SAPIENZA
UNIVERSITÀ DI ROMA

Optimal transportation model applied to healthcare facilities in Covid-19 scenario

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Ingegneria Gestionale

Candidate

Giacomo Bacchetta

ID number 1840949

Thesis Advisor

Prof. Ruggiero Seccia

Academic Year 2020/2021

Optimal transportation model applied to healthcare facilities in Covid-19 scenario

Bachelor's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: bacchetta.1840949@studenti.uniroma1.it

Agli amici della 'foto di gruppo'.
Agli zii e ai cugini che ci sono sempre stati.
A mia madre, mio padre e mia sorella. Per tutto.

Abstract

When the Covid-19 pandemic broke out in the early months of 2020, the world population found itself having to face an emergency which it was not prepared for and which, from the beginning, saw governments and health institutions mobilize to find, regulate and implement measures for countering the spread of the aetiological agent in question, the SARS-CoV-2 virus.
For this purpose, great attention was paid to the possible optimization processes about all the parameters involved such detection and tracking of cases, methods of health care, adaptation or total conversion of hospital departments, management of health personnel, and others.
Keeping this in mind, we built an optimization model, illustrated below, about the sorting of patients to be admitted to the various treatment centers. We considered the general availability of ordinary or intensive care beds and the possible need for newly equipped wards, as well.
This optimization model works in close synergy with other models which focus on the prediction of cases of infection and on the personnel staff shifts in care facilities.

Contents

Chapter 1

Introduction

1.1 The theoretic problem

The model studied in our work is an **optimal transportation model**.
It derives from operational research applied to logistics and it allocates elements (the infected to be hospitalized) from several sources to the relative destinations that we call collection centers (hospitals).
This typology of optimizing problem is used to define which connections to activate between the multiple 'element-center' pairs and which not. Simultaneously the aim of the model is to minimize a function with a parameter which may be the cost to be incurred to activate the connection (transport cost) or the distance between the two extremes of the connection.
This is done through a specific optimizing formulation with constraints related to the capacity or the level of service of the several centers.
For this reason, we speak about a **combinatorial optimization problem** with a finite set of admissible solutions.
Initially, we have only one decision variable associated with the basic formulation of our model. Since this variable can only assume binary values (it is equal to 1 for each active link, 0 otherwise), the problem belongs to the class of problems **PL01** (**binary linear programming**). Only later we add another decision variable (to model the saturation rate of the collection centers) that can take on real values and consequently the problem becomes a **mixed linear programming problem**.

1.2 Context

Since the referent university of our work is "La Sapienza" University of Rome, we choose the capital of Italy as the city object of our study.

Although Rome was not particularly affected by the virus like other cities in northern Italy, it still represents a highly complex logistic scenario due to the elevated number of health facilities available to citizens (both private and public), to the size of the municipal area and also to the chaos that characterizes metropolitan cities.
To simplify the construction of the model, we consider exclusively ten hospitals among the most renowned in the city of Rome. Three are not authorized to treat Covid-19 patients and these are: Sant'Andrea, Sandro Pertini Hospital and Policlinico Casilino. The facilities already enabled are instead Umberto I Polyclinic, Gemelli Polyclinic, Spallanzani Institute, Tor Vergata Polyclinic, Bambin Gesù Hospital, Fatebenefratelli Hospital, San Camillo Hospital.

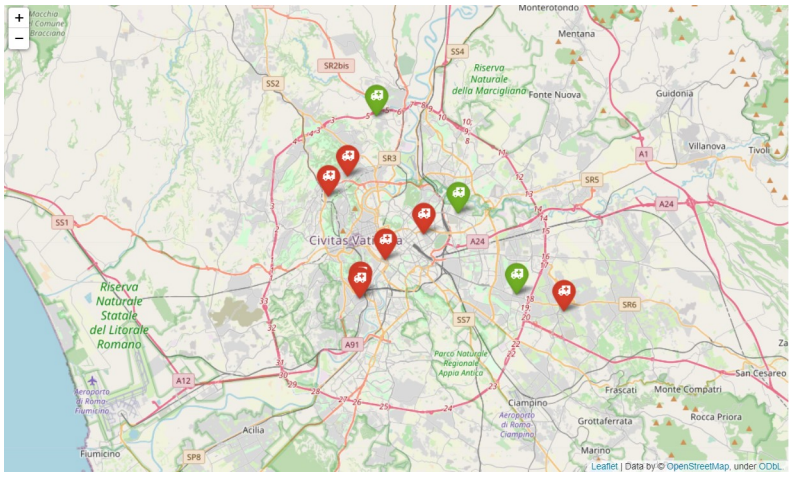

Figure 1.1. Map of the city of Rome on which the hospitals considered are highlighted through the use of the Folium library. We highlight in green the hospital not able to treat Covid patients, in red the others.

1.3 Thesis outline

The goal of the thesis is the formulation of an optimal transportation model as functional as possible to be applied to the city and to the context taken as a reference. This allows us to assess how hospitals manage the health emergency in a given time frame, knowing the expected number of infected to be hospitalized.
In this model, first of all, we define the logistical links between patients and hospitals to minimize the objective function and the only parameter of interest within the basic formulation (the distance).
In the primary formulation (§2.1), there are two basic constraints to be taken into account: the respect for the ordinary capacity of the hospitals and the guarantee of care for each patient.

Proceeding with our work, we need to obtain more exhaustive solutions than those of the basic formulation. The possibility of a more specific study and of possible alterations of the reference context bring us to the addition of constraints to the model and terms to the objective function.
The first change (§2.2) comes from the necessity to balance the percentage of assistance (number of patients / total hospital capacity) ensured by the various hospitals. In order to evenly distribute the workload among the health facilities, we introduce a new variable that is inserted in the objective function.
In the second change (§2.3) we consider the presence of patients who have a greater need for specialist care and adequately equipped beds. Consequently, we take into account the availability of intensive care in each hospital for the calculation of the total capacity.
Lastly, the third change (§2.4) to the context is linked to the hypothesis of an emergency scenario where patients to be hospitalized are more than the total number of beds available among all the hospitals considered. This scenario brings to the conversion of non-Covid hospitals into Covid ones. To carry out the conversion operation, we must bear a cost. To minimize this economic aspect, we add new parameters and variables to the objective function.
All these changes involve adding new constraints or modifying existing ones.
Through the application of our work and the subsequent corrections mentioned above, we can optimize the medical assistance making it more efficient and prompt. In fact, by avoiding the saturation of hospitals, guaranteeing assistance from the nearest structure to every infected person, facilitating the work of health personnel, and opening, if necessary, other health structures, we can achieve this result.

