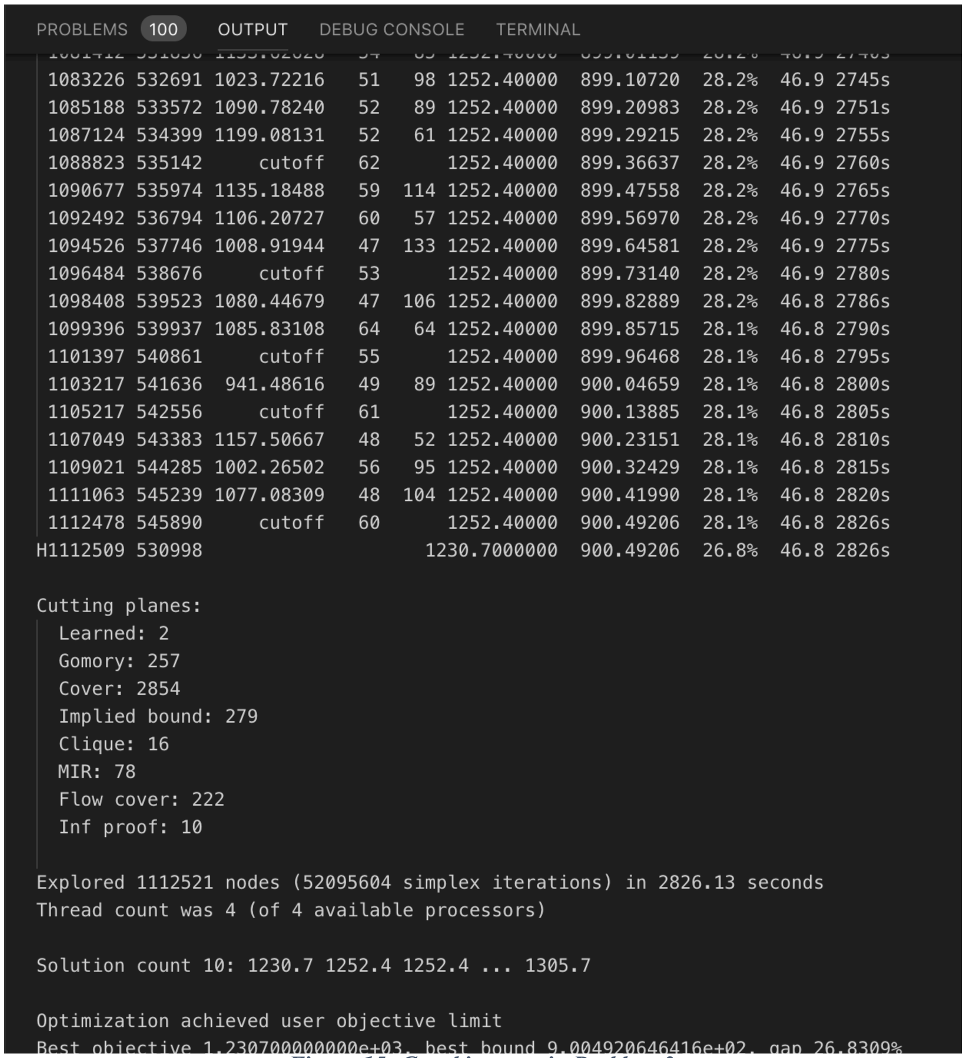


Figure 14: Final layout in Problem 3

The resulting **TOTAL COST** of the **LAYOUT** is = **1230.7**

Once again, gurobi saved the following information after running the problem:



```
  1083226 532691 1023.72216   51   98 1252.40000  899.10720  28.2%  46.9 2745s
  1085188 533572 1090.78240   52   89 1252.40000  899.20983  28.2%  46.9 2751s
  1087124 534399 1199.08131   52   61 1252.40000  899.29215  28.2%  46.9 2755s
  1088823 535142     cutoff   62      1252.40000  899.36637  28.2%  46.9 2760s
  1090677 535974 1135.18488   59  114 1252.40000  899.47558  28.2%  46.9 2765s
  1092492 536794 1106.20727   60   57 1252.40000  899.56970  28.2%  46.9 2770s
  1094526 537746 1008.91944   47  133 1252.40000  899.64581  28.2%  46.9 2775s
  1096484 538676     cutoff   53      1252.40000  899.73140  28.2%  46.9 2780s
  1098408 539523 1080.44679   47  106 1252.40000  899.82889  28.2%  46.8 2786s
  1099396 539937 1085.83108   64   64 1252.40000  899.85715  28.1%  46.8 2790s
  1101397 540861     cutoff   55      1252.40000  899.96468  28.1%  46.8 2795s
  1103217 541636  941.48616   49   89 1252.40000  900.04659  28.1%  46.8 2800s
  1105217 542556     cutoff   61      1252.40000  900.13885  28.1%  46.8 2805s
  1107049 543383 1157.50667   48   52 1252.40000  900.23151  28.1%  46.8 2810s
  1109021 544285 1002.26502   56   95 1252.40000  900.32429  28.1%  46.8 2815s
  1111063 545239 1077.08309   48  104 1252.40000  900.41990  28.1%  46.8 2820s
  1112478 545890     cutoff   60      1252.40000  900.49206  28.1%  46.8 2826s
H1112509 530998                    1230.7000000  900.49206  26.8%  46.8 2826s

Cutting planes:
  Learned: 2
  Gomory: 257
  Cover: 2854
  Implied bound: 279
  Clique: 16
  MIR: 78
  Flow cover: 222
  Inf proof: 10

Explored 1112521 nodes (52095604 simplex iterations) in 2826.13 seconds
Thread count was 4 (of 4 available processors)

Solution count 10: 1230.7 1252.4 1252.4 ... 1305.7

Optimization achieved user objective limit
Best objective 1.230700000000e+03, best bound 9.004920646416e+02, gap 26.8309%
```

Figure 15: Gurobi output in Problem 3

This time the optimal value is 1030.7 units and was achieved after running the problem for 2826 seconds. Although the time is not low, the result is optimized correctly as it will be checked in the summary of results later on. One again, the higher gap value and solving time are highly related to the lack of power of the computer. This factor will also be crucial in Problem 4, which is the most power-demanding problem. Even though these values are not perfect, the optimized layout is good, and the optimization level is really high.

- **<u>PROBLEM 4</u> - (20 DEPARTMENTS)**

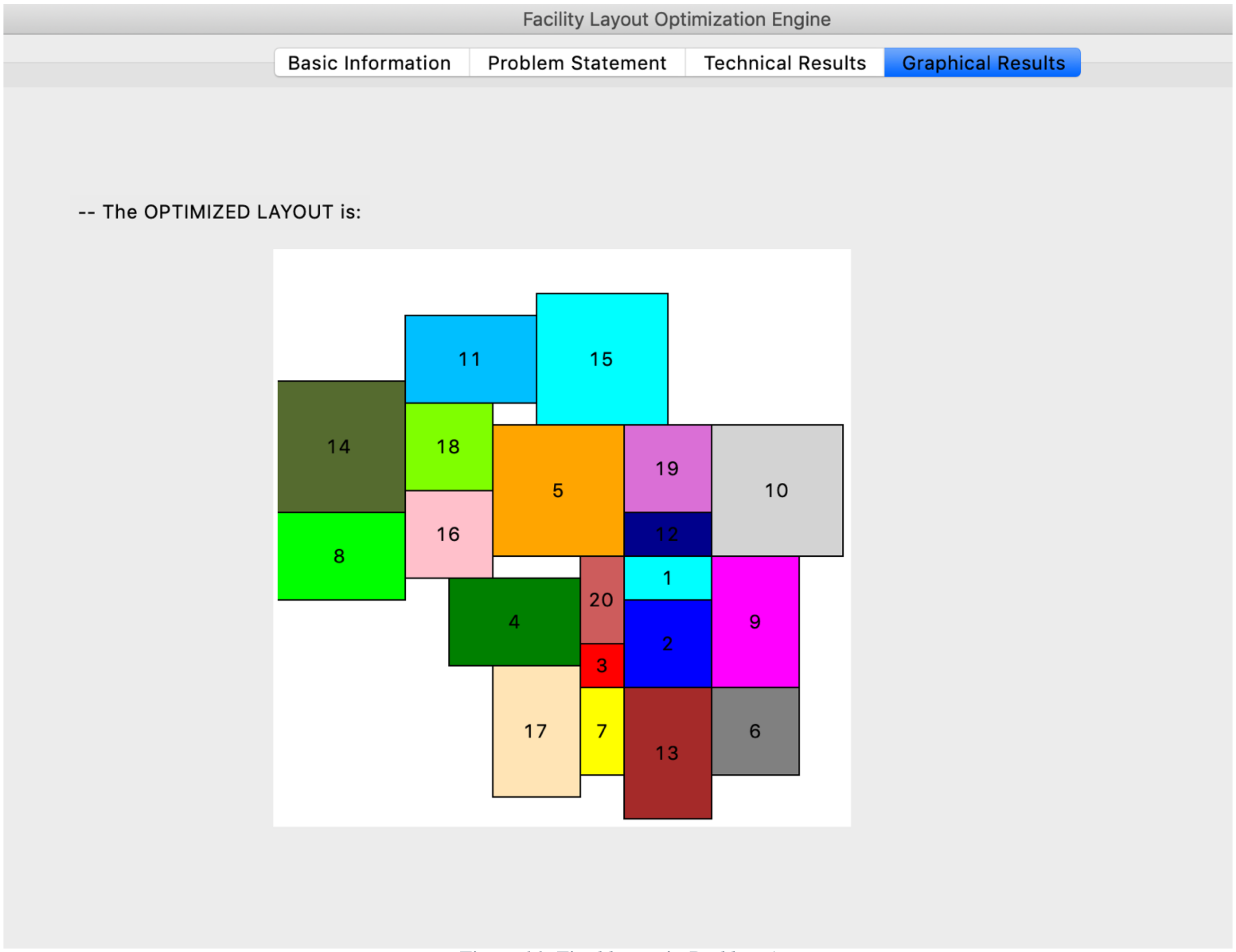
DEPARTMENTS POSITIONS	X AXIS	Y AXIS
1	9	7.5
2	9	9
3	7.5	9.5
4	5.5	8.5
5	6.5	5.5
6	11	11
7	7.5	11
8	1.5	7
9	11	8.5
10	11.5	5.5
11	4.5	2.5
12	9	6.5
13	9	11.5
14	1.5	4.5
15	7.5	2.5
16	4	6.5
17	6	11
18	4	4.5
19	9	5
20	7.5	8

