



School of Social Sciences

Supply Chain Management

Master Thesis

Mathematical modeling of location problems and their applications in  
the supply chain

Foteini Bourou

Supervisor: Angelo Sifaleras

Patras, Greece, June 2022



Theses / Dissertations remain the intellectual property of students (“authors/creators”), but in the context of open access policy they grant to the HOU a non-exclusive license to use the right of reproduction, customisation, public lending, presentation to an audience and digital dissemination thereof internationally, in electronic form and by any means for teaching and research purposes, for no fee and throughout the duration of intellectual property rights. Free access to the full text for studying and reading does not in any way mean that the author/creator shall allocate his/her intellectual property rights, nor shall he/she allow the reproduction, republication, copy, storage, sale, commercial use, transmission, distribution, publication, execution, downloading, uploading, translating, modifying in any way, of any part or summary of the dissertation, without the explicit prior written consent of the author/creator. Creators retain all their moral and property rights.





# Mathematical modeling of location problems and their applications in the supply chain

Foteini Bourou

## Supervising Committee

Supervisor:

Ioannis Giannikos

Angelo Sifaleras

Hellenic Open University

Hellenic Open University

University of Patras

University of Macedonia

Alexandros Diamantidis

Hellenic Open University

Aristotle University of Thessaloniki

Patras, Greece, June 2022



*First, I would like to express my gratitude to my master thesis supervisor, Mr. Angelo Sifaleras, professor at Department of Applied Informatics, School of Information Sciences, of the University of Macedonia, and at the School of Social Science, of the Hellenic Open University. He has been very supportive when I needed his guidance and understanding with the repercussions of lack of knowledge and experience on my behalf.*

*Afterwards, I would like to thank my family for standing by my side during my studies and being supportive to me. But, most of all, I would like to thank my husband, Argyris, for being there for me these two academic years, encouraging me to keep on studying, making me laugh when I needed it and taught me not to give up despite the obstacles I came through.*

*Finally, my Thesis is devoted to my beloved son, Christos, who gave me strength to carry on.*





## **Abstract**

The aim of this Thesis is to study facility location problems and implement their mathematical modeling through computer code. Facility location problems comprise a crucial factor for the strategy of managing the supply chain. The supply chain includes the flows of money, products and information among the parts that comprise the supply chain such as manufacturers, distributors, retailers and customers. Consequently, defining the facility locations of its part either by selecting from already existing or by setting up new facilities, it is one of the most strategic decisions for the managers.

In order to understand the importance of the facility location problems, the Thesis is followed by the next four parts. The first part is an introduction to supply chain and supply chain management. Before applying the models, it is necessary to learn the basic concepts of the supply chain and what adds value to it.

The second part consists of the facility location models where they are presented by their mathematical formulation. Apart from their formulas, it is given their generic description about where they applied, what are their assumptions and what is their objective so that they can be comprehensible.

The third part of the Thesis concerns the computer programming of location models. Here, it is analyzed what programs and tools have been chosen in order to model the facility location problems. Finally, in the last part, there are stated some conclusions and suggestions for further research.

## **Keywords**

Supply chain

Location modelling

Gurobi

Python



Μοντελοποίηση προβλημάτων χωροθέτησης και εφαρμογές τους στην  
εφοδιαστική αλυσίδα

Φωτεινή Μπούρου



## Περίληψη

Στόχος αυτής της διπλωματικής είναι να μελετήσουμε τα προβλήματα χωροθέτησης εγκαταστάσεων και να λύσουμε την μαθηματική μοντελοποίησή τους μέσω κώδικα υπολογιστή. Τα προβλήματα χωροθέτησης εγκαταστάσεων αποτελούν ένα πολύ σημαντικό παράγοντα της στρατηγικής της διαχείρισης της εφοδιαστικής αλυσίδας. Η εφοδιαστική αλυσίδα περιλαμβάνει την ροή των χρήματων, των προϊόντων και της πληροφορίας αναμεσά στα μέρη που αποτελούν την εφοδιαστική αλυσίδα όπως είναι οι κατασκευαστές, οι διανομείς, οι λιανεμπόριο και οι καταναλωτές. Συνεπώς, ορίζοντας την χωροθέτηση των εγκαταστάσεων του κάθε τμήματος της εφοδιαστικής αλυσίδας είτε επιλέγοντας από τις ήδη υπάρχουσες εγκαταστάσεις είτε δημιουργώντας καινούργιες, είναι από τις πιο σημαντικές στρατηγικές αποφάσεις για τους μάνατζερ.

Για να γίνει κατανοητή η σημαντικότητα των μοντέλων χωροθέτησης εγκαταστάσεων, η διπλωματική εργασία ακολουθείται από τα επόμενα τέσσερα μέρη. Το πρώτο είναι μια εισαγωγή στην εφοδιαστική αλυσίδα και την διαχείριση της. Πριν, λοιπόν, να εφαρμοστούν τα μοντέλα, είναι απαραίτητο να μάθουμε κάποιες βασικές έννοιες της εφοδιαστικής αλυσίδας και πως προσθέτουμε αξία σε αυτήν.

Το δεύτερο μέρος αποτελείται από τα μοντέλα χωροθέτησης εγκαταστάσεων όπου παρουσιάζονται οι μαθηματικές τους εξισώσεις. Πέρα από αυτές, δίνεται μια γενική περιγραφή για το που εφαρμόζονται, ποιες είναι οι προϋποθέσεις τους και ποια είναι τα ζητούμενα τους ώστε να μπορέσουν να γίνουν κατανοητά.

Το τρίτο μέρος της διπλωματικής εργασίας αφορά τον προγραμματισμό των μοντέλων αυτών. Εδώ αναλύονται τα προγράμματα και τα εργαλεία που χρησιμοποιώντας ώστε να μοντελοποιηθούν τα προβλήματα χωροθέτησης εγκατάστασης. Τέλος, παρατίθενται κάποια συμπεράσματα και κάποιες προτάσεις για περαιτέρω έρευνα.

### Λέξεις – Κλειδιά

Εφοδιαστική αλυσίδα

Μοντέλα χωροθέτησης

Gurobi

Python



## Table of contents

Abstract .....	ix
Περίληψη.....	xiii
Table of contents .....	xv
Table of figures .....	xix
Table of acronyms .....	xxi
1. A brief introduction to supply chain management .....	1
1.2 Supply chain value .....	1
1.3 Storage.....	2
1.4 Inventory .....	2
1.5 Location.....	2
1.6 Transportation .....	2
1.7 Information.....	2
1.8 Network.....	3
1.8 Environmental impact .....	4
1.9 Recent trends in academic research .....	4
2. Mathematical models of location problems .....	6
2.1 Set covering model.....	6
2.1.1 Generic description .....	6
2.1.2 Mathematical formulation .....	7
2.1.3 Example 1 .....	8
2.1.4 Example 2.....	10
2.1.5 Example 3.....	11
2.2 Set covering model extension .....	14

2.2.1 Generic description .....	14
2.2.2 Mathematical formulation .....	14
2.2.3 Example 1 .....	15
2.2.4 Example 2 .....	17
2.3 Maximum covering location model .....	20
2.3.1 Generic description .....	20
2.3.2 Mathematical formulation .....	20
2.3.3 Example 1 .....	21
2.3.4 Example 2 .....	24
2.3.5 Example 3 .....	25
2.4 Maximum expected covering location model .....	29
2.4.1 Generic description .....	29
2.4.2 Mathematical formulation .....	29
2.4.3 Example 1 .....	30
2.4.4 Example 2 .....	33
2.5 Vertex P-center location model .....	36
2.5.1 Generic description .....	36
2.5.2 Mathematical formulation .....	36
2.5.3 Example 1 .....	38
2.5.4 Example 2 .....	41
2.5.5 Example 3 .....	44
2.5.6 Example 4 .....	46
2.6 P-median location model .....	50
2.6.1 Generic description .....	50
2.6.2 Mathematical formulation .....	50



2.6.3 Example 1 .....	51
2.6.4 Example 2 .....	54
2.6.5 Example 3 .....	56
2.6.6 Example 4 .....	59
2.6.7 Example 5 .....	61
2.7 Uncapacitated fixed charge facility location problems .....	65
2.7.1 Generic description .....	65
2.7.2 Mathematical formulation .....	65
2.7.3 Example 1 .....	66
2.7.4 Example 2 .....	69
2.7.5 Example 3 .....	72
2.7.6 Example 4 .....	74
2.8 Capacitated fixed charge facility location problems .....	78
2.8.1 Generic description .....	78
2.8.2 Mathematical formulation .....	78
2.8.3 Example 1 .....	79
2.8.4 Example 2 .....	82
2.8.5 Example 3 .....	85
3. Computer programming of location models .....	89
3.1 The choice of Python .....	89
3.2 The choice of Gurobi .....	90
3.3 Object-oriented programming .....	90
3.4 Coding remarks .....	90
4. Conclusion and suggestions for further research .....	92
4.1 Code improvement .....	92

4.1.1 Continuation of code .....	92
4.1.2 Open source.....	92
4.1.3 File-based input .....	92
4.1.4 Output visualization .....	93
4.1.5 Automated testing .....	93
4.2 Mathematical modelling of location problems.....	93
4.2.1 Lack of verifiable models.....	94
4.2.2 Lack of a holistic approach .....	94
4.2.3 Lack of modern routing impact .....	94
4.2.4 Load balancing .....	94
4.2.5 Digital twin.....	95
References .....	96