Chapter 2

Models

In this part we deal with the construction of the optimizing model covering both the context of the basic formulation and that of each change described in section 1.3. Each mathematical model, first in general and then applied to our case study, will be followed by its software implementation.

The entire code is available at:
`https://github.com/GiacomoBacchetta/Optimal-transportation-model-applied-to-healthcare-facilities-in-Covid-19-scenario`

2.1 Basic Formulation

2.1.1 General case

As mentioned earlier, the theoretical model closest to our needs is the optimal transportation model. The aim of this model is to establish which links to activate and which not, and therefore to define where to bring the elements of the problem while minimizing the objective function.
We proceed by defining:

- $\boldsymbol{E}$, the set of the elements to be distributed;
- $\boldsymbol{D}$, the set of the destinations;
- $\boldsymbol{r_{d,e}}$, the parameter used to weight the objective function (it can be the distance between each element and each destination or the cost to be incurred to activate the link), $d \in D, e \in E$;
- $\boldsymbol{a_d}$, the parameter that indicates the effective availability of each collection center, $d \in D$;

- $\boldsymbol{y_{d,e}}$, the binary decision variable that establishes which connections to activate, $d \in D, e \in E$.

The optimization process is bound by two constraints:

1. Each element must be brought to only one collection center:

$$\sum_{d \in D} y_{d,e} = 1 \qquad \forall e \in E \tag{2.1}$$

2. The capacity of each collection center must not be exceeded:

$$\sum_{e \in E} y_{d,e} \leq a_d \qquad \forall d \in D \tag{2.2}$$

So, we can define the basic mathematical formulation for the general case:

$$\begin{aligned}
&min && \sum_{e \in E} \sum_{d \in D} y_{d,e}\ r_{d,e} && \\
&s.t. && \sum_{d \in D} y_{d,e} = 1 && \forall\ e \in E \\
& && \sum_{e \in E} y_{d,e} \leq a_d && \forall\ d \in D \\
& && y_{d,e} \in \{0,1\} && \forall\ d \in D, \forall\ e \in E
\end{aligned}$$

2.1.2 Case study: implementation and results

Our work starts from the results of a preliminary study of prediction of the number of daily infected. This forecast is made over a **T** period consisting of eight days and for the entire territory of the Lazio region. In order to develop our optimization model it is necessary to multiply the data for the entire region by the hospitalization and incidence rate relating to Rome.
Taking as input the daily number of infected to be hospitalized ($\boldsymbol{n}$), we build a **boolean optimization model** where the main objective is to minimize the total distance between the infected and the hospitals. The optimization of the hospitalization procedure, through a model that tries to bring each patient to the nearest hospital, will have an obvious impact on the timeliness and effectiveness of medical care administered in the hospital.

Once known n, we can introduce a new set called **I** with cardinality equal to n. Every number included between 1 and n identifies one and only one infected.
Then we go on to identify the hospitals of our interest, already indicated in the previous chapter 'Context' (§1.2) , and to study their characteristics.
The names of these hospitals make up the set **H** and it is useful to define the following parameters for each hospital:

- $\boldsymbol{c_h}$, number of ordinary beds, $h \in H$;
- $\boldsymbol{k_h}$, number of beds equipped for critically ill patients (at the moment we do not use it), $h \in H$;

In the basic mathematical formulation we only consider hospitals that have a Covid ward and that are already able to assist and treat Covid-19 patients.
The model is built with an objective function to try to allocate each patient to the nearest hospital, if possible. This function has a binary variable:

$$y_{h,i} = \begin{cases} 1, \text{ if the i-th infected is transported to the h-th hospital, } i \in I,\ h \in H; \\ 0, \text{ otherwise.} \end{cases}$$

The objective function is:

$$min \sum_{i \in I} \sum_{h \in H} y_{h,i}\ d_{h,i}$$

where $\boldsymbol{d_{h,i}}$ is the distance between each infected and each hospital, expressed in kilometers. In particular, this distance is the geodesic distances calculated on the earth's surface. By geodesic, in fact, in mathematics or more precisely in differential geometry, we mean the shortest curve that joins the two points of a space [1]. In our case, 'colon in space' means every possible infected-hospital couple.
In defining the aforementioned function, the following constraints have been taken into account:

- Each patient is cared for by only one hospital:

$$\sum_{h \in H} y_{h,i} = 1 \qquad \forall i \in I \tag{2.3}$$

- The total capacity (that now coincides with the ordinary capacity) of each hospital must be respected:

$$\sum_{i \in I} y_{h,i} \leq c_h \qquad \forall h \in H \tag{2.4}$$

Software implementation

Since as the input data increase the problem becomes more complex to solve, we have opted for the execution of this model through a programming language. The chosen programming language is Python, which is implemented within Jupyter Notebook. The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access or can be installed on a remote server and accessed through the internet [2].
However, the use of Python alone allows us neither the formulation nor the resolution of the optimal transport problem we need. In fact, to do this we need to accompany the chosen programming language with some libraries. Python libraries are a set of useful functions that eliminate the necessity for writing codes from scratch. Python libraries play a essential role in developing machine learning, data science, data visualization, image and data manipulation applications and more.
In particular, the libraries called within our code are the following:

- **Pandas**, an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language [3];
- **Pyomo**, a Python-based open-source software package that supports a diverse set of optimization capabilities for formulating, solving, and analyzing optimization models [4];
- **Random**, an in-built module of Python which is used to generate random numbers [5];
- **Folium**, a library used for visualizing geospatial data [6];
- **Matplotlib**, a comprehensive library for creating static, animated, and interactive visualizations in Python [7].

The Pandas library is the first to come to our aid as we use it to analyze files such as the excel files.
In fact, the results from which we take initiative for the construction of our model are contained in an Excel file called InfExp (in .xlsx format). In this file there is a table in which each day of the period T (in the column 'date') is accompanied by the expected number of daily infected (in the column 'prevision').