Table 25: Department positions in Problem 4

DISTANCE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1.5	3.5	4.5	4.5	5.5	5	8	3	4.5	9.5	1	4	10.5	6.5	6	6.5	8	2.5	2
2	1.5	0	2	4	6	4	3.5	9.5	2.5	6	11	2.5	2.5	2.5	8	7.5	5	9.5	4	2.5
3	3.5	2	0	3	5	4	1.5	8.5	4.5	8	10	4.5	3.5	3.5	7	6.5	3	8.5	6	1.5
4	4.5	4	3	0	4	8	4.5	5.5	5.5	9	7	5.5	6.5	6.5	8	3.5	3	5.5	7	2.5
5	4.5	6	5	4	0	10	6.5	6.5	7.5	5	5	3.5	8.5	8.5	4	3.5	6	3.5	3	3.5
6	5.5	4	4	8	10	0	3.5	13.5	2.5	6	15	6.5	2.5	2.5	12	11.5	5	13.5	8	6.5
7	5	3.5	1.5	4.5	6.5	3.5	0	10	6	9.5	11.5	6	2	2	8.5	8	1.5	10	7.5	3
8	8	9.5	8.5	5.5	6.5	13.5	10	0	11	11.5	7.5	8	12	12	10.5	3	8.5	5	9.5	7
9	3	2.5	4.5	5.5	7.5	2.5	6	11	0	3.5	12.5	4	5	5	9.5	9	7.5	11	5.5	4
10	4.5	6	8	9	5	6	9.5	11.5	3.1	0.4	9.6	3.1	8.1	8.1	6.6	8.1	10.6	8.1	2.6	6.1
11	9.5	11	10	7	5	15	11.5	7.5	12.5	10	0	8.5	13.5	13.5	3	4.5	10	2.5	7	8.5
12	1	2.5	4.5	5.5	3.5	6.5	6	8	4	3.5	8.5	0	5	5	5.5	5	7.5	7	1.5	3
13	4	2.5	3.5	6.5	8.5	2.5	2	12	5	8.5	13.5	5	0	0	10.5	10	3.5	12	6.5	5
14	10.5	2.5	3.5	6.5	8.5	2.5	2	12	13.5	11	5	9.5	14.5	14.5	8	4.5	11	2.5	8	9.5
15	6.5	8	7	8	4	12	8.5	10.5	9.5	7	3	5.5	10.5	10.5	0	7.5	10	5.5	4	5.5
16	6	7.5	6.5	3.5	3.5	11.5	8	3	9	8.5	4.5	5	10	10	7.5	0	6.5	2	6.5	5
17	6.5	5	3	3	6	5	1.5	8.5	7.5	11	10	7.5	3.5	3.5	10	6.5	0	8.5	9	4.5
18	8	9.5	8.5	5.5	3.5	13.5	10	5	11	8.5	2.5	7	12	12	5.5	2	8.5	0	5.5	7
19	2.5	4	6	7	3	8	7.5	9.5	5.5	3	7	1.5	6.5	6.5	4	6.5	9	5.5	0	4.5
20	2	2.5	1.5	2.5	3.5	6.5	3	7	4	6.5	8.5	3	5	5	5.5	5	4.5	7	4.5	0

Table 26: Distance between departments in Problem 4

The resulting **LAYOUT** of the **FACILITY** is:

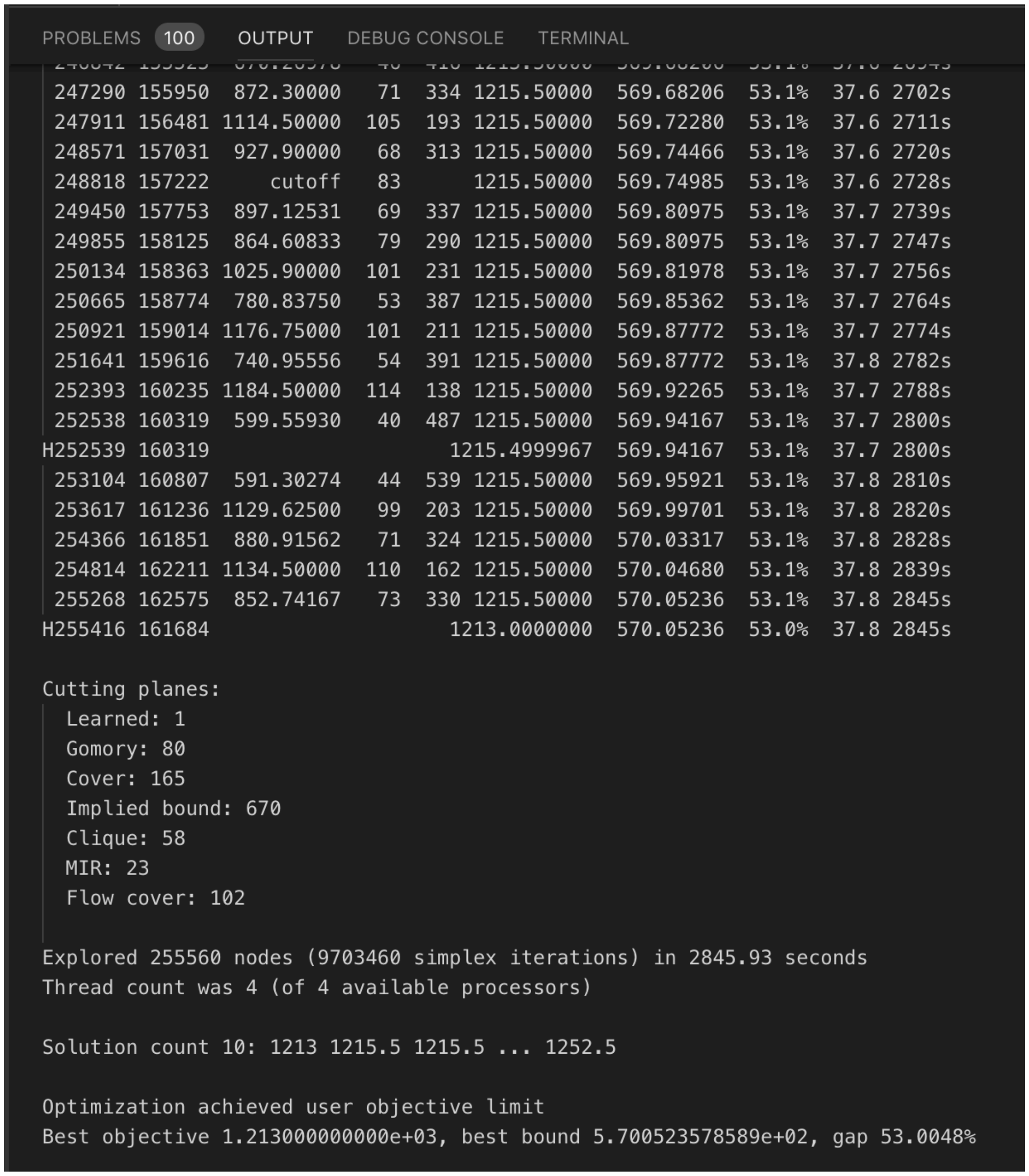


-- The OPTIMIZED LAYOUT is:

Figure 16: Final layout in Problem 4

The resulting **TOTAL COST** of the **LAYOUT** is = **1213**

The output stored from the gurobi console was:



```
 247290 155950  872.30000   71  334 1215.50000  569.68206  53.1%  37.6 2702s
 247911 156481 1114.50000  105  193 1215.50000  569.72280  53.1%  37.6 2711s
 248571 157031  927.90000   68  313 1215.50000  569.74466  53.1%  37.6 2720s
 248818 157222     cutoff   83      1215.50000  569.74985  53.1%  37.6 2728s
 249450 157753  897.12531   69  337 1215.50000  569.80975  53.1%  37.7 2739s
 249855 158125  864.60833   79  290 1215.50000  569.80975  53.1%  37.7 2747s
 250134 158363 1025.90000  101  231 1215.50000  569.81978  53.1%  37.7 2756s
 250665 158774  780.83750   53  387 1215.50000  569.85362  53.1%  37.7 2764s
 250921 159014 1176.75000  101  211 1215.50000  569.87772  53.1%  37.7 2774s
 251641 159616  740.95556   54  391 1215.50000  569.87772  53.1%  37.8 2782s
 252393 160235 1184.50000  114  138 1215.50000  569.92265  53.1%  37.7 2788s
 252538 160319  599.55930   40  487 1215.50000  569.94167  53.1%  37.7 2800s
H252539 160319                    1215.4999967  569.94167  53.1%  37.7 2800s
 253104 160807  591.30274   44  539 1215.50000  569.95921  53.1%  37.8 2810s
 253617 161236 1129.62500   99  203 1215.50000  569.99701  53.1%  37.8 2820s
 254366 161851  880.91562   71  324 1215.50000  570.03317  53.1%  37.8 2828s
 254814 162211 1134.50000  110  162 1215.50000  570.04680  53.1%  37.8 2839s
 255268 162575  852.74167   73  330 1215.50000  570.05236  53.1%  37.8 2845s
H255416 161684                    1213.0000000  570.05236  53.0%  37.8 2845s

Cutting planes:
  Learned: 1
  Gomory: 80
  Cover: 165
  Implied bound: 670
  Clique: 58
  MIR: 23
  Flow cover: 102

Explored 255560 nodes (9703460 simplex iterations) in 2845.93 seconds
Thread count was 4 (of 4 available processors)

Solution count 10: 1213 1215.5 1215.5 ... 1252.5

Optimization achieved user objective limit
Best objective 1.213000000000e+03, best bound 5.700523578589e+02, gap 53.0048%
```

Figure 17: Gurobi output in Problem 4

Once again, looking into the gurobi statistical output it can be proven that the solution encountered is 1213 for the total cost but the solving time raised up to 2845 seconds and the gap

ended at 53%. Although these values are quite high, as it can be seen in the next table, achieving a cost of 1213 for this problem is actually a really good result. Therefore, it is seen that even though DULLE performs way better in smaller problems, after running such a loaded and difficult problem like Problem 4, DULLE performed great and achieved a very reasonable minimized cost.

- **SUMMARY OF RESULTS**

MODEL	**PROBLEM SOLUTION**			
	PROBLEM 1	**PROBLEM 2**	**PROBLEM 3**	**PROBLEM 4**
DULLE	192.5	490	1230.7	1213
***[HASD17]* -** BEST	192.5	-	1222.8	1173.6
***[HASD17]* -** MEAN	192.25	-	1229.8	1191.9
***[HASD17]* -** WORST	192.53	-	1335.6	1219.5
[WONG15]	208.74	-	1335.6	1264.2
[IMAM93]	-	517.8	-	1333.8
[KHAR88]	-	541.07	-	-
[IMAM01]	-	502.36	-	-

9- Real-world implementation

After finishing the academic year 2018-2019 in University of Maryland, College Park, I started an internship in Muebles y Maderas Nueva Linea S.L in their Azpeita factory. Their main working product is parquet, with the firm Surco, they offer extreme high-quality parquet letting the client modify and work hand to hand with the design group to satisfy the customer needs. Muebles y Maderas Nueva Linea S.L is of Azcue Group, a really big company located in Azpeitia since 1933.

Figure 19: Muebles y Maderas Nueva Linea S.L Logo

Figure 18: Surco Parquet Logo

Since I entered the group, the most important task I have had is to implement and use DULLE in the plant. As the factory is already built and has been working for years now, my task is to design and locate several maintenance stations around the plant to ensure a secure and comfortable working environment alongside an optimal layout. Surco firm creates parquet trough a discrete process, giving the possibility to stop or increase the pace easily.

DULLE is now being optimized and edited to ensure that the task assigned gives an optimal solution. As the layout is already stablished, all the lines and machines located in the plant will be treated by DULLE as departments of fixed dimensions and with an initial location in both "X" and "Y" axis. There will be around six additional maintenance stations throughout the plant and these will be treated by DULLE as departments of fixed dimensions ready to be optimized and located.