## Table of figures

Figure 1 Three supply chain flows, (Stanton, 2018) .....	1
Figure 2 Nodes and links in a supply chain, (Stanton, 2018).....	3
Figure 3 SC optimization literature categorization, (Karakostas & Sifaleras, 2021).....	4
Figure 4 SC sustainability indicators categorization, (Karakostas & Sifaleras, 2021) .....	5
Figure 5 A visualization of the Set Covering Model network .....	8
Figure 6 A visualization of the extended Set Covering Model network.....	15
Figure 7 A visualization of the extended Maximum Covering Location Model network .....	22
Figure 8 A visualization of the extended Maximum Expected Covering Covering Location Model network .....	30
Figure 9 A visualization of the Vertex P-Center Model network .....	38
Figure 10 A visualization of the P-median Location Model network.....	51
Figure 11 A visualization of the Uncapacitated Fixed Charge Facility Location Model network .....	66
Figure 12 A visualization of the Capacitated Fixed Charge Facility Location Model network .....	79
Figure 13 Capacitated facility location visualization with matplotlib, (Buchanan, 2022).....	93

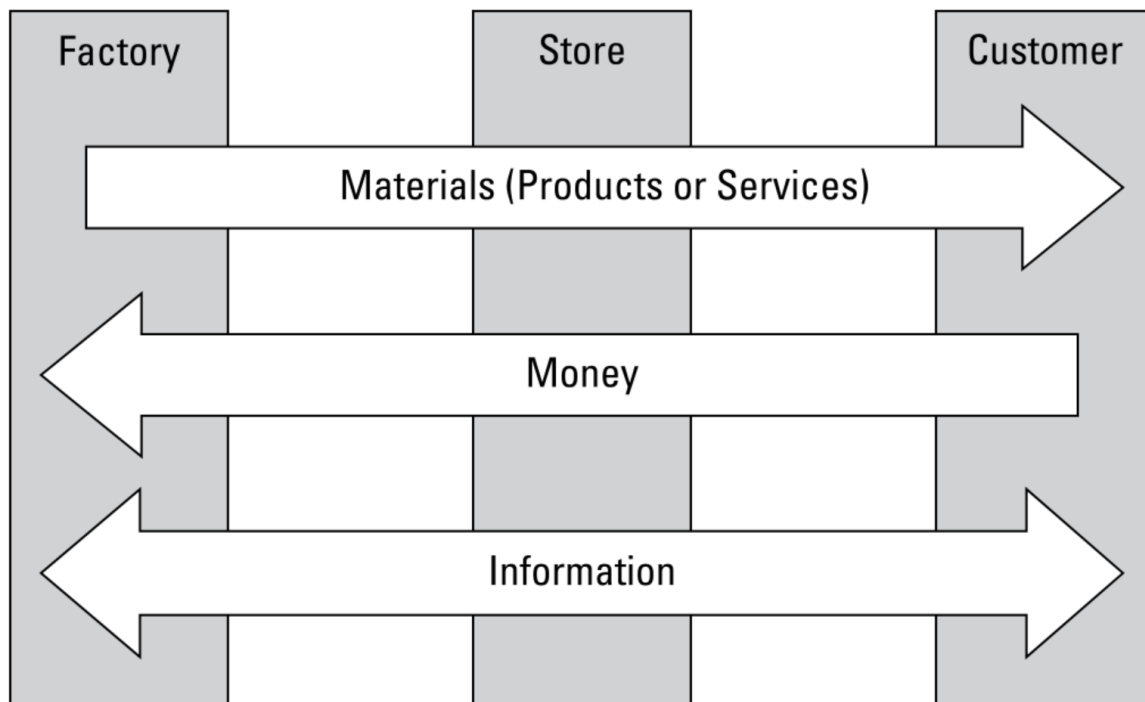


## Table of acronyms

Acronym	Explanation
<b>SC</b>	Supply chain
<b>SCM</b>	Supply chain management
<b>IDE</b>	Integrated development environment
<b>MCLP</b>	Maximum covering location problem
<b>UFL, UFLP</b>	Uncapacitated fixed-charge facility location problem
<b>CLF, CFLP</b>	Capacitated fixed-charge facility location problem

## 1. A brief introduction to supply chain management

Supply Chain (SC) consists of suppliers, manufacturers, distributors, retailers and customers. The customers are the bottom of SC, the downstream part. The suppliers are the top of SC, the upstream part. All of these parts are connected by flows of information, products and money. For example, manufacturers procure raw material by suppliers, they produce goods that distributors issue to the retailers from whom customers buy the final products. This example shows a simple flow of products. However, in order to flow the products downstream, money flows upstream. In addition, customers give in information to the retailers about what they need and this information flows upstream (Chopra, 2019).



**Figure 1 Three supply chain flows, (Stanton, 2018)**

### 1.2 Supply chain value

The price that buyers are willing to pay for a product minus the cost that sustains the SC is the called value of SC. This maximization of the value of SC is the goal of the Supply Chain Management (SCM). Consequently, SCM must take decisions centered around the value of SC (Chopra, 2019). In order to achieve the maximum value of SC, managers of SC must consider the following cost drivers: production, inventory, location, transportation and information (Hugos, 2018).

These cost drivers are very important for a SC because they can add or subtract into the value of SC. Hence SC managers take into account all of these factors and combine them in a such a way so that they can achieve the maximum value of SC through information sharing, planning, resource allocation and personnel coordination that is involved in SC (Stanton, 2018).

### **1.3 Storage**

Production concerns not only making goods, but also their storage. Consequently, the capacity of warehouses plays a key role. The more capacity a warehouse will have, the better will be for a company as it will respond directly at a possible increase at product demand. However, capacity is very costly and it can prove to be costlier if it storage facilities and stored goods or products are not used for a long period of time. As a result, maintaining an optimal compromise in SC and production is a rather tedious task (Hugos, 2018).

### **1.4 Inventory**

Inventory concerns the raw materials, semi-finished products and final products that are reserved by manufacturers, distributors and retailers. Large quantities of inventory can make a SC responsive, but this can also be very costly for a company (Hugos, 2018).

### **1.5 Location**

Location concerns the site of facilities that involved in a SC and their activities. Decisions about facilities affects their cost, the proximity to suppliers, and customers, taxes, the available and skillful workforce. Location decisions are very crucial as through them the products delivered to all parts of the SC and of course a foundation of new facilities is one of the most costly decisions (Hugos, 2018).

### **1.6 Transportation**

Transportation concerns the modes of transports that are used in order to transfer all goods either raw material or final products downstream and upstream as well. Decisions about transportation are about the means of transport, their routing and networks so that products can flow expectedly and reliably to all parts of a SC (Hugos, 2018).

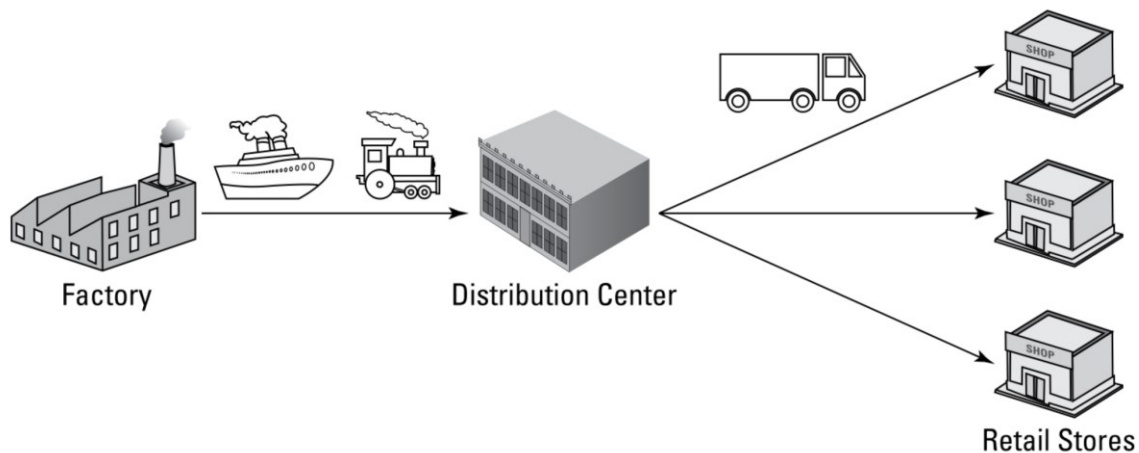
### **1.7 Information**

Information gathering is of paramount importance in modern SCM. Information such as current demand, tracking, exchange rates, network blockades is a crucial asset to SC decision making.

All this information gathering and processing can be used for short- or long-term forecasting, planning of production, storage, mass-transportation and final delivery. Information sharing can also lead to a tighter integration of suppliers and cost reduction. In case a real-time information system is available, then routing can be planned or changed immediately when an anomaly is detected and of course provide tracking to the final customer which also adds a level of transparency (Hugos, 2018).

## 1.8 Network

One of the most important factors and the topic of this Thesis that can add value to a SC is the design of the network of the SC. A network in a SC consists of nodes and links. Nodes can be warehouses, distribution centers and retailers. The links are the connections between two nodes and usually comprise the means of transport. As a result, a network shows how the products flow from one node to another via links (Stanton, 2018).



**Figure 2 Nodes and links in a supply chain, (Stanton, 2018)**

SC managers can change the nodes and the links in order to add value and minimize the costs. This is called network optimization. For example, if an alternative factory is closer to the distribution center, a company could source products from this factory partially or exclusively and save money as the cost of transport would be lower due to smaller distance between these nodes. Another scenario is that perhaps a new highway is planned and there will be an extra link in the network that could affect routing and future location selection (Stanton, 2018).