An example of this Excel file is the following:

date	prevision
2021-09-20 00:00:00	498
2021-09-21 00:00:00	483
2021-09-22 00:00:00	481
2021-09-23 00:00:00	477
2021-09-24 00:00:00	473
2021-09-25 00:00:00	464
2021-09-26 00:00:00	460
2021-09-27 00:00:00	452

We must note, however, that the dates entered in the table are in DateTime type. Next to each date, there is also a fictitious time (this is always 00:00:00). Even though this schedule never changes and is always the same every day, it is difficult for us to work on it. To solve this problem we define the T list which, thanks to the *astype* function (to convert every date in string) and *list* function (to convert an object in a list), contains only the dates.
The Pandas library allows us to analyze the excel file and thus obtain the Prevision series, with which we work within our study. However, we remind us that the values contained in InfExp, and therefore in Prevision, refer to the entire Lazio region. To focus exclusively on the scenario of the city of Rome, we multiply the integer values of Prevision series by the incidence (assumed equal to the ratio between the inhabitants of Rome and the total ones of the Lazio region, 48%) and the hospitalization rate (equal to 3.8% [8]) of the city; so we obtain the Prev series. Each daily prevision represents the cardinality of the daily list I ("Infected").
InfExp is not the only excel file on which our work is based. Another essential Excel file is the one concerning the hospitals.

Hospital	Coordinates	Covid
POLICLINICO UMBERTO I	41.90782653426865, 12.510541266988186	Yes
POLICLINICO GEMELLI	41.93210246077846, 12.428657494836862	Yes
POLICLINICO TOR VERGATA	41.858587072830886, 12.630986236281153	Yes
SANT ANDREA	41.98320988212554, 12.470463782367839	No
OSPEDALE BAMBIN GESU	41.944520734877635, 12.445232901814228	Yes
OSPEDALE SANDRO PERTINI	41.92124073080262, 12.540100651653844	No
OSPEDALE FATEBENEFRATELLI	41.89174868226789, 12.477426389708965	Yes
ISTITUTO SPALLANZANI	41.86717180592309, 12.456050113176275	Yes
POLICLINICO CASILINO	41.86936210887187, 12.590163697269949	No
SAN CAMILLO	41.87042040958799, 12.45631721236428	Yes

This Excel file contains the name of each hospital, its coordinates, and information on whether the hospital is Covid or not. Using pandas we create:

- **H**, list containing the name of each hospital;
- **l**, dictionary that associate a list containing the coordinates to each hospital;
- **Covid**, dictionary which indicates to the code if the h-th hospital is able to treat Covid-19 patients ($Covid_h = Yes$) or not ($Covid_h = No$).

Then we define two other dictionaries relating to hospitals:

- ***c***, dictionary which contains the number of ordinary beds for each hospital;
- ***k***, dictionary which contains the number of intensive beds for each hospital.

The number of the ordinary and of the intensive capacity are chosen randomly within a list containing likely values.
The Random library is useful to us on another occasion, as well.
In fact, we are not aware of the real position of the infected. We assign them a fictitious one. In particular, each patient is associated with a latitude and a longitude value chosen at random (with *randint* tool) from those that define the boundaries of the metropolitan city of Rome. [1]
The position of each infected is then entered into a dictionary, called **p**.

[1] In order not to fall into excessive detail that could damage the modeling aspect we want to tackle, we think about Rome as a territory of quadrilateral shape that circumscribes the Grande Raccordo Anulare, the road that surrounds the capital connecting it with its suburbs.

Now that we are aware not only of the location of the hospitals but also of the infected, we can offer a first graphical interpretation of the information through the Folium library.

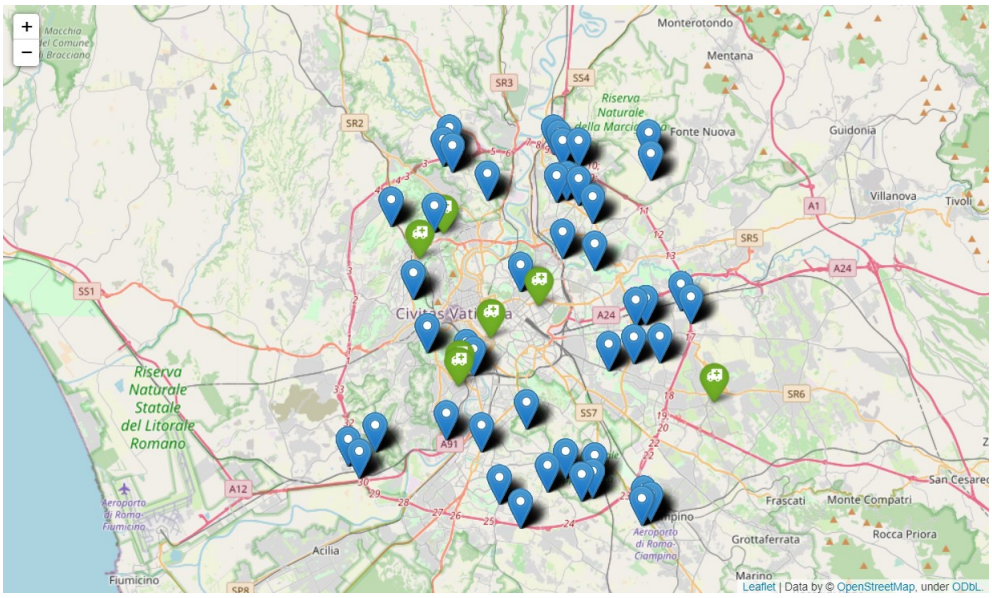

Figure 2.1. We highlight the infected with a blue marker and the Covid hospital with a green one.

Once we know the coordinates both of the infected to be hospitalized and of the hospitals, we can build the **d** matrix (containing every distance between each hospital and each infected) using the following code lines:

```
d = pd.DataFrame(index = H, columns = I)
for h in H:
            lat_osp = pi * l[h][0] / 180
            lon_osp = pi * l[h][1] / 180
            for i in I:
                lat_inf = pi * p[i][0] / 180
                lon_inf = pi * p[i][1] / 180
                fi = fabs(lon_osp -lon_inf)
                q = acos(sin(lat_inf) * sin(lat_osp) + cos(lat_inf) *
                                                    cos(lat_osp) *
                                                   cos(fi))
                distance = q * R
                d.at[h, i] = distance
```

Now, we just have to implement the model described in the chapter about the mathematical formulation. The model is implemented for every day of the period T, thanks to the Pyomo library. At the end of the daily implementation, we update the hospitals capacities according to the optimal solutions of the previous day.