Thanks to the power and facturation of the company, this project is economically really strong. Optimizing the maintance stations means that operators spend less time looking and waling for any needed tool. This decrease on spent time means time spent producing and creating value giving the fact that the procces is discrete, the line needs the operator to produce and it has to stop whenever the operator goes to the maintenance station.

Below is showed a factory plan where all the different production lines can be observed. For all of them, the six new maintenance stations will be optimized, having different amount of operators in each of the lines and different cost of operation and waiting time. All of it easily optimizable thanks to DULLE.

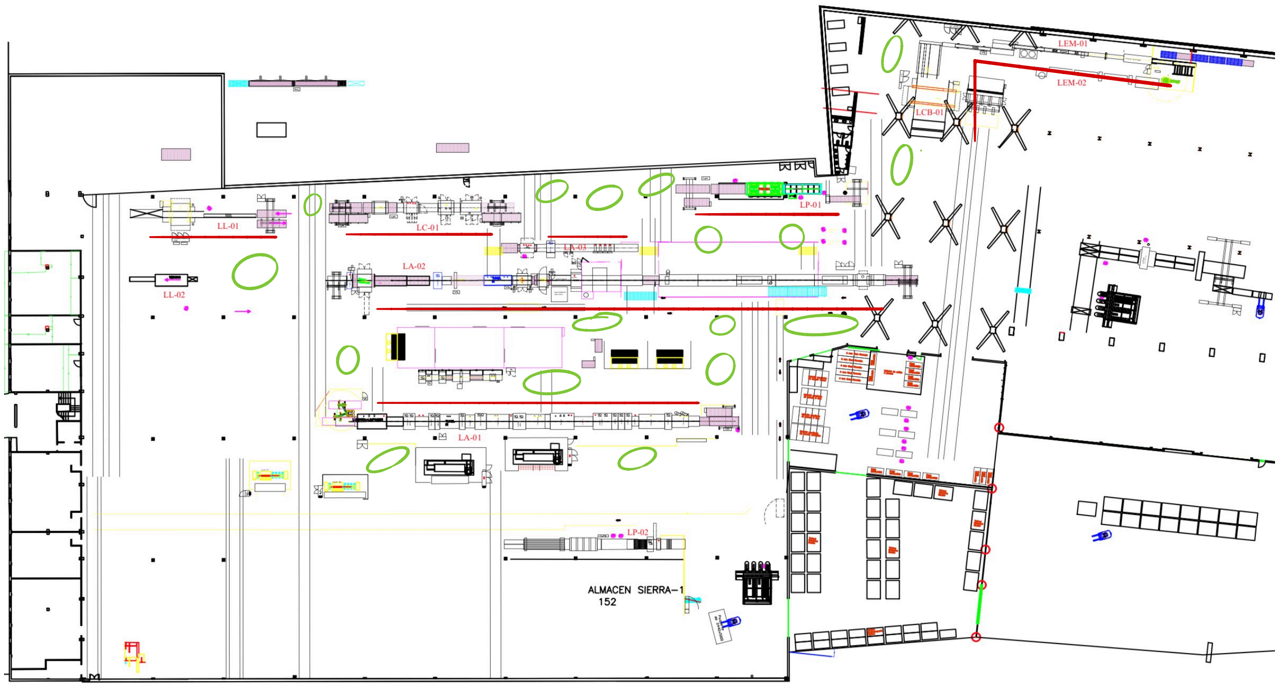


Figure 20: Muebles y Maderas Nueva Linea S.L Plant View

10- Conclusion

As it has been mentioned before in the briefly comment after each problem, there is a clear thought about DULLE: It is absolutely fast and accurate when solving low-loaded problems. It solved Problem 1 very quickly and achieving the same or sometimes a better solution than the methods proposed by other researchers. Problem 2 could also be considered as in the low-loaded problems side, although it has a lot of departments to arrange there is an advantage thanks to having all of them of equal size. One again, and even most noticeable here, DULLE obtained great results for Problem 2. It smashed the records compared with the other results posted by more than 10 units. Furthermore, as mentioned before, DULLE had to be stopped manually, as it was still trying to get a better solution. It is though that running DULLE in a more powerful and suitable computer would lead to better results.

If we pay more attention now to Problems 3 and 4, we can see that the performance of DULLE decreases with respect to the previous problems, but not due to the final results it gets as they are in the mean or even better compared to the rest of the results. The performance of DULLE decreases mainly because the bigger solving time. Once again, it is very likely that this decreased performance would be solved in a better computer, but that hasn't been tried yet. Even though the timing is not the best in these cases, the minimized cost value stills being really good and comparable with the other method's results. The simplicity and straight-forwardness of DULLE's optimization engine and its formulation needs to be remembered now, as it achieves these results having simpler and easier methods than other approaches.

DULLE's performance when handling big problems has been compromised in both problems three and four. As they have eleven and twenty departments respectively, the difficulty of the problems is really high. After solving them, the possibility of using genetic algorithms arises as they would give a better and more powerful tool to solve these difficult problems. As explained in Section 3, genetic algorithms are much more difficult to code and create. Furthermore, mixed integer programming (MIP), which is the method used in DULLE, is the solution that anyone would give to a simple optimization problem whereas none would use a genetic algorithm to solve it, as it is not as straight forward or as quick.

If the optimal solution was the only and needed achievement in this project a genetic algorithm would be more suitable for the already given reasons. Given the conditions and the knowledge of these problems, a mixed integer program was easier and more straightforward for both the creator and the users using the program. For that reason, DULLE is recommended to be only used in smaller problems with not a lot of data and variables. In case of bigger problems such as Problem 3 and Problem 4, a genetic algorithm program would achieve better results.

Nonmatter what the solutions for these problems were, DULLE is working properly so far in the real-world implementation. Creating a product that can then be used and can produce value, save money and improve the quality of an enormous company such as Muebles y Maderas Nueva Linea S.L. A feeling of proudness and also responsibility arises when performing this task and thanks to DULLE the working procedure and difficult parts are much easier and precise.

11- References

[HASD17]- *"Ranjan Kumar Hasda (2017). Contribution to the optimization of Unequal Area Rectangular Facility Layout Problem. Ecole Central de Nantes."*

[WONG15]- *"Asl, A.D., & Wong, K. Y. (2015). Solving unequal-area static and dynamic facility layout problems using modified particle swarm optimization. Journal of Intelligent Manufacturing".*

[IMAM93] - *"Imam, M. H., & Mir, M. 1993). Automated layout of facilities of unequal areas. Computers in Industrial Engineering, 24.3), 355 ± 366. "*

[KHAR88] - *"Khare, V. K., Khare, M. K., & Neema, M. L. (1988). Combined computer-aided approach for the facilities design problem and estimation of the distribution parameter in the case of multigoal optimization. Computers in Industrial Engineering, 14, 465 ± 476."*

[IMAM01] - *"M. Mir & M.H. Imam (2001). A hybrid optimization approach for layout design of unequal-area facilities. College of Engineering, Umm Al-Qura University, P.O. Box 5450, Makkah, Saudi Arabia"*

[MONT91] - *"Montreuil, B. (1991). A modeling framework for integrating layout design and flow network design."*

[GILM63] - *"Gilmore, P.C., & Gomory, R.E. (1963). A linear approach to the cutting stock problem . Part II. "*

[KUSI87] - *"Kusiak, A., & Heragu, S. S. (1987). The facility layout problem. European Journal of Operational Research."*

[KOOP57] - *"Koopmans, T. C & Beckmann, M. (1957). Assignment problems and the location of economic activities. "*

[TOMP96] - *"Tompkins, James A (1996). Facilities planning (2nd ed). John Wiley, New York"*