## 1.8 Environmental impact

During the decision phase of locating facilities, apart from the cost, lately we need or should take into account social and environmental impacts as well. There are several modern concerns on manufacturing and transporting regarding the efficient use of natural resources, work conditions and employment and of course greenhouse gas emissions. Currently there are open problem under academic research on how these factors can be taken into account in location modeling (Skouri, Sifaleras, & Konstantaras, 2018).

## 1.9 Recent trends in academic research

Currently there is a great range of facility location factors and methods research, particularly by taking into account the SC sustainability. Research publications can be categorized into sustainability indicators (economic, environmental, societal), decision levels (strategic, tactical, operational) and the optimization methods or solution algorithms (exact, approximate, heuristic, hybrid) (Karakostas & Sifaleras, 2021).

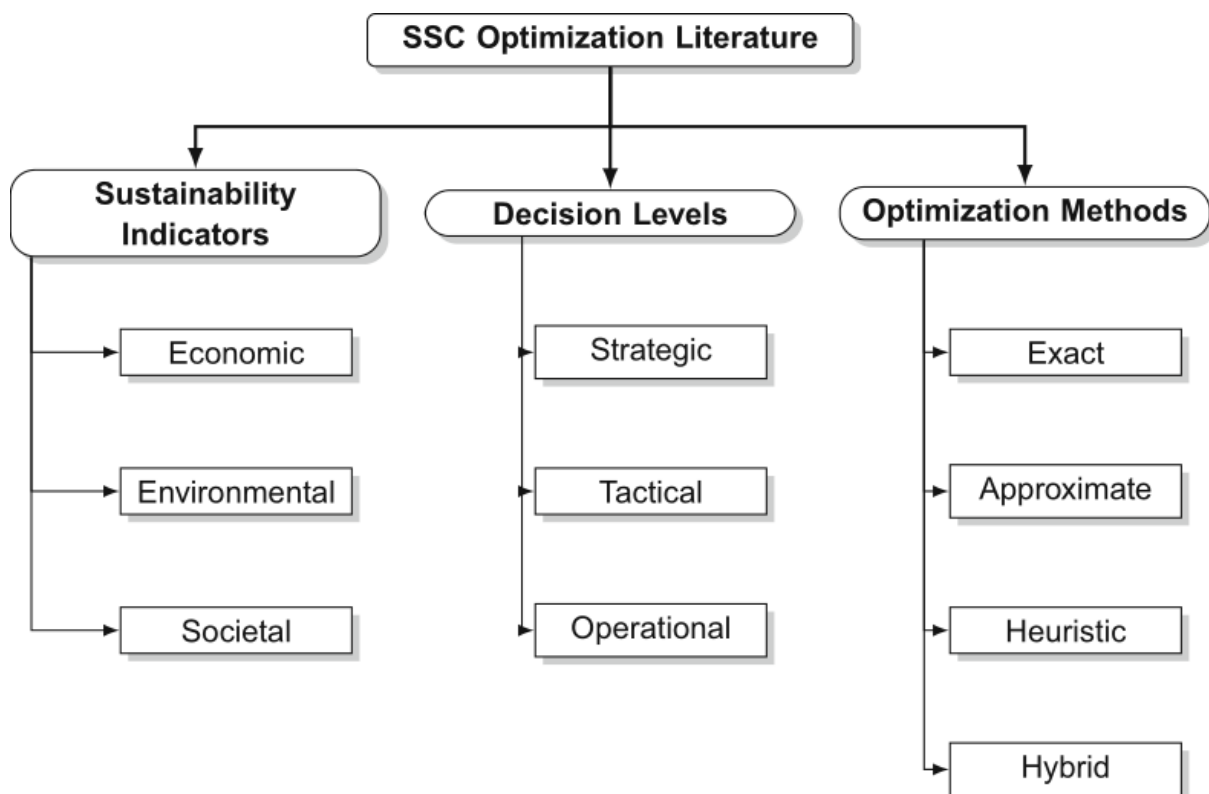
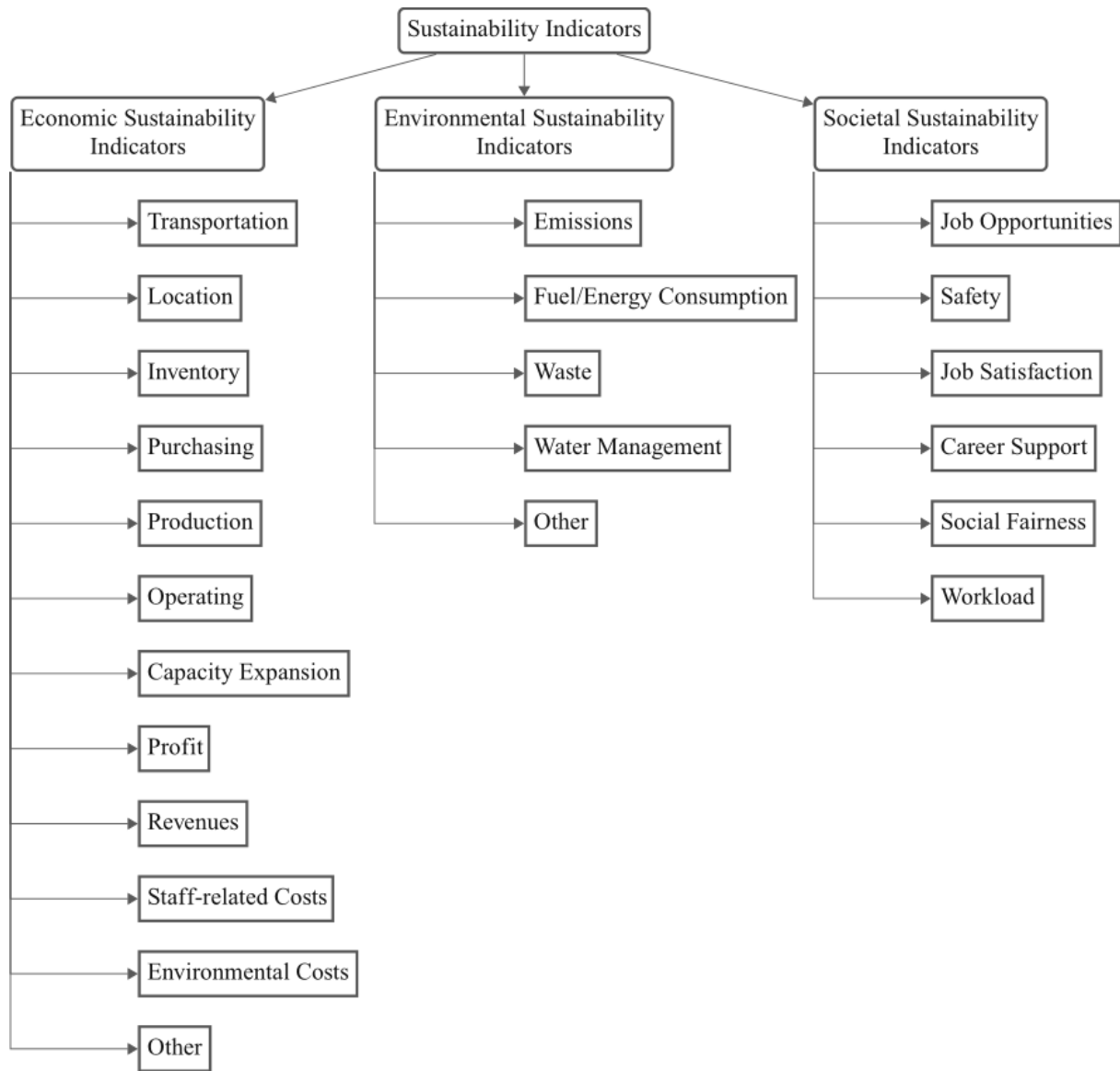


Figure 3 SC optimization literature categorization, (Karakostas & Sifaleras, 2021)

Furthermore, sustainability indicators are quite a wide-ranging field with economic, environmental and societal categories and each of these categories contains numerous indicators. (Karakostas & Sifaleras, 2021). This further exemplifies the complexity of a modern SC location problem and why mathematical modeling and optimization is needed.



**Figure 4 SC sustainability indicators categorization, (Karakostas & Sifaleras, 2021)**

## 2. Mathematical models of location problems

In location problems there is a network that connects facilities that serve the corresponding customers. Consequently, there are demand nodes  $i$  that belong to a set of demand nodes  $I$ , and candidate facilities locations  $j$  that belong to a set of candidate facilities locations  $J$ . There are several model formulations that can be used to extract information on where to place facilities, however, in this Thesis the following 8 models will be discussed.

- Set covering
- Set covering with extension
- Maximum covering location
- Maximum expected covering location
- Vertex P-center
- P-median
- Uncapacitated fixed-charge facility location
- Capacitated fixed-charge facility location

### 2.1 Set covering model

#### 2.1.1 Generic description

In this model the goal is to find the optimum facilities location placement so all nodes of demand can be served at minimum cost (Sitepu, et al., 2019). Ultimately, the model aims to find the minimum facilities that will cover all the demand nodes based on the given coverage distance.

For solving a set covering problem there must be given demand nodes  $i \in I$ , candidate facility locations  $j \in J$ , distances between demand nodes, candidate facility locations, and coverage distance which will determine if a demand node  $i$  can be covered by facility  $j$  as a matrix  $a_{ij}$  (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). This way the set covering model can be applied on product distribution and storage placement.

Set covering models have been applied in many areas of supply chain management. Some examples are the analysis of markets (Storbeck, 1988), crew scheduling (Ceria, Nobili, & Sassano, 1998), deployment of emergency services (Toregas, Swain, ReVelle, & Bergman, 1971), (Eaton, Sánchez, Lantigua, & Morgan, 1986) and nature reserve selection (Church, Stoms, & Davis, Reserve selection as a maximal covering location problem, 1996).