The optimal solution is obtained thanks to the **GLPK** (GNU Linear Programming Kit) solver which implements the simplex algorithm and an interior point method for solving linear problems.
The results are graphically represented as follows:

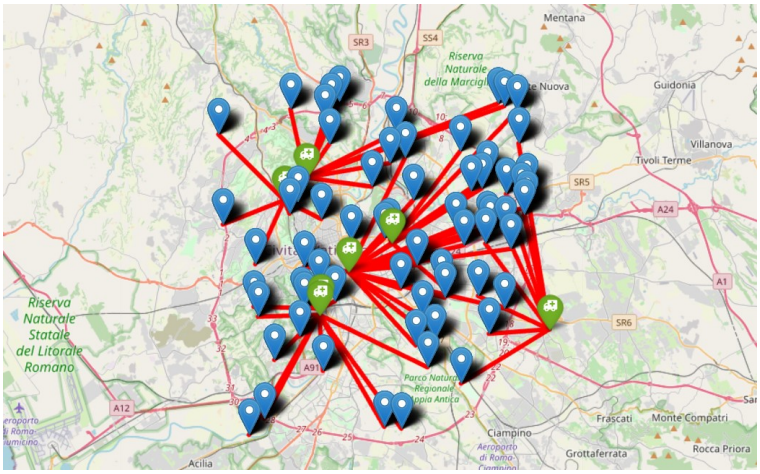

Figure 2.2. This is an example of how the results of the model are represented in our work.

For this graphical representation we define a new function, called **draw_map**, which uses some important tools of the Folium library. These tools are:

- *map*, to indicate the affected area to be circumscribed on the map;
- *marker*, to indicate the presence of an object of interest;
- *polyline*, used to draw lines on the map by connecting two points.

The *draw_map* function is the following:

```
def draw_map(sol, critical = False, conversion = False):

    m = f.Map(location=[41.9109, 12.4818], zoom_start=11)
    for h in H:
        if Covid[h] == 'Yes':
            title = h
            f.Marker(location = l[h],
                     icon = f.Icon(color = 'green', icon = 'ambulance
                                                        ', prefix = '
                                                        fa'),
                     popup = (pos_y, pos_x),
                     tooltip = title).add_to(m)

                                                        # marker for
                                                        Covid
                                                        hospitals
```

```
        elif conversion == True and Covid[h] == 'No' and h in
                                        HospitalConverted:
            f.Marker(location = l[h],
                     icon = f.Icon(color = 'purple', icon = '
                                                    ambulance',
                                                    prefix = 'fa'
                                                    ),
                     popup = 'Converted',
                     tooltip = h).add_to(m)

                                                    # marker for
                                                    converted
                                                    hospitals
        elif conversion == True and Covid[h] == 'No' and h not in
                                        HospitalConverted:
            f.Marker(location = l[h],
                     icon = f.Icon(color = 'beige', icon = 'ambulance
                                                    ', prefix = '
                                                    fa'),
                     popup = 'Not Converted',
                     tooltip = h).add_to(m)

                                                    # marker for
                                                    non-Covid
                                                    hospitals

        if critical == False and conversion == False:  # for basic
                                        formulation and first
                                        change
            for i in p.keys():
                pos = p[i]
                title = i
                f.Marker(location = pos, popup = pos, tooltip = title
                                                ).add_to(m)  #
                                                marker for each
                                                infected
                if sol[i] == h:
                    h_y = l[h][0]
                    h_x = l[h][1]
                    i_y = pos[0]
                    i_x = pos[1]
                    coordinates = [[h_y, h_x], [i_y, i_x]]
                    f.PolyLine(coordinates, color = 'red', weight = 5
                                                    ).add_to(m)  #
                                                    link between
                                                    infected and
                                                    hospital
```

```
        elif critical == True and conversion == False or critical ==
                                            True and conversion ==
                                            True: #for second and
                                            third change
            for i in p.keys():
                pos = p[i]
                title = i
                if i not in S_T:
                    f.Marker(location = pos, popup = pos, tooltip =
                                                    title).add_to(
                                                    m) # marker
                                                    for ordinary
                                                    infected
                else:
                    f.Marker(location=pos,
                     icon = f.Icon(color = 'red', icon = 'flask',
                                                     prefix = 'fa'
                                                     ),
                     popup = 'SERIOUS',
                     tooltip = title).add_to(m)  # marker for serious
                                                      infected

                if sol[i] == h:
                    h_y = l[h][0]
                    h_x = l[h][1]
                    i_y = pos[0]
                    i_x = pos[1]
                    coordinates = [[h_y, h_x], [i_y, i_x]]
                    f.PolyLine(coordinates, color = 'red', weight = 5
                                                    ).add_to(m) #
                                                    link between
                                                    infected and
                                                    hospital

    display(m)
    return
```

In our work, this function is used at the end of each change to represent the results. To do this, however, it is necessary to establish which are the input parameters of the function.
The first one of these is the dictionary of the optimal solution, called **sol**, where the keys are the infected and the values are the respective assigned hospitals. This input parameter is important to use a Polyline Folium's tool.

We then have two input arguments to the function:

- *critical*, it is a TRUE argument only when the model deals the allocation of severe patients. This leads the function to graph infected who need an intensive care beds with a red marker and the ones who only need an ordinary beds with a blue marker;
- *conversion*, it is a TRUE argument only when the model deals the conversion of non-Covid hospitals. This leads the function to indicate the converted hospitals through a purple marker and the not converted ones with a beige marker, both representing an ambulance.