https://en.wikipedia.org/wiki/Taxicab_geometry

12- Appendix

```
import tkinter.ttk as ttk
from tkinter import *
from gurobipy import *
from threading import Thread
from tkinter import Frame
from tkinter.font import Font
import csv
import codecs
from IPython.display import Markdown, display


roota = Tk() # Root for the window
roota.geometry("2000x2000")
roota.configure(background="white")

roota.title("Facility Layout Optimization Engine")

nb = ttk.Notebook(roota) # Create and label all the tabs

first = ttk.Frame(roota)
second= ttk.Frame(roota)
third= ttk.Frame(roota)
fourth = ttk.Frame(roota)

nb.add(first, text='Basic Information' ,)
nb.add(second, text='Problem Statement' , )
nb.add(third, text='Technical Results')
nb.add(fourth, text='Graphical Results')

nb.pack(expand="1" , fill="both")

canvas  = Canvas(first)
canvas2 = Canvas(second)


class ReadFileTrips():
    def __init__ (self, filename):
        with open (filename, 'r', encoding='utf-8') as f_input:
            csv_input = csv.reader(f_input)
            self.trips = list(csv_input)

    def GetTrips (self, row , column):
        return self.trips[row-1][column-1]

class ReadFileProduction():
    def __init__ (self, filename):
        with open (filename, 'r', encoding='utf-8') as f_input:
            csv_input = csv.reader(f_input)
            self.production = list(csv_input)

    def GetProduction (self, row , column):
        return self.production[row-1][column-1]

class ReadFileDimensions():
    def __init__ (self, filename):
        with open (filename, 'r', encoding='utf-8') as f_input:
            csv_input = csv.reader(f_input)
            self.dimensions = list(csv_input)

    def GetWidth (self):
        return self.dimensions[row][1]

    def GetBreadth (self,row):
        return self.dimensions[row][2]

model = Model('Dulle')

C = {}
Trips = {}
Workers = {}
Speed = {}
W = {}
B = {}
x = {}
y = {}
s1 = {}
s2 = {}
s3 = {}
s4 = {}
Dx = {}
Dy = {}
Dxabs = {}
Dyabs = {}
Dij = {}

FacilityInfo = ReadFileTrips('Facility Information.csv')

rows = len(FacilityInfo.trips)
NumDepts = int((rows-4)/3)

for row in range(1,NumDepts+1):
    for column in range(1,NumDepts+1):
        Trips[row,column] = int(FacilityInfo.trips[row][column])
```

```
# for row in range(NumDepts+2,2*NumDepts+2):
#       for column in range(1,NumDepts+1):
#             Workers[row-NumDepts-1,column] = int(FacilityInfo.trips[row][column])

# for row in range(2*NumDepts+3,3*NumDepts+3):
#       for column in range(1,NumDepts+1):
#             Speed[row-2*NumDepts-2,column] = int(FacilityInfo.trips[row][column])

# for row in range(1,NumDepts+1):
#       for column in range(1,NumDepts+1):
#             C[row,column] = Trips[row,column] * Workers[row, column] * Speed [row,column]

for row in range(1,NumDepts+1):
      for column in range(1,NumDepts+1):
            C[row,column] = Trips[row,column]

xUpper = int(FacilityInfo.trips[rows-1][1])
yUpper = int(FacilityInfo.trips[rows-1][3])

DepartmentsInfo = ReadFileDimensions('Department Parameters.csv')

for row in range(1,NumDepts+1):
      W[row] = float(DepartmentsInfo.dimensions[row][1])
      B[row] = float(DepartmentsInfo.dimensions[row][2])

d_label = Label (first, text = " Welcome to ",fg  = 'black', bg="gray92" , font = ("Average",
20)).pack(side="top")

d_label0 = Label (first, text = " DULLE - FACILITY LAYOUT OPTIMIZATION ENGINE",fg  = 'navy',
bg="gray92" , font = ("Average", 25)).pack(side="top")

d_label1= Label(first , text = "- INTRUCTIONS to use DULLE: " , bg="gray92").place(relx=0.10,rely=0.20)
d_label2= Label(first , text = "   1- Fill up both 'Facility Information' and 'Departments Parameters' csv
files with your facility data " , bg="gray92").place(relx=0.10,rely=0.23)
d_label3= Label(first , text = "   2- Save both CSV files correctly " ,
bg="gray92").place(relx=0.10,rely=0.26)
d_label4= Label(first , text = "   3- Run DULLE and let the boss do his work " ,
bg="gray92").place(relx=0.10,rely=0.29)

d_label5= Label(first , text = "- DULLE data VERIFICATION: " , bg="gray92").place(relx=0.10,rely=0.35)
d_label7= Label(first , text = " DULLE has a page called 'PROBLEM STATEMENT' where all the info
imported from the CSV files can be checked in case there was an error" ,
bg="gray92").place(relx=0.15,rely=0.38)
d_label8= Label(first , text = " In any error is found, the user may stop DULLE, update all the info in the
CSV files and run DULLE again " , fg="red", bg="gray92" , font =("Average",
11)).place(relx=0.15,rely=0.41)

d_label5= Label(first , text = "- DULLE RESULTS STUDY: " , bg="gray92").place(relx=0.10,rely=0.46)
```

```
d_label6= Label(first , text = " 1- DULLE gives you TWO different pages of results:   " ,
bg="gray92").place(relx=0.15,rely=0.49)
d_label7= Label(first , text = " 1- TECHNICAL RESULTS: Here all the numerical results suchs as distance
betwween departments and total minimized distances are displayed " , font =("Average", 11),
bg="gray92").place(relx=0.20,rely=0.52)
d_label8= Label(first , text = " 2- GRAPHIC RESULTS: Here the displacement of all the departmenst is
shown alongside with the total minimez cost of the plant " , font =("Average",
11),bg="gray92").place(relx=0.20,rely=0.55)


label_problem= Label(second , text = "-- PROBLEM STATEMENT:  Optimize the layout of a production
plant with " + str(NumDepts) + " departments given the following information and data read from the
csv file: " , bg="gray92").place(relx=0.10,rely=0.05)
label_problem= Label(second , text =
"________________________________________________________________________________
_________________________________________________________________________________
_________________________________" , bg="gray92").place(relx=0.05,rely=0.08)


##--------------------------------------- COSTS

label_problem_C0= Label(second , text = "- COSTS between departments are: " ,
bg="gray92").place(relx=0.12,rely=0.12)
label_problem_C1= Label(second , text = "Cost between departments 1 and 2 is =  " + str(C[1,2]) ,
bg="gray92").place(relx=0.15,rely=0.16)


if (NumDepts>2):
    label_problem_C2= Label(second , text = "Cost between departments 1 and 3 is =  " + str(C[1,3]) ,
bg="gray92").place(relx=0.15,rely=0.19)
    label_problem_C3= Label(second , text = "Cost between departments 2 and 3 is =  " + str(C[2,3]) ,
bg="gray92").place(relx=0.15,rely=0.22)

    if (NumDepts>3):
        label_problem_C4= Label(second , text = "Cost between departments 1 and 4 is =  " +
str(C[1,4]) , bg="gray92").place(relx=0.15,rely=0.25)
        label_problem_C5= Label(second , text = "Cost between departments 2 and 4 is =  " +
str(C[2,4]) , bg="gray92").place(relx=0.15,rely=0.28)
        label_problem_C6= Label(second , text = "Cost between departments 3 and 4 is =  " +
str(C[3,4]) , bg="gray92").place(relx=0.15,rely=0.31)

        if (NumDepts>4):
            label_problem_C7= Label(second , text = "Cost between departments 1 and 5 is =  " +
str(C[1,5]) , bg="gray92").place(relx=0.15,rely=0.34)
            label_problem_C8= Label(second , text = "Cost between departments 2 and 5 is =  " +
str(C[2,5]) , bg="gray92").place(relx=0.15,rely=0.37)
            label_problem_C9= Label(second , text = "Cost between departments 3 and 5 is =  " +
str(C[3,5]) , bg="gray92").place(relx=0.15,rely=0.40)
```

```
                    label_problem_C10= Label(second , text = "Cost between departments 4 and 5 is =  " +
str(C[4,5]) , bg="gray92").place(relx=0.15,rely=0.43)

                    if (NumDepts>5):
                        label_problem_C11= Label(second , text = "Cost between departments 1 and 6 is =  " +
str(C[1,6]) , bg="gray92").place(relx=0.15,rely=0.46)
                        label_problem_C12= Label(second , text = "Cost between departments 2 and 6 is =  " +
str(C[2,6]) , bg="gray92").place(relx=0.15,rely=0.49)
                        label_problem_C13= Label(second , text = "Cost between departments 3 and 6 is =  " +
str(C[3,6]) , bg="gray92").place(relx=0.15,rely=0.52)
                        label_problem_C14= Label(second , text = "Cost between departments 4 and 6 is =  " +
str(C[4,6]) , bg="gray92").place(relx=0.15,rely=0.55)
                        label_problem_C15= Label(second , text = "Cost between departments 5 and 6 is =  " +
str(C[5,6]) , bg="gray92").place(relx=0.15,rely=0.58)

                        if (NumDepts>6):
                            label_problem_C16= Label(second , text = "Cost between departments 1 and 7 is
=  " + str(C[1,7]) , bg="gray92").place(relx=0.15,rely=0.61)
                            label_problem_C17= Label(second , text = "Cost between departments 2 and 7 is
=  " + str(C[2,7]) , bg="gray92").place(relx=0.15,rely=0.64)
                            label_problem_C18= Label(second , text = "Cost between departments 3 and 7 is
=  " + str(C[3,7]) , bg="gray92").place(relx=0.15,rely=0.67)
                            label_problem_C19= Label(second , text = "Cost between departments 4 and 7 is
=  " + str(C[4,7]) , bg="gray92").place(relx=0.15,rely=0.70)
                            label_problem_C20= Label(second , text = "Cost between departments 5 and 7 is
=  " + str(C[5,7]) , bg="gray92").place(relx=0.15,rely=0.73)
                            label_problem_C21= Label(second , text = "Cost between departments 6 and 7 is
=  " + str(C[6,7]) , bg="gray92").place(relx=0.15,rely=0.76)

                            if (NumDepts>7):
                                label_problem_C22= Label(second , text = "Cost between departments 1 and
8 is =  " + str(C[1,8]) , bg="gray92").place(relx=0.15,rely=0.79)
                                label_problem_C23= Label(second , text = "Cost between departments 2 and
8 is =  " + str(C[2,8]) , bg="gray92").place(relx=0.15,rely=0.82)
                                label_problem_C24= Label(second , text = "Cost between departments 3 and
8 is =  " + str(C[3,8]) , bg="gray92").place(relx=0.15,rely=0.85)
                                label_problem_C25= Label(second , text = "Cost between departments 4 and
8 is =  " + str(C[4,8]) , bg="gray92").place(relx=0.15,rely=0.88)
                                label_problem_C26= Label(second , text = "Cost between departments 5 and
8 is =  " + str(C[5,8]) , bg="gray92").place(relx=0.15,rely=0.91)
                                label_problem_C27= Label(second , text = "Cost between departments 6 and
8 is =  " + str(C[6,8]) , bg="gray92").place(relx=0.15,rely=0.94)
                                label_problem_C28= Label(second , text = "Cost between departments 7 and
8 is =  " + str(C[7,8]) , bg="gray92").place(relx=0.15,rely=0.97)

label_problem_D0= Label(second , text = "- DEPARTMENTS dimensions are: " ,