It is worth remarking the example of covering model applied by (Hale & Moberg, 2005). Using covering model, (Hale & Moberg, 2005) managed to solve the location problem of secure backup facilities in case of emergency. The emergency could be caused due to natural, technological, or even human hazards.

Another notable problem at which is applied the set covering model is the line planning problem. After a survey about the demand of passengers about the lines which are the routes, (Caprara, Kroon, Monaci, Peeters, & Toth, 2007) and (van Hoesel, Goossens, & Kroon, 2004) used set covering model to find the optimum lines that maximizes the available service to the passengers (Caprara, Kroon, Monaci, Peeters, & Toth, 2007) (van Hoesel, Goossens, & Kroon, 2004).

### 2.1.2 Mathematical formulation

The mathematical formulation is the following:

Decision variables

$$X_j = \begin{cases} 1 & \text{if we locate a facility at candidate site } j \in J \\ 0 & \text{if not} \end{cases} \quad (1)$$

Constants

$$a_{ij} = \begin{cases} 1 & \text{if candidate site } j \in J \text{ can cover demands at node } i \in I \\ 0 & \text{if not} \end{cases} \quad (2)$$

$$f_j = \text{cost of locating a facility at candidate site } j \in J \quad (3)$$

Minimize

$$\sum_{j \in J} f_j X_j \quad (4)$$

Subject to

$$\sum_{j \in J} a_{ij} X_j \geq 1 \quad \forall i \in I \quad (5)$$

$$X_j \in \{0,1\} \quad \forall j \in J \quad (6)$$

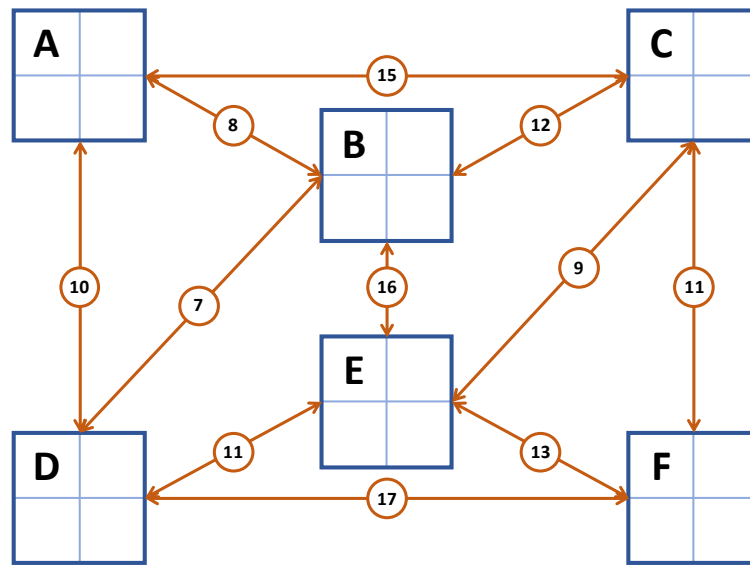
The objective function minimizes the total costs of the facilities that have been chosen. Then, the first constraint specifies that every demand node  $i \in I$  must be covered by at least one

facility. And the last constraint declares that the variable  $X_j$  is binary (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013).

It must be noticed that the set covering model appears to have some drawbacks. The first drawback is that in order to cover the total demand, many facilities are usually required. Consequently, the cost of facility location is huge. Furthermore, there are usually many alternative optimal solutions that can be applied. Finally, the model does not take into account the level of demand at the nodes (Daskin, What you should know about location modeling, 2008).

### 2.1.3 Example 1

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013).



**Figure 5 A visualization of the Set Covering Model network**

The mathematical formulation of this model is as follows. First, we need to minimize the number of nodes selected

$$\text{Minimize } Z = X_A + X_B + X_C + X_D + X_E + X_F$$

then we need to ensure that nodes are covered by the selection

$$\begin{array}{rcl}
 X_A + X_B & + X_D & \geq 1 \\
 X_A + X_B & + X_D & \geq 1 \\
 & X_C & + X_E + X_F \geq 1 \\
 X_A + X_B & + X_D + X_E & \geq 1 \\
 & X_C + X_D + X_E & \geq 1 \\
 & X_C & + X_F \geq 1
 \end{array}$$

and finally, we have to include the integrality constraint

$$X_A, X_B, X_C, X_D, X_E, X_F \in \{0,1\}$$

Executing the monolithic version of the code, we see that results match the solution of the book

$$X_C, X_D = 1, Z = 2$$

### 2.1.3.1 Code

```
#####
# Set covering model                                     #
# Chapter 4.2, p.125                                     #
# Network and Discrete Location Models, 2e, Daskin      #
#                                                       #
# Non-parametric solution                               #
# For covering distance 11                             #
#                                                       #
# Code by Foteini Bourou, 2022                         #
#                                                       #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    m = gp.Model("mip1")
    xa=m.addVar(vtype=GRB.BINARY, name="xa")
    xb=m.addVar(vtype=GRB.BINARY, name="xb")
    xc=m.addVar(vtype=GRB.BINARY, name="xc")
    xd=m.addVar(vtype=GRB.BINARY, name="xd")
    xe=m.addVar(vtype=GRB.BINARY, name="xe")
    xf=m.addVar(vtype=GRB.BINARY, name="xf")
    m.setObjective(xa + xb + xc + xd + xe + xf, GRB.MINIMIZE)
    m.addConstr(xa + xb + xd >=1, "c0")
    m.addConstr(xc + xe + xf >=1, "c1")
    m.addConstr(xa + xb + xd + xe >=1, "c2")
    m.addConstr(xc + xd + xe >=1, "c3")
    m.addConstr(xc + xf >=1, "c4")
    m.optimize()
    print('#####3')
    for v in m.getVars():
```

```
print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```

### 2.1.4 Example 2

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). The problem formulation is the same as the previous paragraph. Executing the parametric version of the code, we see that results match the solution of the book.

#### 2.1.4.1 Code

```
#####
# Set covering model
# Chapter 4.2, p.125
# Network and Discrete Location Models, 2e, Daskin
#
# Parametric solution
# For covering distance 11
#
#
# Code by Foteini Bourou, 2022
#
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set locations and demand nodes
    locations = ['a', 'b', 'c', 'd', 'e', 'f']

    # const_ajj -> If candidate site j can cover demand at node i
    const_ajj = [[1, 1, 0, 1, 0, 0],
                  [1, 1, 0, 1, 0, 0],
                  [0, 0, 1, 0, 1, 1],
                  [1, 1, 0, 1, 1, 0],
                  [0, 0, 1, 1, 1, 0],
                  [0, 0, 1, 0, 0, 1]]

    #####

    m = gp.Model('LSCP')

    # Add variables
    var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x{loc}') for loc in locations]
```

```
# Constraint
# sum(a_ij * xj) >= 1 // sum over j // for every i
for ai in const_ajj:
    temp_lin_expr = gp.LinExpr()
    for ajj, xj in zip(ai, var_xj):
        temp_lin_expr.add(xj, ajj)
    m.addConstr(temp_lin_expr >= 1, f'sum(ai * var_xj) >= 1')

# Constraint
# sum(a_ij * xj) - S_i >= 1 // sum over j // for every i
i = 0
for ai in const_ajj:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for ajj, xj in zip(ai, var_xj):
        temp_lin_expr.add(xj, ajj)
    m.addConstr(temp_lin_expr >= 1, f'sum(a_{i}_j * x_j >= 1')

# Objective function
# Minimize sum xj
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####3')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```

### 2.1.5 Example 3

This is an example from (Sitepu, et al., 2019) . The mathematical formulation of this model is as follows. First, we need to minimize the number of nodes selected

$$\text{Minimize } Z = X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + X_8$$

then we need to ensure that nodes are covered by the selection



$$\begin{array}{rcl}
 X_1 & & \geq 1 \\
 X_2 & & \geq 1 \\
 X_2 + X_3 & & \geq 1 \\
 & X_4 & + X_6 \geq 1 \\
 & & X_5 \geq 1 \\
 & X_4 & + X_6 \geq 1 \\
 & & X_7 \geq 1 \\
 & & X_8 \geq 1
 \end{array}$$

and finally, we have to include the integrality constraint

$$X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8 \in \{0,1\}$$

Running the parametric type code, we see that results from Gurobi solver and the results from mathematical solution are the same. There is a perfect match.

$$X_1, X_2, X_5, X_6, X_7, X_8 = 1, Z = 6$$

### 2.1.5.1 Code

```
#####
# Set covering model                                     #
# p.2                                                    #
# Set covering models in optimizing the emergency unit location #
# of health facility in Palembang, Sitepu et al.        #
#                                                         #
#                                                         #
#                                                         #
# Code by Foteini Bourou, 2022                           #
#                                                         #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set locations and demand nodes
    locations = ['1 - Ilir Timur II',
                '2 - Kalidoni',
                '3 - Kemuning',
                '4 - Plaju',
                '5 - Sako',
                '6 - Seberang Ulu II',
                '7 - Sematan Borang',
                '8 - Sukarami']

    # const_ajj -> If candidate site j can cover demand at node i
    const_ajj = [[1, 0, 0, 0, 0, 0, 0, 0],
                 [0, 1, 0, 0, 0, 0, 0, 0],
                 [0, 1, 1, 0, 0, 0, 0, 0],
```

```

[0, 0, 0, 1, 0, 1, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 1, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 1]]

#####

m = gp.Model('LSCP')

# Add variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x{loc}') for loc in locations]

# Constraint
# sum(a_ij * xj) >= 1 // sum over j // for every i
for ai in const_aij:
    temp_lin_expr = gp.LinExpr()
    for aij, xj in zip(ai, var_xj):
        temp_lin_expr.add(xj, aij)
    m.addConstr(temp_lin_expr >= 1, f'sum(ai * var_xj) >= 1')

# Constraint
# sum(a_ij * xj) - S_i >= 1 // sum over j // for every i
i = 0
for ai in const_aij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for aij, xj in zip(ai, var_xj):
        temp_lin_expr.add(xj, aij)
    m.addConstr(temp_lin_expr >= 1, f'sum(a_{i}_j * x_j >= 1')

# Objective function
# Minimize sum xj
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####3')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()

```

## 2.2 Set covering model extension

### 2.2.1 Generic description

The set covering model will provide alternate configurations of optimal location selection as the distances increase. To mitigate this, we provide an extension to the original problem such that it has the goal is to find a combination of the given facilities that maximizes the number of demand nodes covered twice. So, while the set covering model will provide e.g. 5 equally optimal configurations, this extension will select the combination that covers most nodes twice.