The results offered by the solver at the end of the model implementation can also be summarized in an orthogram.
The orthogram (or tape diagram) is a particular type of diagram in which the relative frequency of interest is placed in the y-axis and the different variables are associated with the x-axis. The values are represented as rectangles to horizontal development. In our case on the x-axis there are the names of the ten hospitals considered, on the y-axis there is the percentage of assistance held by the hospital of reference. With percentage of care we define the relationship between the number of hospitalized and the total capacity of a hospital.
This graph is constructed by implementing the TableResults function as follows:

```
def TableResults(sol, critical = False, conversion = False):
    ValueCap = {}
    for h in H:
        count = 0                  # counter of the infected assisted
                                           by each hospital
        for i in sol.keys():
            if sol[i] == h:
                count = count + 1
        if count == 0:
            ValueCap[h] = count
        else:
            if critical == False and conversion == False:
                ValueCap[h] = count / c[h]
            elif critical == True and conversion == False:
                ValueCap[h] = count / (c[h] + k[h])
            elif critical == True and conversion == True:
                ValueCap[h] = count / (c[h] + k[h] + u mdl3.delta[h]
                                                   ())

    BasicGraph = pd.DataFrame(index = ValueCap.keys(), columns = ['
                                       Value'])
    BasicGraph.Value = ValueCap.values()
```

```
%matplotlib inline
BasicGraph.plot(kind = 'bar')
plt.xlabel('Hospital')
plt.ylabel('Percentual assistance')
plt.show()
```

The TableResults function has the same input parameters (*critical* and *conversion*) of the draw_map function (2.1.2) and therefore its output is differentiated between the changes' solutions to which it is applied.
In the case of the basic formulation, the graph is as follows:

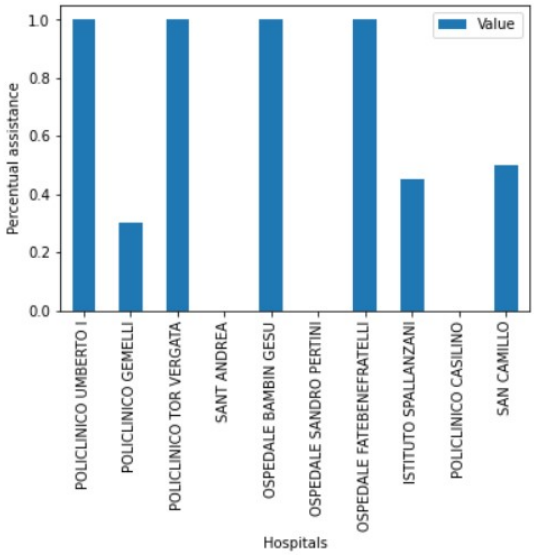

Figure 2.3. Percentage of assistance of each hospital according to the results of the basic formulation

As we can see from the graph, no infected person is transported properly to hospitals that do not have a Covid ward. On the other hand, the minimization of distances alone leads to the saturation of several hospitals.

2.2 First change: balancing of hospital filling

2.2.1 General case

As mentioned previously, in the course of the work we could have the necessity to make changes, some significant others less, to the basic formulation. These changes are justified either by the need for more detailed solutions or by alterations in the context which has to be redefined.
About the first change, we want to balance the filling of the various collection centers.

Achieving this allows us not to overload the centers and therefore to guarantee an adequate and constant level of service.
This is possible thanks to the introduction in the objective function of a new decision variable, $\boldsymbol{z}$. This variable takes continuous values in the range from 0 to 1 and indicates the maximum percentage of care for each collection center.
We are starting to work with two decision variables, one real and one boolean. This means that from now on the problem is a **mixed programming problem**.
To prevent any collection center from exceeding the limit z, we have to add a constraint as well. This is:

$$\frac{\sum_{e \in E} y_{d,e}}{a_d} \leq z \qquad \forall d \in D \tag{2.5}$$

Therefore, for each placement center, the ratio between the number of perceived elements and its capacity must be at most z.

The theoretical model becomes:

$$\begin{aligned}
min \quad & \sum_{e \in E} \sum_{d \in D} y_{d,e}\ r_{d,e} + w\ z & \\
s.t. \quad & \sum_{d \in D} y_{d,e} = 1 & \forall\ e \in E \\
& \sum_{e \in E} y_{d,e} \leq a_d & \forall\ d \in D \\
& \frac{\sum_{e \in E} y_{d,e}}{a_d} \leq z & \forall\ d \in D \\
& y_{d,e} \in \{0,1\} & \forall\ d \in D, \forall\ e \in E \\
& z \in (0,1] &
\end{aligned}$$

where the parameter $\boldsymbol{w}$ can be estimated at will according to how much we want to weight that part of the objective function.

2.2.2 Case study: implementation and results

Moving on to our case study, to not burden the operators of each hospital too much and to ensure adequate service, we must try to uniform the load among the hospitals considered. To achieve this, we consider the ratio between the number of people assisted by a hospital and its capacity (c). Within the relative constraint, this value must not exceed the one associated by the solver to the new decision variable z.

As in the general case, z is a decision variable between 0 and 1 that here indicates the maximum filling percentage of each hospital.
Therefore, in order to avoid an excessive workload of hospitals, the following constraint should be added to the model:

$$\frac{\sum_{i \in I} y_{h,i}}{c_h} \leq z \qquad \forall h \in H, z \in (0,1]$$

What we have from now on is therefore a more complex optimal transport model, characterized by two decision variables of different nature and referred to as a mixed programming problem.
Thus linking the general case to our case study, we have the following objective function:

$$min \sum_{i \in I} \sum_{h \in H} y_{h,i} \; d_{h,i} + w \; z$$

Software implementation

Moving to the implementation part of the change, this begins with the definition of the weight w present in the second addend of the target function.
Within our code we have set the value of w to 100. Larger values could lead the solver to focus too much on minimizing the second term of the target function, thus sacrificing the total distance traveled by the patients. On the contrary values too small of w, or tending to zero, could be the cause of solutions very similar, if not equal, to that of the basic formulation.
We are now ready to build the model thanks to Pyomo. After running this model as well, the results are represented using the same draw_map function (2.1.2) built earlier.
By performing the TableResults (2.1.2) function with the solution of the first change we see how the modifications made lead the hospitals with a high filling in the first change to leave several patients to other hospitals.

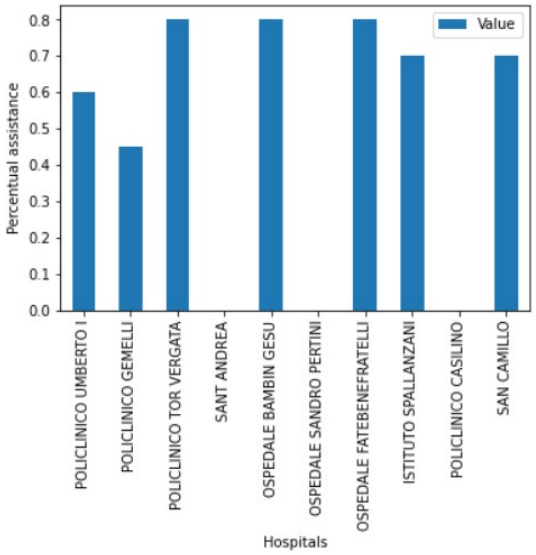

Figure 2.4. Percentage of assistance for each hospital according to the results of the first change.