bg="gray92").place(relx=0.62,rely=0.23)
```

```
label_problem_D1= Label(second , text = "Department 1 has a width of " + str(W[1]) ,
bg="gray92").place(relx=0.65,rely=0.27)
label_problem_D2= Label(second , text = "Department 1 has a breadth of " + str(B[1]) ,
bg="gray92").place(relx=0.65,rely=0.30)

label_problem_D3= Label(second , text = "Department 2 has a width of " + str(W[2]) ,
bg="gray92").place(relx=0.65,rely=0.33)
label_problem_D4= Label(second , text = "Department 2 has a breadth of " + str(B[2]) ,
bg="gray92").place(relx=0.65,rely=0.36)

if (NumDepts>2):
    label_problem_D5= Label(second , text = "Department 3 has a width of " + str(W[3]) ,
bg="gray92").place(relx=0.65,rely=0.39)
    label_problem_D6= Label(second , text = "Department 3 has a breadth of " + str(B[3]) ,
bg="gray92").place(relx=0.65,rely=0.42)

    if (NumDepts>3):
        label_problem_D7= Label(second , text = "Department 4 has a width of " + str(W[4]) ,
bg="gray92").place(relx=0.65,rely=0.45)
        label_problem_D8= Label(second , text = "Department 4 has a breadth of " + str(B[4]) ,
bg="gray92").place(relx=0.65,rely=0.48)

        if (NumDepts>4):
            label_problem_D9= Label(second , text = "Department 5 has a width of " + str(W[5]) ,
bg="gray92").place(relx=0.65,rely=0.51)
            label_problem_D10= Label(second , text = "Department 5 has a breadth of " + str(B[5]) ,
bg="gray92").place(relx=0.65,rely=0.54)

            if (NumDepts>5):
                label_problem_D11= Label(second , text = "Department 6 has a width of " + str(W[6]) ,
bg="gray92").place(relx=0.65,rely=0.57)
                label_problem_D12= Label(second , text = "Department 6 has a breadth of " + str(B[6])
, bg="gray92").place(relx=0.65,rely=0.60)

                if (NumDepts>6):
                    label_problem_D13= Label(second , text = "Department 7 has a width of " +
str(W[7]) , bg="gray92").place(relx=0.65,rely=0.63)
                    label_problem_D14= Label(second , text = "Department 7 has a breadth of " +
str(B[7]) , bg="gray92").place(relx=0.65,rely=0.66)

                    if (NumDepts>7):
                        label_problem_D15= Label(second , text = "Department 8 has a width of " +
str(W[8]) , bg="gray92").place(relx=0.65,rely=0.69)
                        label_problem_D16= Label(second , text = "Department 8 has a breadth of " +
str(B[8]) , bg="gray92").place(relx=0.65,rely=0.72)

label_problem_P0= Label(second , text = "- PRODUCTION PLANT dimensions are: " ,
bg="gray92").place(relx=0.62,rely=0.12)
```

```
label_problem_P1= Label(second , text = "The plant has a WIDTH of " + str(xUpper) ,
bg="gray92").place(relx=0.65,rely=0.16)
label_problem_P2= Label(second , text = "The plant has a BREADTH of " + str(yUpper) ,
bg="gray92").place(relx=0.65,rely=0.19)


#DECLARING X AND Y OBJETIVE VARIABLES
for row in range(1,NumDepts+1):
        x[row]= model.addVar(vtype=GRB.CONTINUOUS, name='x_{}'.format(row))
        y[row]= model.addVar(vtype=GRB.CONTINUOUS, name='y_{}'.format(row))

#DECLARING DISTANCE VARIABLES
for row in range(1,NumDepts+1):
    for column in range (1,NumDepts+1):
            Dij[row,column] = model.addVar(vtype=GRB.CONTINUOUS, lb=-10000 , ub =1000,
name='d_{}{}'.format(row,column))
            Dx[row,column] = model.addVar(vtype=GRB.CONTINUOUS, lb=-10000 ,ub =
1000,name='d_x{}{}'.format(row,column))
            Dy[row,column] = model.addVar(vtype=GRB.CONTINUOUS, lb=-10000,ub =
1000,name='d_y{}{}'.format(row,column))
            Dxabs[row,column] = model.addVar(vtype=GRB.CONTINUOUS, lb=-10000,ub = 1000,
name='d_x{}{}'.format(row,column))
            Dyabs[row,column] = model.addVar(vtype=GRB.CONTINUOUS, lb=-10000 ,ub =
1000,name='d_y{}{}'.format(row,column))

#DECLARING S1 BINARY VARIABLES
for row in range (1,NumDepts+1):
    for column in range (1,NumDepts+1):
        if column>row:
            s1[row,column] = model.addVar(vtype=GRB.BINARY, name="s_1{}{}".format(row,column))
            s2[row,column] = model.addVar(vtype=GRB.BINARY, name="s_2{}{}".format(row,column))
            s3[row,column] = model.addVar(vtype=GRB.BINARY, name="s_3{}{}".format(row,column))
            s4[row,column] = model.addVar(vtype=GRB.BINARY, name="s_4{}{}".format(row,column))

model.update()

# FACILITY DIMENSIONS CONSTRAINTS
for row in range(1,NumDepts+1):
    model.addConstr( x[row]+(W[row]/2) <= xUpper)
    model.addConstr( x[row]-(W[row]/2) >= 0)
    model.addConstr( y[row]+(B[row]/2) <= yUpper)
    model.addConstr( y[row]-(B[row]/2) >= 0)

#CONSTRAINTS FOR DISTANCE AND OVERLAPPING
for row in range (1,NumDepts+1):
    for column in range (1,NumDepts+1):
        if(column>row):
            model.addConstr (Dx[row,column] == (x[row] - x[column]))
            model.addConstr (Dy[row,column] == (y[row] - y[column]))
            model.addConstr (Dxabs[row,column] == abs_(Dx[row,column]))
            model.addConstr (Dyabs[row,column] == abs_(Dy[row,column]))
            model.addConstr (Dij[row,column] == Dxabs[row,column] + Dyabs[row,column])

            model.addConstr((s1[row,column] == 1) >> (x[row] + (W[row]/2) <= x[column] -
(W[column]/2)))
            model.addConstr((s2[row,column] == 1) >> ( x[column]+(W[column]/2) <= x[row]-
(W[row]/2)))
            model.addConstr((s3[row,column] == 1) >> ( y[column]+(B[column]/2) <= y[row]-
(B[row]/2)))
            model.addConstr((s4[row,column] == 1) >> ( y[row]+(B[row]/2) <= y[column]-
(B[column]/2)))

            model.addConstr(s1[row,column] + s2[row,column] + s3[row,column] + s4[row,column] >=
1)

model.addConstr (x[1]==1)
model.addConstr (y[1]==4)

obj = quicksum ( Dij[row,column] * C[row,column]
    for row in range (1,NumDepts+1)
    for column in range (1,NumDepts+1)
    if column>row
)


model.setObjective(obj, GRB.MINIMIZE)

#model.Params.TimeLimit = 1800
#model.Params.MipFocus=1
#model.Params.BestObjStop=1220


model.optimize()

obj = model.objVal
obj = round (obj,4)

x_1 = model.getVarByName("x_1")
x_1 = round (x_1.X , 4)
y_1 = model.getVarByName("y_1")
y_1 = round (y_1.X , 4)

x_2 = model.getVarByName("x_2")
x_2 = round (x_2.X , 4)
y_2 = model.getVarByName("y_2")
```

```
y_2 = round (y_2.X , 4)

d_12 = model.getVarByName("d_12")
d_12 = round (d_12.X , 4)

if NumDepts>2:
    x_3 = model.getVarByName("x_3")
    x_3 = round (x_3.X , 4)
    y_3 = model.getVarByName("y_3")
    y_3 = round (y_3.X , 4)
    d_13 = model.getVarByName("d_13")
    d_13 = round (d_13.X , 4)
    d_23 = model.getVarByName("d_23")
    d_23 = round (d_23.X , 4)

    if NumDepts>3:
        x_4 = model.getVarByName("x_4")
        x_4 = round (x_4.X , 4)
        y_4 = model.getVarByName("y_4")
        y_4 = round (y_4.X , 4)
        d_14 = model.getVarByName("d_14")
        d_14 = round (d_14.X , 4)
        d_24 = model.getVarByName("d_24")
        d_24 = round (d_24.X , 4)
        d_34 = model.getVarByName("d_34")
        d_34 = round (d_34.X , 4)

        if NumDepts>4:
            x_5 = model.getVarByName("x_5")
            x_5 = round (x_5.X , 4)
            y_5 = model.getVarByName("y_5")
            y_5 = round (y_5.X , 4)
            d_15 = model.getVarByName("d_15")
            d_15 = round (d_15.X , 4)
            d_25 = model.getVarByName("d_25")
            d_25 = round (d_25.X , 4)
            d_35 = model.getVarByName("d_35")
            d_35 = round (d_35.X , 4)
            d_45 = model.getVarByName("d_45")
            d_45 = round (d_45.X , 4)

            if NumDepts>5:
                x_6 = model.getVarByName("x_6")
                x_6 = round (x_6.X , 4)
                y_6 = model.getVarByName("y_6")
                y_6 = round (y_6.X , 4)
                d_16 = model.getVarByName("d_16")
                d_16 = round (d_16.X , 4)
```

```
d_26 = model.getVarByName("d_26")
d_26 = round (d_26.X , 4)
d_36 = model.getVarByName("d_36")
d_36 = round (d_36.X , 4)
d_46 = model.getVarByName("d_46")
d_46 = round (d_46.X , 4)
d_56 = model.getVarByName("d_56")
d_56 = round (d_56.X , 4)

if NumDepts>6:
    x_7 = model.getVarByName("x_7")
    x_7 = round (x_7.X , 4)
    y_7 = model.getVarByName("y_7")
    y_7 = round (y_7.X , 4)
    d_17 = model.getVarByName("d_17")
    d_17 = round (d_17.X , 4)
    d_27 = model.getVarByName("d_27")
    d_27 = round (d_27.X , 4)
    d_37 = model.getVarByName("d_37")
    d_37 = round (d_37.X , 4)
    d_47 = model.getVarByName("d_47")
    d_47 = round (d_47.X , 4)
    d_57 = model.getVarByName("d_57")
    d_57 = round (d_57.X , 4)
    d_67 = model.getVarByName("d_67")
    d_67 = round (d_67.X , 4)

    if NumDepts>7:
        x_8 = model.getVarByName("x_8")
        x_8 = round (x_8.X , 4)
        y_8 = model.getVarByName("y_8")
        y_8 = round (y_8.X , 4)
        d_18 = model.getVarByName("d_18")
        d_18 = round (d_18.X , 4)
        d_28 = model.getVarByName("d_28")
        d_28 = round (d_28.X , 4)
        d_38 = model.getVarByName("d_38")
        d_38 = round (d_38.X , 4)
        d_48 = model.getVarByName("d_48")
        d_48 = round (d_48.X , 4)
        d_58 = model.getVarByName("d_58")
        d_58 = round (d_58.X , 4)
        d_68 = model.getVarByName("d_68")
        d_68 = round (d_68.X , 4)
        d_78 = model.getVarByName("d_78")
        d_78 = round (d_78.X , 4)

        if NumDepts>8:
```

```
x_9 = model.getVarByName("x_9")
x_9 = round (x_9.X , 4)
y_9 = model.getVarByName("y_9")
y_9 = round (y_9.X , 4)
d_19 = model.getVarByName("d_19")
d_19 = round (d_19.X , 4)
d_29 = model.getVarByName("d_29")
d_29 = round (d_29.X , 4)
d_39 = model.getVarByName("d_39")
d_39 = round (d_39.X , 4)
d_49 = model.getVarByName("d_49")
d_49 = round (d_49.X , 4)
d_59 = model.getVarByName("d_59")
d_59 = round (d_59.X , 4)
d_69 = model.getVarByName("d_69")
d_69 = round (d_69.X , 4)
d_79 = model.getVarByName("d_79")
d_79 = round (d_79.X , 4)
d_89 = model.getVarByName("d_89")
d_89 = round (d_89.X , 4)

if NumDepts>9:
    x_10 = model.getVarByName("x_10")
    x_10 = round (x_10.X , 4)
    y_10 = model.getVarByName("y_10")
    y_10 = round (y_10.X , 4)
    d_110 = model.getVarByName("d_110")
    d_110= round (d_110.X , 4)
    d_210 = model.getVarByName("d_210")
    d_210 = round (d_210.X , 4)
    d_310 = model.getVarByName("d_310")
    d_310 = round (d_310.X , 4)
    d_410 = model.getVarByName("d_410")
    d_410 = round (d_410.X , 4)
    d_510 = model.getVarByName("d_510")
    d_510 = round (d_510.X , 4)
    d_610 = model.getVarByName("d_610")
    d_610 = round (d_610.X , 4)
    d_710 = model.getVarByName("d_710")
    d_710 = round (d_710.X , 4)
    d_810 = model.getVarByName("d_810")
    d_810 = round (d_810.X , 4)
    d_910 = model.getVarByName("d_910")
    d_910 = round (d_910.X , 4)

    if NumDepts>10:
        x_11 = model.getVarByName("x_11")
        x_11 = round (x_11.X , 4)
```

```
y_11 = model.getVarByName("y_11")
y_11 = round (y_11.X , 4)
d_111 = model.getVarByName("d_111")
d_111= round (d_111.X , 4)
d_211 = model.getVarByName("d_211")
d_211 = round (d_211.X , 4)
d_311 = model.getVarByName("d_311")
d_311 = round (d_311.X , 4)
d_411 = model.getVarByName("d_411")
d_411 = round (d_411.X , 4)
d_511 = model.getVarByName("d_511")
d_511 = round (d_511.X , 4)
d_611 = model.getVarByName("d_611")
d_611 = round (d_611.X , 4)
d_711 = model.getVarByName("d_711")
d_711 = round (d_711.X , 4)
d_811 = model.getVarByName("d_811")
d_811 = round (d_811.X , 4)
d_911 = model.getVarByName("d_911")
d_911 = round (d_911.X , 4)
d_1011 = model.getVarByName("d_1011")
d_1011 = round (d_1011.X , 4)

if NumDepts>11:
    x_12 = model.getVarByName("x_12")
    x_12 = round (x_12.X , 4)
    y_12 = model.getVarByName("y_12")
    y_12 = round (y_12.X , 4)
    d_112 = model.getVarByName("d_112")
    d_112= round (d_112.X , 4)
    d_212 = model.getVarByName("d_212")
    d_212 = round (d_212.X , 4)
    d_312 = model.getVarByName("d_312")
    d_312 = round (d_312.X , 4)
    d_412 = model.getVarByName("d_412")
    d_412 = round (d_412.X , 4)
    d_512 = model.getVarByName("d_512")
    d_512 = round (d_512.X , 4)
    d_612 = model.getVarByName("d_612")
    d_612 = round (d_612.X , 4)
    d_712 = model.getVarByName("d_712")
    d_712 = round (d_712.X , 4)
    d_812 = model.getVarByName("d_812")
    d_812 = round (d_812.X , 4)
    d_912 = model.getVarByName("d_912")
    d_912 = round (d_912.X , 4)
    d_1012 = model.getVarByName("d_1012")
    d_1012 = round (d_1012.X , 4)
```

```
 d_1112 = model.getVarByName("d_1112")
 d_1112 = round (d_1112.X , 4)

 if NumDepts>12:
        x_13 = model.getVarByName("x_13")
        x_13 = round (x_13.X , 4)
        y_13 = model.getVarByName("y_13")
        y_13 = round (y_13.X , 4)
        d_113 = model.getVarByName("d_113")
        d_113= round (d_113.X , 4)
        d_213 = model.getVarByName("d_213")
        d_213 = round (d_213.X , 4)
        d_313 = model.getVarByName("d_313")
        d_313 = round (d_313.X , 4)
        d_413 = model.getVarByName("d_413")
        d_413 = round (d_413.X , 4)
        d_513 = model.getVarByName("d_513")
        d_513 = round (d_513.X , 4)
        d_613 = model.getVarByName("d_613")
        d_613 = round (d_613.X , 4)
        d_713 = model.getVarByName("d_713")
        d_713 = round (d_713.X , 4)
        d_813 = model.getVarByName("d_813")
        d_813 = round (d_813.X , 4)
        d_913 = model.getVarByName("d_913")
        d_913 = round (d_913.X , 4)
 d_1013 = model.getVarByName("d_1013")
 d_1013 = round (d_1013.X , 4)
 d_1113 = model.getVarByName("d_1113")
 d_1113 = round (d_1113.X , 4)
  d_1213 = model.getVarByName("d_1213")
  d_1213 = round (d_1213.X , 4)

if NumDepts>13:
           x_14 = model.getVarByName("x_14")
           x_14 = round (x_14.X , 4)
            y_14 = model.getVarByName("y_14")
           y_14 = round (y_14.X , 4)
         d_114 = model.getVarByName("d_114")
          d_114= round (d_114.X , 4)
        d_214 = model.getVarByName("d_214")
          d_214 = round (d_214.X , 4)
        d_314 = model.getVarByName("d_314")
           d_314 = round (d_314.X , 4)
        d_414 = model.getVarByName("d_414")
            d_414 = round (d_414.X , 4)
        d_514 = model.getVarByName("d_514")
             d_514 = round (d_514.X , 4)
 d_614 = model.getVarByName("d_614")
   d_614 = round (d_614.X , 4)
d_714 = model.getVarByName("d_714")
      d_714 = round (d_714.X , 4)
d_814 = model.getVarByName("d_814")
      d_814 = round (d_814.X , 4)
d_914 = model.getVarByName("d_914")
      d_914 = round (d_914.X , 4)
      d_1014 = model.getVarByName("d_1014")
      d_1014 = round (d_1014.X , 4)
      d_1114 = model.getVarByName("d_1114")
      d_1114 = round (d_1114.X , 4)
      d_1214 = model.getVarByName("d_1214")
      d_1214 = round (d_1214.X , 4)
      d_1314 = model.getVarByName("d_1314")
      d_1314 = round (d_1314.X , 4)

      if NumDepts>14:
            x_15 = model.getVarByName("x_15")
            x_15 = round (x_15.X , 4)
            y_15 = model.getVarByName("y_15")
            y_15 = round (y_15.X , 4)
            d_115 = model.getVarByName("d_115")
            d_115= round (d_115.X , 4)
            d_215 = model.getVarByName("d_215")
            d_215 = round (d_215.X , 4)
            d_315 = model.getVarByName("d_315")
            d_315 = round (d_315.X , 4)
            d_415 = model.getVarByName("d_415")
            d_415 = round (d_415.X , 4)
            d_515 = model.getVarByName("d_515")
            d_515 = round (d_515.X , 4)
            d_615 = model.getVarByName("d_615")
            d_615 = round (d_615.X , 4)
            d_715 = model.getVarByName("d_715")
            d_715 = round (d_715.X , 4)
            d_815 = model.getVarByName("d_815")
            d_815 = round (d_815.X , 4)
            d_915 = model.getVarByName("d_915")
            d_915 = round (d_915.X , 4)
            d_1015 = model.getVarByName("d_1015")
            d_1015 = round (d_1015.X , 4)
            d_1115 = model.getVarByName("d_1115")
            d_1115 = round (d_1115.X , 4)
            d_1215 = model.getVarByName("d_1215")
            d_1215 = round (d_1215.X , 4)
            d_1315 = model.getVarByName("d_1315")
            d_1315 = round (d_1315.X , 4)
```

```
                                        d_1415 = model.getVarByName("d_1415")
                                        d_1415 = round (d_1415.X , 4)

                                        if NumDepts>15:
                                            x_16 = model.getVarByName("x_16")
                                            x_16 = round (x_16.X , 4)
                                            y_16 = model.getVarByName("y_16")
                                            y_16 = round (y_16.X , 4)
                                            d_116 = model.getVarByName("d_116")
                                            d_116= round (d_116.X , 4)
                                            d_216 = model.getVarByName("d_216")
                                            d_216 = round (d_216.X , 4)
                                            d_316 = model.getVarByName("d_316")
                                            d_316 = round (d_316.X , 4)
                                            d_416 = model.getVarByName("d_416")
                                            d_416 = round (d_416.X , 4)
                                            d_516 = model.getVarByName("d_516")
                                            d_516 = round (d_516.X , 4)
                                            d_616 = model.getVarByName("d_616")
                                            d_616 = round (d_616.X , 4)
                                            d_716 = model.getVarByName("d_716")
                                            d_716 = round (d_716.X , 4)
                                            d_816 = model.getVarByName("d_816")
                                            d_816 = round (d_816.X , 4)
                                            d_916 = model.getVarByName("d_916")
                                            d_916 = round (d_916.X , 4)
                                            d_1016 =
model.getVarByName("d_1016")
                                            d_1016 = round (d_1016.X , 4)
                                            d_1116 =
model.getVarByName("d_1116")
                                            d_1116 = round (d_1116.X , 4)
                                            d_1216 =
model.getVarByName("d_1216")
                                            d_1216 = round (d_1216.X , 4)
                                            d_1316 =
model.getVarByName("d_1316")
                                            d_1316 = round (d_1316.X , 4)
                                            d_1416 =
model.getVarByName("d_1416")
                                            d_1416 = round (d_1416.X , 4)
                                            d_1516 =
model.getVarByName("d_1516")
                                            d_1516 = round (d_1516.X , 4)

                                            if NumDepts>16:
                                                x_17 = model.getVarByName("x_17")
                                                x_17 = round (x_17.X , 4)
```

```
y_17 = model.getVarByName("y_17")
y_17 = round (y_17.X , 4)
d_117 =
model.getVarByName("d_117")

d_117= round (d_117.X , 4)
d_217 =
model.getVarByName("d_217")

d_217 = round (d_217.X , 4)
d_317 =
model.getVarByName("d_317")

d_317 = round (d_317.X , 4)
d_417 =
model.getVarByName("d_417")

d_417 = round (d_417.X , 4)
d_517 =
model.getVarByName("d_517")

d_517 = round (d_517.X , 4)
d_617 =
model.getVarByName("d_617")

d_617 = round (d_617.X , 4)
d_717 =
model.getVarByName("d_717")

d_717 = round (d_717.X , 4)
d_817 =
model.getVarByName("d_817")

d_817 = round (d_817.X , 4)
d_917 =
model.getVarByName("d_917")

d_917 = round (d_917.X , 4)
d_1017 =
model.getVarByName("d_1017")

d_1017 = round (d_1017.X , 4)
d_1117 =
model.getVarByName("d_1117")

d_1117 = round (d_1117.X , 4)
d_1217 =
model.getVarByName("d_1217")

d_1217 = round (d_1217.X , 4)
d_1317 =
model.getVarByName("d_1317")

d_1317 = round (d_1317.X , 4)
d_1417 =
model.getVarByName("d_1417")

d_1417 = round (d_1417.X , 4)
d_1517 =
model.getVarByName("d_1517")

d_1517 = round (d_1517.X , 4)
```

```
d_1617 =
model.getVarByName("d_1617")
d_1617 = round (d_1617.X , 4)

if NumDepts>17:
    x_18 =
model.getVarByName("x_18")
    x_18 = round (x_18.X , 4)
    y_18 =
model.getVarByName("y_18")
    y_18 = round (y_18.X , 4)
    d_118 =
model.getVarByName("d_118")
    d_118= round (d_118.X , 4)
    d_218 =
model.getVarByName("d_218")
    d_218 = round (d_218.X , 4)
    d_318 =
model.getVarByName("d_318")
    d_318 = round (d_318.X , 4)
    d_418 =
model.getVarByName("d_418")
    d_418 = round (d_418.X , 4)
    d_518 =
model.getVarByName("d_518")
    d_518 = round (d_518.X , 4)
    d_618 =
model.getVarByName("d_618")
    d_618 = round (d_618.X , 4)
    d_718 =
model.getVarByName("d_718")
    d_718 = round (d_718.X , 4)
    d_818 =
model.getVarByName("d_818")
    d_818 = round (d_818.X , 4)
    d_918 =
model.getVarByName("d_918")
    d_918 = round (d_918.X , 4)
    d_1018 =
model.getVarByName("d_1018")
    d_1018 = round (d_1018.X , 4)
    d_1118 =
model.getVarByName("d_1118")
    d_1118 = round (d_1118.X , 4)
    d_1218 =
model.getVarByName("d_1218")
    d_1218 = round (d_1218.X , 4)
```

```
d_1318 =
model.getVarByName("d_1318")
d_1318 = round (d_1318.X , 4)