### 2.2.2 Mathematical formulation

The mathematical formulation is the following:

Decision variables

$$X_j = \begin{cases} 1 & \text{if we locate a facility at candidate site } j \in J \\ 0 & \text{if not} \end{cases} \quad (7)$$

$$S_i = \begin{cases} 1 & \text{if demand node } i \in I \text{ is covered at least twice} \\ 0 & \text{if not} \end{cases} \quad (8)$$

Constants

$$a_{ij} = \begin{cases} 1 & \text{if candidate site } j \in J \text{ can cover demands at node } i \in I \\ 0 & \text{if not} \end{cases} \quad (9)$$

Minimize

$$(|I| + 1) \sum_{j \in J} X_j - \sum_{i \in I} S_i \quad (10)$$

Subject to

$$\sum_{j \in J} a_{ij} X_j - S_i \geq 1 \quad \forall i \in I \quad (11)$$

$$X_j \in \{0,1\} \quad \forall j \in J \quad (12)$$

$$S_i \in \{0,1\} \quad \forall i \in I \quad (13)$$

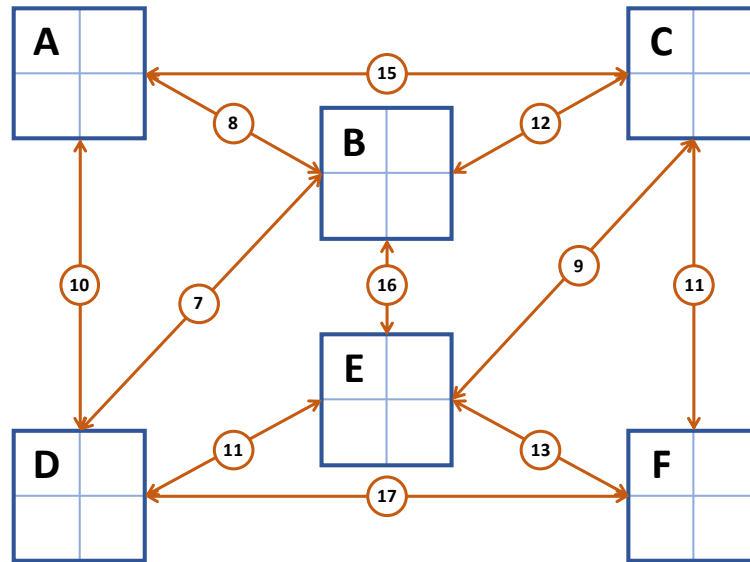
The objective function maximizes the number of nodes covered twice while minimizing the amount of facilities placed. Then, the first constraint specifies that every demand node  $i \in I$  must be covered by at least one facility. And the last constraints declares that the variables  $X_j, S_j$

are binary (Daskin & Stern, A Hierarchical Objective Set Covering Model for Emergency Medical Service Vehicle Deployment, 1981). An important notice on this model is that it is erroneously presented in the literature (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013).

It must be noticed that the set covering model as presented here does not take into account already existing locations. There are many ways presented in literature on how to achieve this (Plane & Hendrick, 1977).

### 2.2.3 Example 1

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013).



**Figure 6 A visualization of the extended Set Covering Model network**

First, we need to minimize the number of nodes selected and maximize the amount of nodes covered twice. In this case the cardinality of the set is  $|I| = 6$ .

$$\text{Minimize } Z = 7 * (X_A + X_B + X_C + X_D + X_E + X_F) - (S_A + S_B + S_C + S_D + S_E + S_F)$$

then we need to ensure that nodes are covered by the selection and that  $S_i$  variables are constrained – else they would all be equal to 1

$$\begin{array}{rclclcl}
 X_A + & X_B + & & X_D & & -S_A & \geq 1 \\
 X_A + & X_B + & & X_D & & -S_B & \geq 1 \\
 & & X_C + & & X_E + & X_F & -S_C \geq 1 \\
 X_A + & X_B + & & X_D + & X_E & & -S_D \geq 1 \\
 & & X_C + & X_D + & X_E & & -S_E \geq 1 \\
 & & X_C + & & & X_F & -S_F \geq 1
 \end{array}$$

and finally, we have to include the integrality constraint

$$X_A, X_B, X_C, X_D, X_E, X_F, S_A, S_B, S_C, S_D, S_E, S_F \in \{0,1\}$$

Executing the monolithic version of the code, we see that results match the solution of the book

$$X_C, X_E, S_C, S_E, S_F = 1, Z = 11$$

Executing the monolithic version of the code, we see that results match the solution of the book.

### 2.2.3.1 Code

```
#####
# Set covering model - extension                                     #
# Chapter 4.4, p.140                                              #
# Network and Discrete Location Models, 2e, Daskin               #
#                                                                    #
# Non-parametric solution                                         #
#                                                                    #
# One extension of the set covering problem is to select the     #
# combination of sites that maximizes the number of demand nodes #
# covered twice from among the alternate optima to the           #
# set covering problem                                           #
#                                                                    #
#                                                                    #
# Code by Foteini Bourou, 2022                                     #
#                                                                    #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    m = gp.Model("mip1")
    xa=m.addVar(vtype=GRB.BINARY, name="xa")
    xb=m.addVar(vtype=GRB.BINARY, name="xb")
    xc=m.addVar(vtype=GRB.BINARY, name="xc")
    xd=m.addVar(vtype=GRB.BINARY, name="xd")
    xe=m.addVar(vtype=GRB.BINARY, name="xe")
    xf=m.addVar(vtype=GRB.BINARY, name="xf")
```

```

sa=m.addVar(vtype=GRB.BINARY, name="sa")
sb=m.addVar(vtype=GRB.BINARY, name="sb")
sc=m.addVar(vtype=GRB.BINARY, name="sc")
sd=m.addVar(vtype=GRB.BINARY, name="sd")
se=m.addVar(vtype=GRB.BINARY, name="se")
sf=m.addVar(vtype=GRB.BINARY, name="sf")

m.setObjective(7 * (xa + xb + xc + xd + xe + xf) - (sa + sb + sc + sd + se +
sf), GRB.MINIMIZE)

m.addConstr(xa + xb + xc + xd - sa >= 1, "c1")
m.addConstr(xa + xb + xc + xd - sb >= 1, "c2")
m.addConstr(xa + xb + xc + xe + xf - sc >= 1, "c3")
m.addConstr(xa + xb + xd + xe - sd >= 1, "c4")
m.addConstr(xc + xd + xe + xf - se >= 1, "c5")
m.addConstr(xc + xe + xf - sf >= 1, "c6")

m.optimize()
print('#####3')
for v in m.getVars():
    print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()

```

## 2.2.4 Example 2

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). The problem formulation is the same as the previous paragraph. Executing the parametric version of the code, we see that results match the solution of the book.

### 2.2.4.1 Code

```

#####
# Set covering model - extension                                     #
# Chapter 4.4, p.140                                                #
# Network and Discrete Location Models, 2e, Daskin                 #
#                                                                    #
# Parametric solution                                              #
#                                                                    #
# One extension of the set covering problem is to select the      #
# combination of sites that maximizes the number of demand nodes  #
# covered twice from among the alternate optima to the            #
# set covering problem                                             #
#                                                                    #
#                                                                    #
#                                                                    #
# Code by Foteini Bourou, 2022                                     #
#                                                                    #
#####