We are now able to make a first comparison between the solution of the basic formulation model and that of the first modification model. From the graph we can see how the percentage of assistance varies greatly for each hospital. Moving from the basic formulation to that which has the objective of balancing the workload between all hospitals and adding the related constraint 2.6, the percentage values tend to be to the same level.

2.3 Second change: severe patients and intensive care

2.3.1 General Case

While the first change arises from the necessity to balance the filling of each collection center, the second change is due to the need to distinguish different types of elements that have different needs and not every collection center can guarantee them. We consider only two different types of elements.
This leads us to the definition of a new model as different elements need different approaches, not always present in all collection centers, and therefore not all structures can receive elements of both types.
We, therefore, define **G** (a subset for the set E), containing the different items. From now on we consider the parameter $\boldsymbol{v}$ that defines the maximum number of elements contained in G that can be received by each collection center.

In order to work with the new parameters and sets, we have to:

1. Add a new constraint which obliges us to transport the G elements exclusively in these centers that have a dedicated capacity (v_d). The constraint is:

$$\sum_{e \in G} y_{d,e} \leq v_d \qquad \forall d \in D \tag{2.6}$$

2. Modify the existing constraint 2.2 that deals with the respect of the total capacity ($a_d + v_d$) of each destination:

$$\sum_{e \in E} y_{d,e} \leq a_d + v_d \qquad \forall d \in D \tag{2.7}$$

3. Modify the existing constraint 2.5 about the calculation of the percentage of capacity occupied. Now this constraint is:

$$\frac{\sum_{e \in E} y_{d,e}}{a_d + v_d} \leq z \qquad \forall\ d \in D \tag{2.8}$$

Consequently, the theoretical problem is the following:

$$\begin{aligned}
min \quad & \sum_{e \in E} \sum_{d \in D} y_{d,e}\ r_{d,e} + w\ z & \\
s.t. \quad & \sum_{d \in D} y_{d,e} = 1 & \forall\ e \in E \\
& \sum_{e \in E} y_{d,e} \leq a_d + v_d & \forall\ d \in D \\
& \frac{\sum_{e \in E} y_{d,e}}{a_d + v_d} \leq z & \forall\ d \in D \\
& \sum_{e \in G} y_{d,e} \leq v_d & \forall d \in D \\
& y_{d,e} \in \{0,1\} & \forall\ d \in D, \forall\ e \in E \\
& z \in (0,1] &
\end{aligned}$$

2.3.2 Case study: implementation and results

Passing from the general case to the specific one, we identify the two types of elements as severe and non-severe patients.
Starting from public data and epidemiological studies, the estimate of the percentage

of infected with severe clinical symptoms currently stands at around 6% [9] . The severe patients need an ICU (Intensive Care Unit) place characterized by intensive treatments and continuous monitoring of the patient's vital signs. In the Covid area, the rooms dedicated to ICU must be at negative pressure and with an unidirectional air washing flow, so that the internal air does not contaminate the external one and there is no environmental passage of the virus [10].
It is, therefore, necessary to allocate these more serious patients, whose together daily form **S** subset of I, in appropriately equipped hospitals, always minimizing the total distance and the parameter z.
It is necessary to insert a new constraint such that severe patients are automatically referred to equipped hospitals, not exceeding their receptive capacities. This is:

$$\sum_{i \in S} y_{h,i} \leq k_h \qquad \forall h \in H$$

where k is the parameter that indicates intensive capacity of each hospital.

Since from this change we consider not only the ordinary capacity of each hospital but also the intensive one appointed to severe patients we have to modify two constraint of the model of the previous change:

1. Constraint 2.4 becomes:

$$\sum_{i \in I} y_{h,i} \leq c_h + k_h \qquad \forall\ h \in H \tag{2.9}$$

2. Constraint 2.6 becomes:

$$\frac{\sum_{i \in I} y_{h,i}}{c_h + k_h} \leq z \qquad \forall\ h \in H \tag{2.10}$$

Software implementation

We can just copy the previous model and insert the constraint defined in the mathematical phase. Once the new model is implemented with its new constraints, we have a different solution and, therefore, a different representation of the results respect to that of the previous model. This is a consequence of having considered two different types of infected: severe and non-severe patients to be hospitalized.
S is the list that contains daily severe patients. The list S is initialized inside the cell that deals with the iteration of the model for each day of the period T considered. This will be filled every day with 6% (2.3.2) of the daily infected, as for the above data.
At the end of the implementation of the model on the complete period T we have as output not only the optimal solution but also a new list, called $\boldsymbol{S_T}$. This one

contains all the serious infected from the period we are considering. We use this list to mark on the map severe patients differently from those who only need ordinary care. This is done always thanks to the draw_map function (2.1.2) which from now on has activated the input argument *critical*.

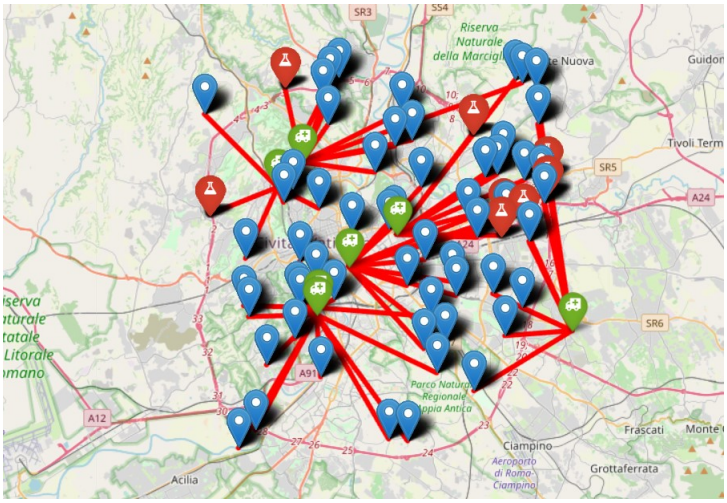

Figure 2.5. An example of how the results obtained from the second modification can be represented. It is important to note that serious patients are distinguished from "ordinary" ones and are indicated with a red marker representing an ampoule.