d_1418 =
model.getVarByName("d_1418")
d_1418 = round (d_1418.X , 4)
d_1518 =
model.getVarByName("d_1518")
d_1518 = round (d_1518.X , 4)
d_1618 =
model.getVarByName("d_1618")
d_1618 = round (d_1618.X , 4)
d_1718 =
model.getVarByName("d_1718")
d_1718 = round (d_1718.X , 4)

if NumDepts>18:
    x_19 =
model.getVarByName("x_19")
    x_19 = round (x_19.X , 4)
    y_19 =
model.getVarByName("y_19")
    y_19 = round (y_19.X , 4)
    d_119 =
model.getVarByName("d_119")
    d_119= round (d_119.X , 4)
    d_219 =
model.getVarByName("d_219")
    d_219 = round (d_219.X , 4)
    d_319 =
model.getVarByName("d_319")
    d_319 = round (d_319.X , 4)
    d_419 =
model.getVarByName("d_419")
    d_419 = round (d_419.X , 4)
    d_519 =
model.getVarByName("d_519")
    d_519 = round (d_519.X , 4)
    d_619 =
model.getVarByName("d_619")
    d_619 = round (d_619.X , 4)
    d_719 =
model.getVarByName("d_719")
    d_719 = round (d_719.X , 4)
    d_819 =
model.getVarByName("d_819")
    d_819 = round (d_819.X , 4)
```

```
d_919 =
model.getVarByName("d_919")
d_919 = round (d_919.X , 4)
d_1019 =
model.getVarByName("d_1019")
d_1019 = round (d_1019.X ,
4)
d_1119 =
model.getVarByName("d_1119")
d_1119 = round (d_1119.X ,
4)
d_1219 =
model.getVarByName("d_1219")
d_1219 = round (d_1219.X ,
4)
d_1319 =
model.getVarByName("d_1319")
d_1319 = round (d_1319.X ,
4)
d_1419 =
model.getVarByName("d_1419")
d_1419 = round (d_1419.X ,
4)
d_1519 =
model.getVarByName("d_1519")
d_1519 = round (d_1519.X ,
4)
d_1619 =
model.getVarByName("d_1619")
d_1619 = round (d_1619.X ,
4)
d_1719 =
model.getVarByName("d_1719")
d_1719 = round (d_1719.X ,
4)
d_1819 =
model.getVarByName("d_1819")
d_1819 = round (d_1819.X ,
4)

if NumDepts>19:
    x_20 =
model.getVarByName("x_20")
    x_20 = round (x_20.X ,
4)
    y_20 =
model.getVarByName("y_20")
```

```
y_20 = round (y_20.X ,
4)
d_120 =
model.getVarByName("d_120")
d_120= round (d_120.X
, 4)
d_220 =
model.getVarByName("d_220")
d_220 = round (d_220.X
, 4)
d_320 =
model.getVarByName("d_320")
d_320 = round (d_320.X
, 4)
d_420 =
model.getVarByName("d_420")
d_420 = round (d_420.X
, 4)
d_520 =
model.getVarByName("d_520")
d_520 = round (d_520.X
, 4)
d_620 =
model.getVarByName("d_620")
d_620 = round (d_620.X
, 4)
d_720 =
model.getVarByName("d_720")
d_720 = round (d_720.X
, 4)
d_820 =
model.getVarByName("d_820")
d_820 = round (d_820.X
, 4)
d_920 =
model.getVarByName("d_920")
d_920 = round (d_920.X
, 4)
d_1020 =
model.getVarByName("d_1020")
d_1020 = round
(d_1020.X , 4)
d_1120 =
model.getVarByName("d_1120")
d_1120 = round
(d_1120.X , 4)
d_1220 =
model.getVarByName("d_1220")
```

```
                                                                  d_1220 = round
(d_1220.X , 4)
                                                                  d_1320 =
model.getVarByName("d_1320")
                                                                  d_1320 = round
(d_1320.X , 4)
                                                                  d_1420 =
model.getVarByName("d_1420")
                                                                  d_1420 = round
(d_1420.X , 4)
                                                                  d_1520 =
model.getVarByName("d_1520")
                                                                  d_1520 = round
(d_1520.X , 4)
                                                                  d_1620 =
model.getVarByName("d_1620")
                                                                  d_1620 = round
(d_1620.X , 4)
                                                                  d_1720 =
model.getVarByName("d_1720")
                                                                  d_1720 = round
(d_1720.X , 4)
                                                                  d_1820 =
model.getVarByName("d_1820")
                                                                  d_1820 = round
(d_1820.X , 4)
                                                                  d_1920 =
model.getVarByName("d_1920")
                                                                  d_1920 = round
(d_1920.X , 4)


label_results_L0= Label(third, text = "-- Results for the LOCATION of each department are:",
bg="gray92").place(relx=0.08,rely=0.15)

label_results_D0= Label(third, text = "-- Results for the DISTANCE between departments are:",
bg="gray92").place(relx=0.58,rely=0.05)

label_results_T0= Label(third, text = "-- The MINIMIZED TOTAL COST of the LAYOUT is = " + str(obj) ,
bg="gray92").place(relx=0.08,rely=0.)

label_results_L1= Label(third, text = "The DEPARTMENT 1 location is X = " + str(x_1) + " and Y = " +
str(y_1), bg="gray92").place(relx=0.10,rely=0.19)
label_results_L2= Label(third, text = "The DEPARTMENT 2 location is X = " + str(x_2) + " and Y = " +
str(y_2), bg="gray92").place(relx=0.10,rely=0.22)
```

```
label_results_D1= Label(third, text = "The distance between DEPARTMENTS 1 and 2 is = " + str(d_12) ,
bg="gray92").place(relx=0.60,rely=0.09)

if NumDepts>2:
        label_results_L3= Label(third, text = "The DEPARTMENT 3 location is X = " + str(x_3) + " and Y = " +
str(y_3), bg="gray92").place(relx=0.10,rely=0.25)
        label_results_D2= Label(third, text = "The distance between DEPARTMENTS 1 and 3 is = " +
str(d_13) , bg="gray92").place(relx=0.60,rely=0.12)
        label_results_D3= Label(third, text = "The distance between DEPARTMENTS 2 and 3 is = " +
str(d_23) , bg="gray92").place(relx=0.60,rely=0.15)

        if NumDepts>3:
                label_results_L4= Label(third, text = "The DEPARTMENT 4 location is X = " + str(x_4) + " and Y =
" + str(y_4), bg="gray92").place(relx=0.10,rely=0.28)
                label_results_D4= Label(third, text = "The distance between DEPARTMENTS 1 and 4 is = " +
str(d_14) , bg="gray92").place(relx=0.60,rely=0.18)
                label_results_D5= Label(third, text = "The distance between DEPARTMENTS 2 and 4 is = " +
str(d_24) , bg="gray92").place(relx=0.60,rely=0.21)
                label_results_D6= Label(third, text = "The distance between DEPARTMENTS 3 and 4 is = " +
str(d_34) , bg="gray92").place(relx=0.60,rely=0.24)

                if NumDepts>4:
                        label_results_L5= Label(third, text = "The DEPARTMENT 5 location is X = " + str(x_5) + " and
Y = " + str(y_5), bg="gray92").place(relx=0.10,rely=0.31)
                        label_results_D7= Label(third, text = "The distance between DEPARTMENTS 1 and 5 is = " +
str(d_15) , bg="gray92").place(relx=0.60,rely=0.27)
                        label_results_D8= Label(third, text = "The distance between DEPARTMENTS 2 and 5 is = " +
str(d_25) , bg="gray92").place(relx=0.60,rely=0.30)
                        label_results_D9= Label(third, text = "The distance between DEPARTMENTS 3 and 5 is = " +
str(d_35) , bg="gray92").place(relx=0.60,rely=0.33)
                        label_results_D10= Label(third, text = "The distance between DEPARTMENTS 4 and 5 is = "
+ str(d_45) , bg="gray92").place(relx=0.60,rely=0.36)

                        if NumDepts>5:
                                label_results_L6= Label(third, text = "The DEPARTMENT 6 location is X = " + str(x_6) + "
and Y = " + str(y_6), bg="gray92").place(relx=0.10,rely=0.34)
                                abel_results_D11= Label(third, text = "The distance between DEPARTMENTS 1 and 6 is
= " + str(d_16) , bg="gray92").place(relx=0.60,rely=0.39)
                                label_results_D12= Label(third, text = "The distance between DEPARTMENTS 2 and 6 is
= " + str(d_26) , bg="gray92").place(relx=0.60,rely=0.42)
                                label_results_D13= Label(third, text = "The distance between DEPARTMENTS 3 and 6 is
= " + str(d_36) , bg="gray92").place(relx=0.60,rely=0.45)
                                label_results_D14= Label(third, text = "The distance between DEPARTMENTS 4 and 6 is
= " + str(d_46) , bg="gray92").place(relx=0.60,rely=0.48)
                                label_results_D15= Label(third, text = "The distance between DEPARTMENTS 5 and 6 is
= " + str(d_56) , bg="gray92").place(relx=0.60,rely=0.51)

                                if NumDepts>6:
```

```
                    label_results_L7= Label(third, text = "The DEPARTMENT 7 location is X = " +
str(x_7) + " and Y = " + str(y_7), bg="gray92").place(relx=0.10,rely=0.37)
                    label_results_D16= Label(third, text = "The distance between DEPARTMENTS 1
and 7 is = " + str(d_17) , bg="gray92").place(relx=0.60,rely=0.54)
                    label_results_D17= Label(third, text = "The distance between DEPARTMENTS 2
and 7 is = " + str(d_27) , bg="gray92").place(relx=0.60,rely=0.57)
                    label_results_D18= Label(third, text = "The distance between DEPARTMENTS 3
and 7 is = " + str(d_37) , bg="gray92").place(relx=0.60,rely=0.60)
                    label_results_D19= Label(third, text = "The distance between DEPARTMENTS 4
and 7 is = " + str(d_47) , bg="gray92").place(relx=0.60,rely=0.63)
                    label_results_D20= Label(third, text = "The distance between DEPARTMENTS 5
and 7 is = " + str(d_57) , bg="gray92").place(relx=0.60,rely=0.66)
                    label_results_D21= Label(third, text = "The distance between DEPARTMENTS 6
and 7 is = " + str(d_67) , bg="gray92").place(relx=0.60,rely=0.69)

                    if NumDepts>7:
                        label_results_L8= Label(third, text = "The DEPARTMENT 8 location is X = " +
str(x_8) + " and Y = " + str(y_8), bg="gray92").place(relx=0.10,rely=0.40)
                        label_results_D22= Label(third, text = "The distance between DEPARTMENTS
1 and 8 is = " + str(d_18) , bg="gray92").place(relx=0.60,rely=0.72)
                        label_results_D23= Label(third, text = "The distance between DEPARTMENTS
2 and 8 is = " + str(d_28) , bg="gray92").place(relx=0.60,rely=0.75)
                        label_results_D24= Label(third, text = "The distance between DEPARTMENTS
3 and 8 is = " + str(d_38) , bg="gray92").place(relx=0.60,rely=0.78)
                        label_results_D25= Label(third, text = "The distance between DEPARTMENTS
4 and 8 is = " + str(d_48) , bg="gray92").place(relx=0.60,rely=0.81)
                        label_results_D26= Label(third, text = "The distance between DEPARTMENTS
5 and 8 is = " + str(d_58) , bg="gray92").place(relx=0.60,rely=0.84)
                        label_results_D27= Label(third, text = "The distance between DEPARTMENTS
6 and 8 is = " + str(d_68) , bg="gray92").place(relx=0.60,rely=0.87)
                        label_results_D28= Label(third, text = "The distance between DEPARTMENTS
7 and 8 is = " + str(d_78) , bg="gray92").place(relx=0.60,rely=0.90)