```

```
import gurobipy as gp
from gurobipy import GRB

def main():
    # Set locations and demand nodes
    locations = ['a', 'b', 'c', 'd', 'e', 'f']

    # const_ajj -> If candidate site j can cover demand at node i
    const_ajj = [[1, 1, 1, 1, 0, 0],
                  [1, 1, 1, 1, 0, 0],
                  [1, 1, 1, 0, 1, 1],
                  [1, 1, 0, 1, 1, 0],
                  [0, 0, 1, 1, 1, 1],
                  [0, 0, 1, 0, 1, 1]]

    #####

    m = gp.Model("LSCPe")

    # Add variables
    var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x{loc}') for loc in locations]
    var_sj = [m.addVar(vtype=GRB.BINARY, name=f's{loc}') for loc in locations]

    # Constraint
    # sum(a_ij * xj) - S_i >= 1 // sum over j // for every i
    i = 0
    for ai, si in zip(const_ajj, var_sj):
        i += 1
        temp_lin_expr = gp.LinExpr()
        for aij, xj in zip(ai, var_xj):
            temp_lin_expr.add(xj, aij)
        temp_lin_expr.add(si, -1)
        m.addConstr(temp_lin_expr >= 1, f'sum(a_{i}_j * x_j - s_{i}) >= 1')

    # Objective function
    # Minimize sum xj
    temp_lin_expr = gp.LinExpr()
    xj_sum_factor = len(var_xj) + 1
    for xj in var_xj:
        temp_lin_expr.add(xj, xj_sum_factor)
    for sj in var_sj:
        temp_lin_expr.add(sj, -1)
    m.setObjective(temp_lin_expr, GRB.MINIMIZE)

    m.optimize()
    print('#####3')
    for v in m.getVars():
        if v.X:
            print('%s %g' % (v.VarName, v.X))

    print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```





## 2.3 Maximum covering location model

### 2.3.1 Generic description

The goal of this model is to find the appropriate facilities that maximize the number of covered demands. It requires to have a known coverage distance  $D_c$ , the number of facilities to locate  $P$  and the demand level at demand nodes. This model takes into account the demand level for each demand node as the coverage distance as criterion. This comes in contrast to the set covering model where they proved to be extremely large (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013) (Daskin, Snyder, & Berger, Facility Location in Supply Chain Design, 2005).

Maximum covering location models (MCLPs) have been a powerful and widely useful tool too in many planning processes. MCLP gives the opportunity to managers to maximize their benefits through optimum distribution of limited resources. Some examples are the placement of fire stations (Indriasari, Mahmud, Ahmad, & Shariff, 2010) and health centers (Bennett, Eaton, & Church, 1982), (Griffin, Scherrer, & Swann, 2008), (Ratick, Osleeb, & Hozumi, 2009), (Verter & Lapierre, 2002).

More particular, about health sector, managers in Austin, Texas used MCLP. They succeeded in selecting the most appropriate permanent facilities in case of emergency medical call so that 95% of the emergency calls would be reached within five minutes. (Eaton, Daskin, Simmons, Bulloch, & Jansma, 1985) (Swersey, 1994).

### 2.3.2 Mathematical formulation

The mathematical formulation is the following:

Decision variables

$$Z_i = \begin{cases} 1 & \text{if demand node } i \in I \text{ is covered} \\ 0 & \text{if not} \end{cases} \quad (14)$$

Constants

$$h_i = \text{demand at node } i \in I \quad (15)$$

$$P = \text{number of facilities to locate} \quad (16)$$

Minimize

$$\sum_{i \in I} h_i Z_i \quad (17)$$

Subject to

$$Z_i \leq \sum_{j \in J} a_{ij} X_j \quad \forall i \in I \quad (18)$$

$$\sum_{j \in J} X_j \leq P \quad (19)$$

$$X_j \in \{0,1\} \quad \forall j \in J \quad (20)$$

$$Z_i \in \{0,1\} \quad \forall i \in I \quad (21)$$

The objective function maximizes the number of covered demands. The first constraint declares that demand at node  $i \in I$  is covered only in case that at least one facility location that cover demand node  $i$  is selected. The second constraint declares that no more than  $P$  facilities can be located. Finally, the last two constraints declare that the variables  $X_j$  and  $Z_i$  are binary.

In conclusion, it must be noticed that the maximum covering location model takes into account the level of demand on the nodes. Consequently, it prioritizes the nodes with the highest level of demand. The larger the number of facility locations it is, the more demand nodes will be covered.

### 2.3.3 Example 1

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013), where  $h$  is the node demand.

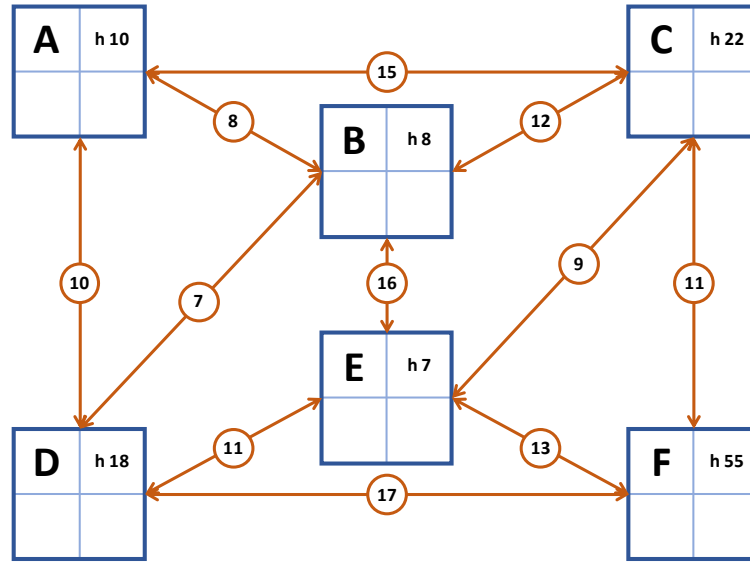


Figure 7 A visualization of the extended Maximum Covering Location Model network

The mathematical formulation of this model is as follows. First, we need to maximize the covered demands

$$\text{Minimize } Z = 10Z_A + 8Z_B + 22Z_C + 18Z_D + 7Z_E + 55Z_F$$

then we need to ensure that supply covers the demand

$$\begin{array}{rcll} X_A + & X_B + & & X_D & \geq Z_A \\ X_A + & X_B + & & X_D & \geq Z_B \\ & & X_C + & & X_E + & X_F & \geq Z_C \\ X_A + & X_B + & & X_D + & X_E & \geq Z_D \\ & & X_C + & X_D + & X_E & \geq Z_E \\ & & X_C + & & & X_F & \geq Z_F \end{array}$$

and finally, we have to include the integrality constraint

$$X_A, X_B, X_C, X_D, X_E, X_F, Z_A, Z_B, Z_C, Z_D, Z_E, Z_F \in \{0,1\}$$

Executing the monolithic version of the code, we see that results match the solution of the book

$$X_C, Z_C, Z_E, Z_F = 1, Z = 84$$

### 2.3.3.1 Code

```
#####
# Maximum covering location model                                     #
# Chapter 4.5, p.143                                                #
# Network and Discrete Location Models, 2e, Daskin                 #
#                                                                    #
```

```
# Non-parametric solution                                     #
#                                                             #
#                                                             #
#                                                             #
# Code by Foteini Bourou, 2022                               #
#                                                             #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    m = gp.Model("mip1")
    xa=m.addVar(vtype=GRB.BINARY, name="xa")
    xb=m.addVar(vtype=GRB.BINARY, name="xb")
    xc=m.addVar(vtype=GRB.BINARY, name="xc")
    xd=m.addVar(vtype=GRB.BINARY, name="xd")
    xe=m.addVar(vtype=GRB.BINARY, name="xe")
    xf=m.addVar(vtype=GRB.BINARY, name="xf")
    za=m.addVar(vtype=GRB.BINARY, name="za")
    zb=m.addVar(vtype=GRB.BINARY, name="zb")
    zc=m.addVar(vtype=GRB.BINARY, name="zc")
    zd=m.addVar(vtype=GRB.BINARY, name="zd")
    ze=m.addVar(vtype=GRB.BINARY, name="ze")
    zf=m.addVar(vtype=GRB.BINARY, name="zf")

    m.setObjective(10*za + 8*zb + 22*zc + 18*zd + 7*ze + 55*zf, GRB.MAXIMIZE)
    m.addConstr(xa + xb + xd >= za, "c0")
    m.addConstr(xa + xb + xd >= zb, "c1")
    m.addConstr(xc + xe + xf >= zc, "c2")
    m.addConstr(xa + xb + xd + xe >= zd, "c3")
    m.addConstr(xc + xd + xe >= ze, "c4")
    m.addConstr(xc + xf >= zf, "c5")
    m.addConstr(xa + xb +xc +xd +xe + xf <= 1, "c6")
    m.optimize()
    print('#####3')
    for v in m.getVars():
        print('%s %g' % (v.VarName, v.X))

    print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). Running the code in monolithic type, we see that results from Gurobi solver and the results from mathematical solution are the same. There is a perfect match.

### 2.3.4 Example 2

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). The problem formulation is the same as the previous paragraph. Executing the parametric version of the code, we see that results match the solution of the book.