The different results between the first and second modifications are mainly visible thanks to the bar diagram. In this case we see that the percentages suffer a decrease for each hospital. This is due to the fact that, even if the number of infected is of previous change, we increase the total capacity of treatment centers. From this change, the total capacity is the sum of the ordinary and the intensive ones.

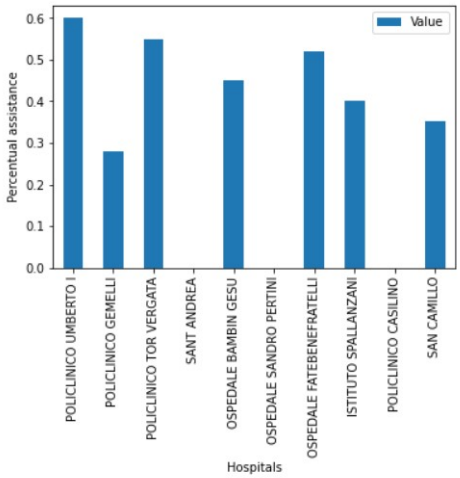

Figure 2.6. Percentage of assistance of each hospital according to the results of the second change.

2.4 Third change: emergency scenario

2.4.1 General case

While the previous changes are due to new goals or new computing needs, the third one arises from the need to be ready to face a possible emergency scenario.
It can happen that there is a sudden increase in demand and, therefore, in the number of elements to distribute among the collection centers.
To support this increase, it could be necessary to convert collection centers or departments that up to the point did not deal with the elements treated.
In this regard, we define:

- $\boldsymbol{D_N}$, the subset of D which contain every destination (or collection center) not able to receive the considered elements;
- $\boldsymbol{b}$, the cost incurred for the conversion of a collection center contained in D_N;
- $\boldsymbol{u}$, which indicates how many elements can be allocated to each enabled center contained in D_N;
- $\boldsymbol{\delta_d}$, a new binary variable equal to 1 if a collection centre contained in D_N begins to be able to receive treated E elements.

Considering the optimizer aspect, to achieve the objectives set in this change it is necessary to modify the objective function used so far, which becomes:

$$min \sum_{e \in E} \sum_{d \in D} y_{d,e}\ r_{c,e} + w\ z + b \sum_{d \in D_N} \delta_d \tag{2.11}$$

Therefore now the minimization of the objective function also concerns the total of the costs that are incurred in order to have more systems that can manage the elements of our interest.
To complete the model we must consider that, in the existing constraint 2.7 related to the respect of the capacity, also the centers contained in D_N have a capacity if they are converted. These in fact will not have more capacity equal to zero. The constraint becomes:

$$\sum_{e \in E} y_{d,e} \leq a_d + v_d + u\ \delta_d \qquad \forall d \in D \tag{2.12}$$

So the new theoretical model is the following:

$$
\begin{aligned}
min \quad & \sum_{e \in E} \sum_{d \in D} y_{d,e}\ r_{d,e} + w\ z + b \sum_{d \in D_N} \delta_d & \\
s.t. \quad & \sum_{d \in D} y_{d,e} = 1 & \forall\ e \in E \\
& \sum_{e \in E} y_{d,e} \leq a_d + v_d + u\ \delta_d & \forall\ d \in D \\
& \frac{\sum_{e \in E} y_{d,e}}{a_d + v_d} \leq z & \forall\ d \in D \\
& \sum_{e \in G} y_{d,e} \leq v_c & \forall d \in D \\
& y_{d,e} \in \{0,1\} & \forall\ d \in D, \forall\ e \in E \\
& z \in (0,1] & \\
& \delta_d \in \{0,1\} & \forall d \in D_N
\end{aligned}
$$

2.4.2 Case study: implementation and results

Shifting the focus to our case study, with the arrival of the flu season and the probable consequent increase in Covid infections, we must be ready to face a much larger demand than expected.
COVID hospitals could be unable to offer the health service that citizens need, with a consequent impossibility of finding the optimum solution if we continue to use the previous models. It is therefore essential to build a new model to ensure that initially not equipped hospitals (grouped in H_N) can host Covid-19 patients.
To solve this problem, we build a new model that assists us in choosing which non-Covid hospitals to convert, always having in mind the goal of minimizing both the total distances and the previously introduced z (2.2) parameter, but also the total costs incurred by institutions for conversions.
This is done by introducing new sets,variables, and parameters like:

- $\boldsymbol{H_N}$, the equivalent of D_N in the theoretical formulation and it defines the set of hospitals not authorized to treat Covid-19 patient;
- **b**, the cost incurred for the conversion of a hospital;
- **u**, the number of ordinary beds that can be added in the hospitals contained in H_N that are converted (by hypothesis it is equal to 20);

- $\boldsymbol{\delta_h}$, the new binary variable equal to 1 if the h-th non-COVID hospital is converted, 0 otherwise.

Taking up the general case, the new objective function is:

$$\min \sum_{i \in I} \sum_{h \in H} y_{h,i}\ d_{h,i} + w\ z + b \sum_{h \in H_N} \delta_h \tag{2.13}$$

For this change, we have to modify the constraint 2.9 that deals with the ordinary capacity of each hospital. In fact, while in the previous model this capacity was absent for the hospitals contained in H_N, now there is the possibility to open a Covid ward in these ones. So the new constraint is:

$$\sum_{i \in I} y_{h,i} \leq c_h + k_h + u\ \delta_h \qquad \forall h \in H \tag{2.14}$$

where the increase of capacity (u) is available only for the non-Covid hospitals that are converted by the solver ($\delta_h = 1$).
Consequently, after studying the different scenarios and applying the relative modifications to the model, we arrive at the following final model:

$$\begin{aligned}
min \quad & \sum_{i \in I} \sum_{h \in H} y_{h,i}\ d_{h,i} + w\ z + b \sum_{h \in H_N} \delta_h & \\
s.t. \quad & \sum_{h \in H} y_{h,i} = 1 & \forall\ i \in I \\
& \sum_{i \in I} y_{h,i} \leq c_h + k_h + u\ \delta_h & \forall\ h \in H \\
& \frac{\sum_{i \in I} y_{h,i}}{c_h + k_h} \leq z & \forall\ h \in H \\
& \sum_{i \in S} y_{h,i} \leq k_h & \forall\ h \in H \\
& y_{h,i} \in \{0,1\} & \forall\ h \in H, \forall\ i \in I \\
& z \in (0,1] & \\
& \delta_h \in \{0,1\} & \forall\ h \in H_N
\end{aligned}$$