                        if NumDepts>8:
                            label_results_L9= Label(third, text = "The DEPARTMENT 9 location is X = "
+ str(x_9) + " and Y = " + str(y_9), bg="gray92").place(relx=0.10,rely=0.43)

                            if NumDepts>9:
                                label_results_L10= Label(third, text = "The DEPARTMENT 10 location
is X = " + str(x_10) + " and Y = " + str(y_10), bg="gray92").place(relx=0.10,rely=0.46)

                                if NumDepts>10:
                                    label_results_L10= Label(third, text = "The DEPARTMENT 11
location is X = " + str(x_11) + " and Y = " + str(y_11), bg="gray92").place(relx=0.10,rely=0.49)

                                    if NumDepts>11:
                                        label_results_L12= Label(third, text = "The DEPARTMENT 12
location is X = " + str(x_12) + " and Y = " + str(y_12), bg="gray92").place(relx=0.10,rely=0.52)

                                        if NumDepts>12:
                                            label_results_L13= Label(third, text = "The
DEPARTMENT 13 location is X = " + str(x_13) + " and Y = " + str(y_13),
bg="gray92").place(relx=0.10,rely=0.55)

                                            if NumDepts>13:
                                                label_results_L14= Label(third, text = "The
DEPARTMENT 14 location is X = " + str(x_14) + " and Y = " + str(y_14),
bg="gray92").place(relx=0.10,rely=0.58)

                                                if NumDepts>14:
                                                    label_results_L15= Label(third, text = "The
DEPARTMENT 15 location is X = " + str(x_15) + " and Y = " + str(y_15),
bg="gray92").place(relx=0.10,rely=0.61)

                                                    if NumDepts>15:
                                                        label_results_L16= Label(third, text = "The
DEPARTMENT 16 location is X = " + str(x_16) + " and Y = " + str(y_16),
bg="gray92").place(relx=0.10,rely=0.64)

                                                        if NumDepts>16:
                                                            label_results_L17= Label(third, text =
"The DEPARTMENT 17 location is X = " + str(x_17) + " and Y = " + str(y_17),
bg="gray92").place(relx=0.10,rely=0.67)

                                                            if NumDepts>17:
                                                                label_results_L18= Label(third,
text = "The DEPARTMENT 18 location is X = " + str(x_18) + " and Y = " + str(y_18),
bg="gray92").place(relx=0.10,rely=0.70)

                                                                if NumDepts>18:
                                                                    label_results_L19=
Label(third, text = "The DEPARTMENT 19 location is X = " + str(x_19) + " and Y = " + str(y_19),
bg="gray92").place(relx=0.10,rely=0.73)

                                                                    if NumDepts>19:
                                                                        label_results_L20=
Label(third, text = "The DEPARTMENT 20 location is X = " + str(x_20) + " and Y = " + str(y_20),
bg="gray92").place(relx=0.10,rely=0.76)


label_results_graph= Label(fourth, text = "-- The OPTIMIZED LAYOUT is:",
bg="gray92").place(relx=0.2,rely=0.1)

heightcanvas= yUpper * 30
```

```
widthcanvas = xUpper * 30

canvas3 = Canvas(fourth, height= heightcanvas, width= widthcanvas, bg = "white")

canvas3.place(relx= 0.30 , rely = 0.15)

Dept1_rectangle = canvas3.create_rectangle( 30*(x_1 - (W[1]/2)), 30*(y_1 + (B[1]/2)), 30*(x_1 +
(W[1]/2)), 30*(y_1 - (B[1]/2)), fill="cyan")
Dept1_lable = canvas3.create_text( 30*x_1, 30*y_1 , text = "1" , fill = "black")
Dept2_rectangle = canvas3.create_rectangle( 30*(x_2 - (W[2]/2)), 30*(y_2 + (B[2]/2)), 30*(x_2 +
(W[2]/2)), 30*(y_2 - (B[2]/2)), fill="blue")
Dept2_lable = canvas3.create_text( 30*x_2, 30*y_2 , text = "2" , fill = "black")

if NumDepts>2:
    Dept3_rectangle = canvas3.create_rectangle( 30*(x_3 - (W[3]/2)), 30*(y_3 + (B[3]/2)), 30*(x_3 +
(W[3]/2)), 30*(y_3 - (B[3]/2)), fill="red")
    Dept3_lable = canvas3.create_text( 30*x_3, 30*y_3 , text = "3" , fill = "black")

    if NumDepts>3:
        Dept4_rectangle = canvas3.create_rectangle(30*(x_4 - (W[4]/2)), 30*(y_4 + (B[4]/2)), 30*(x_4 +
(W[4]/2)), 30*(y_4 - (B[4]/2)), fill="green")
        Dept4_lable = canvas3.create_text( 30*x_4, 30*y_4 , text = "4" , fill = "black")

        if NumDepts>4:
            Dept5_rectangle = canvas3.create_rectangle( 30*(x_5 - (W[5]/2)), 30*(y_5 + (B[5]/2)) ,
30*(x_5 + (W[5]/2)) , 30*(y_5 - (B[5]/2)), fill="orange")
            Dept5_lable = canvas3.create_text( 30*x_5, 30*y_5 , text = "5" , fill = "black")

            if NumDepts>5:
                Dept6_rectangle = canvas3.create_rectangle( 30*(x_6 - (W[6]/2)), 30*(y_6 + (B[6]/2)) ,
30*(x_6 + (W[6]/2)) , 30*(y_6 - (B[6]/2)), fill="gray")
                Dept6_lable = canvas3.create_text( 30*x_6, 30*y_6 , text = "6" , fill = "black")

                if NumDepts>6:
                    Dept7_rectangle = canvas3.create_rectangle( 30*(x_7 - (W[7]/2)), 30*(y_7 +
(B[7]/2)) , 30*(x_7 + (W[7]/2)) , 30*(y_7 - (B[7]/2)), fill="yellow")
                    Dept7_lable = canvas3.create_text( 30*x_7, 30*y_7 , text = "7" , fill = "black")

                    if NumDepts>7:
                        Dept8_rectangle = canvas3.create_rectangle( 30*(x_8 - (W[8]/2)), 30*(y_8 +
(B[8]/2)) , 30*(x_8 + (W[8]/2)) , 30*(y_8 - (B[8]/2)), fill="lime")
                        Dept8_lable = canvas3.create_text( 30*x_8, 30*y_8 , text = "8" , fill = "black")

                        if NumDepts>8:
                            Dept9_rectangle = canvas3.create_rectangle( 30*(x_9 - (W[9]/2)),
30*(y_9 + (B[9]/2)) , 30*(x_9 + (W[9]/2)) , 30*(y_9 - (B[9]/2)), fill="magenta")
                            Dept9_lable = canvas3.create_text( 30*x_9, 30*y_9 , text = "9" , fill =
"black")
```

```
                            if NumDepts>9:
                                Dept10_rectangle = canvas3.create_rectangle( 30*(x_10 -
(W[10]/2)), 30*(y_10 + (B[10]/2)) , 30*(x_10 + (W[10]/2)) , 30*(y_10 - (B[10]/2)), fill="lightgray")
                                Dept10_lable = canvas3.create_text( 30*x_10, 30*y_10 , text = "10" ,
fill = "black")

                                if NumDepts>10:
                                    Dept11_rectangle = canvas3.create_rectangle( 30*(x_11 -
(W[11]/2)), 30*(y_11 + (B[11]/2)) , 30*(x_11 + (W[11]/2)) , 30*(y_11 - (B[11]/2)), fill="deepskyblue")
                                 Dept11_lable = canvas3.create_text( 30*x_11, 30*y_11 , text = "11"
, fill = "black")

                                    if NumDepts>11:
                                        Dept12_rectangle = canvas3.create_rectangle( 30*(x_12 -
(W[12]/2)), 30*(y_12 + (B[12]/2)) , 30*(x_12 + (W[12]/2)) , 30*(y_12 - (B[12]/2)), fill="darkblue")
                                        Dept12_lable = canvas3.create_text( 30*x_12, 30*y_12 ,
text = "12" , fill = "black")

                                        if NumDepts>12:
                                            Dept13_rectangle = canvas3.create_rectangle(
30*(x_13 - (W[13]/2)), 30*(y_13 + (B[13]/2)) , 30*(x_13 + (W[13]/2)) , 30*(y_13 - (B[13]/2)),
fill="brown")
                                            Dept13_lable = canvas3.create_text( 30*x_13, 30*y_13
, text = "13" , fill = "black")

                                            if NumDepts>13:
                                                Dept14_rectangle = canvas3.create_rectangle(
30*(x_14 - (W[14]/2)), 30*(y_14 + (B[14]/2)) , 30*(x_14 + (W[14]/2)) , 30*(y_14 - (B[14]/2)),
fill="darkolivegreen")
                                                Dept14_lable = canvas3.create_text( 30*x_14,
30*y_14 , text = "14" , fill = "black")

                                                if NumDepts>14:
                                                    Dept15_rectangle = canvas3.create_rectangle(
30*(x_15 - (W[15]/2)), 30*(y_15 + (B[15]/2)) , 30*(x_15 + (W[15]/2)) , 30*(y_15 - (B[15]/2)), fill="aqua")
                                                    Dept15_lable = canvas3.create_text( 30*x_15,
30*y_15 , text = "15" , fill = "black")

                                                    if NumDepts>15:
                                                        Dept16_rectangle =
canvas3.create_rectangle( 30*(x_16 - (W[16]/2)), 30*(y_16 + (B[16]/2)) , 30*(x_16 + (W[16]/2)) ,
30*(y_16 - (B[16]/2)), fill="pink")
                                                        Dept16_lable = canvas3.create_text(
30*x_16, 30*y_16 , text = "16" , fill = "black")

                                                        if NumDepts>16:
```

```
                                                                            Dept17_rectangle =
canvas3.create_rectangle( 30*(x_17 - (W[17]/2)), 30*(y_17 + (B[17]/2)) , 30*(x_17 + (W[17]/2)) ,
30*(y_17 - (B[17]/2)), fill="moccasin")
                                                                            Dept17_lable = canvas3.create_text(
30*x_17, 30*y_17 , text = "17" , fill = "black")

                                                                            if NumDepts>17:
                                                                                Dept18_rectangle =
canvas3.create_rectangle( 30*(x_18 - (W[18]/2)), 30*(y_18 + (B[18]/2)) , 30*(x_18 + (W[18]/2)) ,
30*(y_18 - (B[18]/2)), fill="chartreuse")
                                                                                Dept18_lable =
canvas3.create_text( 30*x_18, 30*y_18 , text = "18" , fill = "black")

                                                                                if NumDepts>18:
                                                                                    Dept19_rectangle =
canvas3.create_rectangle( 30*(x_19 - (W[19]/2)), 30*(y_19 + (B[19]/2)) , 30*(x_19 + (W[19]/2)) ,
30*(y_19 - (B[19]/2)), fill="orchid")
                                                                                    Dept19_lable =
canvas3.create_text( 30*x_19, 30*y_19 , text = "19" , fill = "black")

                                                                                    if NumDepts>19:
                                                                                        Dept20_rectangle =
canvas3.create_rectangle( 30*(x_20 - (W[20]/2)), 30*(y_20 + (B[20]/2)) , 30*(x_20 + (W[20]/2)) ,
30*(y_20 - (B[20]/2)), fill="indianred")
                                                                                        Dept20_lable =
canvas3.create_text( 30*x_20, 30*y_20 , text = "20" , fill = "black")



roota.mainloop()
```