#### 2.3.4.1 Code

```
#####
# Maximum covering location model                                     #
# Chapter 4.5, p.143                                                #
# Network and Discrete Location Models, 2e, Daskin                 #
#                                                                    #
# Parametric solution                                              #
#                                                                    #
#                                                                    #
# Code by Foteini Bourou, 2022                                     #
#                                                                    #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set locations and demand nodes
    locations = ['a', 'b', 'c', 'd', 'e', 'f']

    # Desired locations
    P = 1

    # const_ajj -> If candidate site j can cover demand at node i
    const_ajj = [[1, 1, 0, 1, 0, 0],
                 [1, 1, 0, 1, 0, 0],
                 [0, 0, 1, 0, 1, 1],
                 [1, 1, 0, 1, 1, 0],
                 [0, 0, 1, 1, 1, 0],
                 [0, 0, 1, 0, 0, 1]]

    # const_hi -> Demand at node j
    const_hi = [10, 8, 22, 18, 7, 55]

    #####

    m = gp.Model('MCLP')

    # Add variables
    # If warehouse is placed at j
    var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]
    # If demand at i is covered
    var_zj = [m.addVar(vtype=GRB.BINARY, name=f'z_{loc}') for loc in locations]
```

```
# Constraint
# sum(a_ij * xj) - zi >= 0 // sum over j // for every i
i = 0
j = 0
for ai, zi in zip(const_aij, var_zj):
    i += 1
    temp_lin_expr = gp.LinExpr()
    for aij, xj in zip(ai, var_xj):
        temp_lin_expr.add(xj, aij)
    temp_lin_expr.add(zi, -1.0)
    m.addConstr(temp_lin_expr >= 0, f'sum(a_{i}_j * x_j - z_{i}) >= 0')

# Constraint
# sum(X_j) <= P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr <= P, f'sum(var_xj) <= {P}')

# Objective function
# Maximize sum h_i*z_i
temp_lin_expr = gp.LinExpr()
for hi, zi in zip(const_hi, var_zj):
    temp_lin_expr.add(zi, hi)
m.setObjective(temp_lin_expr, GRB.MAXIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```

### 2.3.5 Example 3

This is an example from (Sitepu, et al., 2019). The mathematical formulation of this model is as follows. First, we need to maximize the covered demands

$$\text{Minimize } Z = 12Y_1 + 5Y_2 + 6Y_3 + 7Y_4 + 4Y_5 + 7Y_6 + 4Y_7 + 7Y_8$$

then we need to ensure that supply covers the demand

$$\begin{array}{rcl}
X_1 & & \geq Y_1 \\
X_2 & & \geq Y_2 \\
X_2 + X_3 & & \geq Y_3 \\
X_4 & + X_6 & \geq Y_4 \\
X_5 & & \geq Y_5 \\
X_4 & + X_6 & \geq Y_6 \\
X_7 & & \geq Y_7 \\
X_8 & \geq & Y_8
\end{array}$$

and finally, we have to include the integrality constraint

$$X_A, X_B, X_C, X_D, X_E, X_F, Y_A, Y_B, Y_C, Y_D, Y_E, Y_F \in \{0,1\}$$

Running the parametric version of the code, we see that the results from this code partially match the reported optimal solution. The objective function is equal but the reported variable values do not match.

$$X_1, X_2, X_4, X_5, X_8, Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_8 = 1, Z = 84$$

By forcing the solution to be the reported optimal solution, we get the same objective function, meaning that the optimal solution is not unique.

$$X_1, X_2, X_5, X_6, X_8, Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_8 = 1, Z = 84$$

### 2.3.5.1 Code

```
#####
# Maximum covering location model                                     #
# p.3                                                                #
# Set covering models in optimizing the emergency unit location      #
# of health facility in Palembang, Sitepu et al.                   #
#                                                                    #
#                                                                    #
#                                                                    #
# Code by Foteini Bourou, 2022                                       #
#                                                                    #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set Locations and demand nodes
    locations = ['1 - Ilir Timur II',
                '2 - Kalidoni',
                '3 - Kemuning',
                '4 - Plaju',
                '5 - Sako',
                '6 - Seberang Ulu II',
```

```

    '7 - Sematan Borang',
    '8 - Sukarami']

# Desired Locations
P = 5

# const_aij -> If candidate site j can cover demand at node i
const_aij = [[1, 0, 0, 0, 0, 0, 0, 0],
              [0, 1, 0, 0, 0, 0, 0, 0],
              [0, 1, 1, 0, 0, 0, 0, 0],
              [0, 0, 0, 1, 0, 1, 0, 0],
              [0, 0, 0, 0, 1, 0, 0, 0],
              [0, 0, 0, 1, 0, 1, 0, 0],
              [0, 0, 0, 0, 0, 0, 1, 0],
              [0, 0, 0, 0, 0, 0, 0, 1]]

# const_hi -> Demand at node j
const_hi = [12, 5, 6, 7, 4, 7, 4, 7]

#####

m = gp.Model('MCLP')

# Add variables
# If warehouse is placed at j
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]
# If demand at i is covered
var_zj = [m.addVar(vtype=GRB.BINARY, name=f'z_{loc}') for loc in locations]

# Uncomment the below lines to force the solution of Sitepu
#m.addConstr(var_xj[0] == 1, 'forced_solution')
#m.addConstr(var_xj[1] == 1, 'forced_solution')
#m.addConstr(var_xj[4] == 1, 'forced_solution')
#m.addConstr(var_xj[5] == 1, 'forced_solution')
#m.addConstr(var_xj[7] == 1, 'forced_solution')

# Constraint
# sum(a_ij * x_j) - z_i >= 0 // sum over j // for every i
i = 0
j = 0
for ai, zi in zip(const_aij, var_zj):
    i += 1
    temp_lin_expr = gp.LinExpr()
    for aij, xj in zip(ai, var_xj):
        temp_lin_expr.add(xj, aij)
    temp_lin_expr.add(zi, -1.0)
    m.addConstr(temp_lin_expr >= 0, f'sum(a_{i}_j * x_j - z_{i}) >= 0')

# Constraint
# sum(X_j) <= P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr <= P, f'sum(var_xj) <= {P}')
```



```
# Objective function
# Maximize sum h_i*z_i
temp_lin_expr = gp.LinExpr()
for hi,zi in zip(const_hi, var_zj):
    temp_lin_expr.add(zi, hi)
m.setObjective(temp_lin_expr, GRB.MAXIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```

## 2.4 Maximum expected covering location model

### 2.4.1 Generic description

In this model the goal is not only to cover the demand node, but also the facility to be available to cover the demands. For example, there might be a fast fire truck near to a demand node that it could cover the demands, but the question is if it is available. Consequently, we are seeking to find a facility that is available to cover demand node when is needed.

As a result, one new factor is added in this model, which is the probability of the facility being busy when is needed. This is called the system-wide average availability probability and symbolized by the letter  $q$ . Moreover, there is the assumption at this model that the probability of being busy a facility at one node  $j_1$  is independent from the probability of being busy a facility at another node  $j_2$ .

### 2.4.2 Mathematical formulation

The mathematical formulation is the following:

Decision variables

$$X_j = \text{Number of facilities to locate at node } j \quad (22)$$

$$Z_{ik} = \begin{cases} 1 & \text{if demand node } i \in I \text{ is covered at least } k \text{ times} \\ 0 & \text{if not} \end{cases} \quad (23)$$

Constants

$$q = \text{system – wide average probability} \quad (24)$$

$$a_{ij} = \begin{cases} 1 & \text{if candidate site } j \in J \text{ can cover demands at node } i \in I \\ 0 & \text{if not} \end{cases} \quad (25)$$

Maximize

$$(1 - q) \sum_{i \in I} h_i \left\{ \sum_{k=1}^P q^{k-1} Z_{ik} \right\} \quad (26)$$

Subject to

$$\sum_{k=1}^P Z_{ik} \leq \sum_{j \in J} a_{ij} X_j \quad \forall i \in I \quad (27)$$

$$\sum_{j \in J} X_j \leq P \quad (28)$$

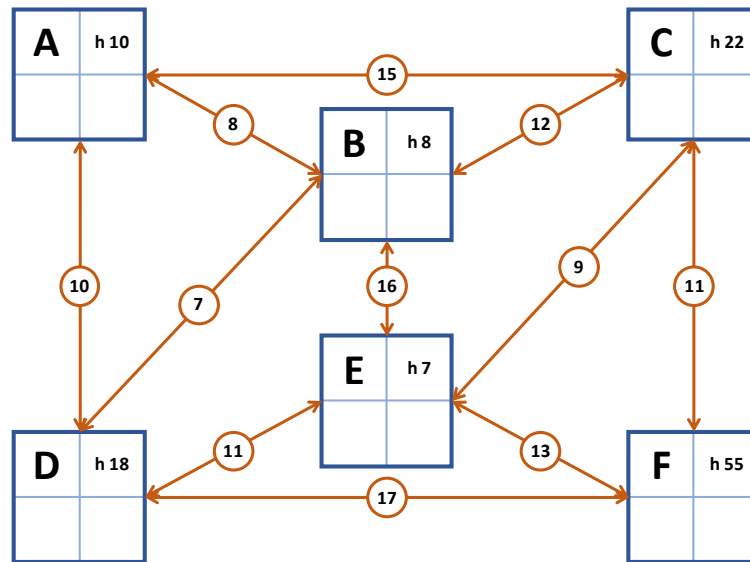
$$X_j \in \{0,1, \dots, P\} \quad \forall j \in J \quad (29)$$

$$Z_{ik} \in \{0,1\} \quad \forall i \in I, k = 1,2, \dots, P \quad (30)$$

The objective function maximizes the expected number of covered demands. The first constraint declares states that node  $i \in I$  can be counted as being covered at least  $k$  times only if at least  $k$  facilities are located at nodes that cover node  $i \in I$ . The second constraint declares that at most  $P$  facilities are to be located. The last constraints shows the what values can take the variables  $X_j$  and  $Z_{ik}$ .

### 2.4.3 Example 1

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013), where  $h$  is the node demand.