Software implementation

The implementation of the change described in the previous chapter begins with the initialization of the H_N list which contains the names of the non-Covid hospitals according to the information of the Hospitals Excel file. We also define the constant b and set it a very high value equal to 20000.
After having initialized, through Pyomo, the new parameter u and the new variable

δ, we are now able to run the model, modified with new constraints, and solve it with GLPK solver. As we have repeated several times the pattern is executed on each day of the period we are considering.
The peculiarity of this change is that we hypothesize the sudden increase of infected by hospitalization on the last day.
Once implemented and solved, the model will be graphically represented using the draw_map function (pg.12) with the care of activating both input arguments.
If *critical* in fact allows us from the second modification to distinguish severe patients, *conversion* allows us to mark the converted hospitals differently.
At the end of the work this will be the result that is returned to us:

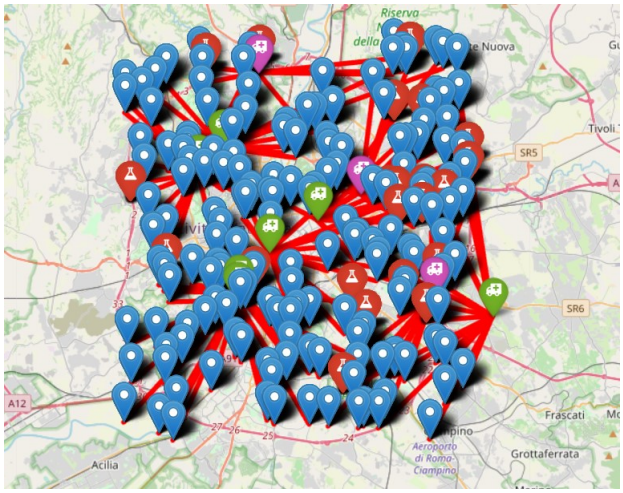

Figure 2.7. An example of representation of the results related to the third change. The converted hospitals are represented with a purple marker and an ambulance depicted on it.

Let's summarize the results of this change in the orthogram.

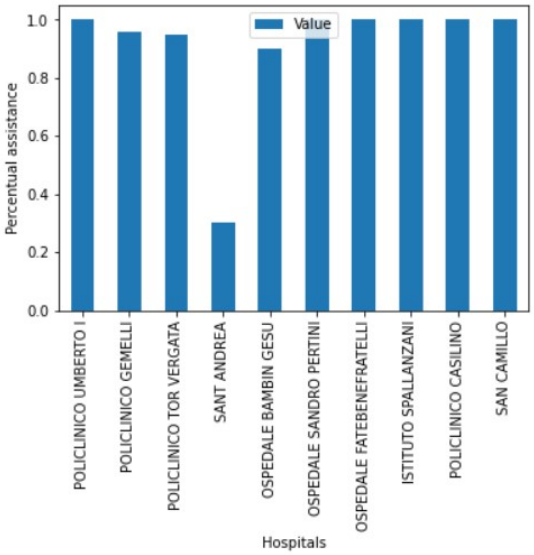

Figure 2.8. Percentage of assistance of each hospital according to the results of the third change.

From this graph we can deduce how, in the period considered since the beginning of work and randomizing the number of the suddenly infected, the solver suggests us to convert all three hospitals at our disposal. Conversion, however, is carried out only after reaching full coverage of hospitals that have always had a department dedicated to Covid-19.

Chapter 3

Conclusions

In this work we took a first look at applied operational research.
In particular we approached the world of logistics, studying the optimal transportation model.
Despite the difficulties we have found, such as the inability to find information about the exact and real hospital capacity in Covid departments, we were able to get a total view on the behavior of the hospitals themselves.
However, we have been able to see how, once the constraint on the percentage of care held by hospitals is inserted, they work together to avoid the overload.
In the course of our study we also found the need to enter more specifically into the context considering the percentage of infected people who need special care and, in the most serious cases, an ICU place. We have seen how this consideration brings us totally new and different solutions of the model, especially because not all hospitals do not have the same availability of intensive care.
Lastly, we have hypothesized to be in an emergency state in which we had to convert totally or partially departments that until then were not predisposed for the assistance of patients with infectious diseases. Obviously, in this case the solution of the model led us in our tests to convert non-Covid hospitals only after reaching the saturation of Covid ones.
Thanks to the application of the model studied in our work we are therefore able to make medical care timely and more efficient by adapting it to the different degrees of severity of patients.

Bibliography

[1] Geodetic: https://it.wikipedia.org/wiki/Geodetica

[2] Jupyter Notebook: https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#what-is-the-jupyter-notebook

[3] Pandas: https://pandas.pydata.org/docs/#:~:text=pandas%20is%20an%20open%20source,for%20the%20Python%20programming%20language.

[4] Pyomo: http://www.pyomo.org/

[5] Random: https://www.geeksforgeeks.org/python-random-module/#:~:text=Python%20Random%20module%20is%20an,a%20list%20or%20string%2C%20etc.

[6] Folium: https://www.analyticsvidhya.com/blog/2020/06/guide-geospatial-analysis-folium-python/

[7] Matplotlib: https://matplotlib.org/

[8] Italian hospitalization rate: https://www.agenas.gov.it/covid19/web/index.php?r=site%2Fgraph1

[9] Percentage of intensive care: https://www.agenas.gov.it/covid19/web/index.php?r=site%2Fgraph1

[10] Intensive Care Unit (ICU): https://www.gvmnet.it/specialita/terapia-intensiva

[11] An optimal solution for transportation problem using computing modelling: https://www.researchgate.net/publication/306507838_An_Optimal_Solution_for_Transportation_Problem_Using_Computing_Modelling

[12] Finding optimal locations of new stores: Chicago coffee shops, by IBM optimization decision

[13] Optimization in transportation problem: https://towardsdatascience.com/optimization-in-transportation-problem-f8137044b371