**Figure 8 A visualization of the extended Maximum Expected Covering Location Model network**

The mathematical formulation of this model is as follows. First we need to maximize the redundancy probability

$$\begin{aligned} \text{Maximize } Z = & 10(1-q)(Z_{A1} + qZ_{A2} + q^2Z_{A3}) \\ & + 8(1-q)(Z_{B1} + qZ_{B2} + q^2Z_{B3}) \\ & + 22(1-q)(Z_{C1} + qZ_{C2} + q^2Z_{C3}) \\ & + 18(1-q)(Z_{D1} + qZ_{D2} + q^2Z_{D3}) \\ & + 7(1-q)(Z_{E1} + qZ_{E2} + q^2Z_{E3}) \\ & + 55(1-q)(Z_{F1} + qZ_{F2} + q^2Z_{F3}) \end{aligned}$$

then we need to ensure that multiple supply reach/coverage is correctly correlated between the  $X_j$  supply centers and the  $Z_{jk}$  variables that keep track that node  $j$  is covered  $k$  times

$$\begin{aligned} Z_{A1} + Z_{A2} + Z_{A3} & \leq X_A + X_B + X_D \\ Z_{B1} + Z_{B2} + Z_{B3} & \leq X_A + X_B + X_D \\ Z_{C1} + Z_{C2} + Z_{C3} & \leq X_C + X_E \\ Z_{D1} + Z_{D2} + Z_{D3} & \leq X_B + X_D \\ Z_{E1} + Z_{E2} + Z_{E3} & \leq X_C + X_E \\ Z_{F1} + Z_{F2} + Z_{F3} & \leq X_F \end{aligned}$$

then we need to define the amount of supply center to be located

$$X_A + X_B + X_C + X_D + X_E + X_F \leq 3$$

and finally we have to include the integrality constraints

$$X_A, X_B, X_C, X_D, X_E, X_F \in \{0,3\}$$

$$Z_{Ak}, Z_{Bk}, Z_{Ck}, Z_{Dk}, Z_{Ek}, Z_{Fk} \in \{0,1\}, k \in \{1,3\}$$

Executing the monolithic version of the code, we see that results match the solution of the book

$$X_D, X_F, Z_{B1}, Z_{D1}, Z_{F1}, Z_{F2} = 1, Z = 49.6$$

### 2.4.3.1 Code

```
#####
# Maximum expected covering location model                                #
# Chapter 4.7, p.170                                                         #
# Network and Discrete Location Models, 2e, Daskin                         #
#                                                                           #
# Non-parametric solution                                                  #
#                                                                           #
#                                                                           #
# Code by Foteini Bourou, 2022                                             #
#                                                                           #
#####

import gurobipy as gp
```

```
from gurobipy import GRB

def main():
    m = gp.Model('MECLP')
    P=3
    ha=10
    hb=8
    hc=22
    hd=18
    he=7
    hf=55
    q=0.6
    q1 = 1
    q2 = q1 * q
    q3 = q2 * q

    xa=m.addVar(lb=0, ub=P, vtype=GRB.INTEGER, name="xa")
    xb=m.addVar(lb=0, ub=P, vtype=GRB.INTEGER, name="xb")
    xc=m.addVar(lb=0, ub=P, vtype=GRB.INTEGER, name="xc")
    xd=m.addVar(lb=0, ub=P, vtype=GRB.INTEGER, name="xd")
    xe=m.addVar(lb=0, ub=P, vtype=GRB.INTEGER, name="xe")
    xf=m.addVar(lb=0, ub=P, vtype=GRB.INTEGER, name="xf")

    za1=m.addVar(vtype=GRB.BINARY, name="za1")
    za2=m.addVar(vtype=GRB.BINARY, name="za2")
    za3=m.addVar(vtype=GRB.BINARY, name="za3")
    zb1=m.addVar(vtype=GRB.BINARY, name="zb1")
    zb2=m.addVar(vtype=GRB.BINARY, name="zb2")
    zb3=m.addVar(vtype=GRB.BINARY, name="zb3")
    zc1=m.addVar(vtype=GRB.BINARY, name="zc1")
    zc2=m.addVar(vtype=GRB.BINARY, name="zc2")
    zc3=m.addVar(vtype=GRB.BINARY, name="zc3")
    zd1=m.addVar(vtype=GRB.BINARY, name="zd1")
    zd2=m.addVar(vtype=GRB.BINARY, name="zd2")
    zd3=m.addVar(vtype=GRB.BINARY, name="zd3")
    ze1=m.addVar(vtype=GRB.BINARY, name="ze1")
    ze2=m.addVar(vtype=GRB.BINARY, name="ze2")
    ze3=m.addVar(vtype=GRB.BINARY, name="ze3")
    zf1=m.addVar(vtype=GRB.BINARY, name="zf1")
    zf2=m.addVar(vtype=GRB.BINARY, name="zf2")
    zf3=m.addVar(vtype=GRB.BINARY, name="zf3")

    m.addConstr(za1 + za2 + za3 <= xa + xb + xd, "c0")
    m.addConstr(zb1 + zb2 + zb3 <= xa + xb + xd, "c1")
    m.addConstr(zc1 + zc2 + zc3 <= xc + xe, "c2")
    m.addConstr(zd1 + zd2 + zd3 <= xb + xd, "c3")
    m.addConstr(ze1 + ze2 + ze3 <= xc + xe, "c4")
    m.addConstr(zf1 + zf2 + zf3 <= xf, "c5")

    m.addConstr(xa + xb + xc + xd + xe + xf <= P, "c6")

    m.setObjective((1-q)*(
        ha * (q1 * za1 + q2 * za2 + q3 * za3)
        + hb * (q1 * zb1 + q2 * zb2 + q3 * zb3)
        + hc * (q1 * zc1 + q2 * zc2 + q3 * zc3)
```

```

+ hd * (q1 * zd1 + q2 * zd2 + q3 * zd3)
+ he * (q1 * ze1 + q2 * ze2 + q3 * ze3)
+ hf * (q1 * zf1 + q2 * zf2 + q3 * zf3))
, GRB.MAXIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()

```

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). Running the code in monolithic type, we see that results from Gurobi solver and the results from mathematical solution are the same. There is a perfect match.

## 2.4.4 Example 2

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). The problem formulation is the same as the previous paragraph. Executing the parametric version of the code, we see that results match the solution of the book.

### 2.4.4.1 Code

```

#####
# Maximum expected covering location model                                     #
# Chapter 4.7, p.170                                                         #
# Network and Discrete Location Models, 2e, Daskin                         #
#                                                                           #
# Parametric solution                                                         #
#                                                                           #
#                                                                           #
# Code by Foteini Bourou, 2022                                              #
#                                                                           #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set Locations and demand nodes
    locations = ['a', 'b', 'c', 'd', 'e', 'f']

```

```
# Desired Locations
P = 3

# System wide probability that a facility is busy
q = 0.6

# const_aij -> If candidate site j can cover demand at node i
const_aij = [[1, 1, 0, 1, 0, 0],
             [1, 1, 0, 1, 0, 0],
             [0, 0, 1, 0, 1, 0],
             [0, 1, 0, 1, 0, 0],
             [0, 0, 1, 0, 1, 0],
             [0, 0, 0, 0, 0, 1]]

# const_hi -> Demand at node j
const_hi = [10, 8, 22, 18, 7, 55]

#####

m = gp.Model('MECLP')

# Add variables
var_xj = [m.addVar(lb = 0, ub = P, vtype=GRB.INTEGER, name=f'x_{loc}') for loc
in locations]
var_zik = []
for i in locations:
    var_zi = [] # temporary List
    for k in range(1, P + 1): # k = if demands at node i are covered at least k
times
        var_zi.append(m.addVar(vtype=GRB.BINARY, name=f'z_{i}_{k}'))
    var_zik.append(var_zi)

# Constraint
# sum(a_ij * x_j) - sum(zik) >= 0 // sum over j,k // for every i
i = 0
k = 0
for ai, zi in zip(const_aij, var_zik):
    i += 1
    k += 1
    temp_lin_expr = gp.LinExpr()
    for aij, xj in zip(ai, var_xj):
        temp_lin_expr.add(xj, aij)
    for zik in zi:
        temp_lin_expr.add(zik, -1.0)
    m.addConstr(temp_lin_expr >= 0, f'sum(a_{i}_j * x_j - sum_z_{i}_k) >= 0')

# Constraint
# sum(X_j) <= P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr <= P, f'sum_var_x_j <= {P}')
```

# Objective function

```
temp_lin_expr = gp.LinExpr()
```

```
for h, zi in zip(const_hi, var_zik):
    factor = (1 - q) * h
    temp_lin_expr_2 = gp.LinExpr()
    for k in range(1, P + 1):
        temp_lin_expr_2.add(zi[k-1] , q**(k-1))
    temp_lin_expr.add(temp_lin_expr_2, factor)
m.setObjective(temp_lin_expr, GRB.MAXIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). Running the code in parametric type, we see that results from Gurobi solver and the results from mathematical solution are the same. There is a perfect match.



## 2.5 Vertex P-center location model

### 2.5.1 Generic description

The goal of this problem is to minimize the coverage distance by locating  $P$  facilities which will cover some of the demands. In this model, it is not necessary that all demand nodes are covered by the facilities that will be located at the nodes of the network. In addition, every demand node can have a weight (demand) which in turn can reflect time per unit or even the cost per unit distance. This comes in contrast to the covering models, where demand-weighting was neglected.

In order to solve this problem, there are some requirements. First of all, the facilities can be located only on the nodes of the network and their capacity is unlimited. Then, the number of facilities that can be located is  $P$ . Finally, the demand at the nodes can have a distinct weighting factor (demand) (Hakimi, 1965).

Furthermore, vertex-p center problem is commonly used in supply chain processes such as in finding the appropriate firehouses (Plane & Hendrick, 1977), hospitals (Chu & Chu, 2000) or mailboxes (Labbé & Laporte, 1986). Vertex-p center model was also used in retail site location. One such model, by (Church, Niblett, & Gerrard, Modeling the Potential for Critical Habitat, 2015), is intended for retail site location. It trades off the number of facilities with the separation distance between facilities in order to maximize retail profits, while adding some amount of over-coverage in order to prevent competition from other franchises to profitably enter a market.

### 2.5.2 Mathematical formulation

The mathematical formulation is the following:

Decision variables

$$X_j = \begin{cases} 1 & \text{if we locate a facility at candidate site } j \in J \\ 0 & \text{if not} \end{cases} \quad (31)$$

$$Y_{ij} = \text{fraction of demand at node } i \in I \text{ that is served by a facility at node } j \in J \quad (32)$$

$$W = \text{maximum distance between a demand node and the nearest facility} \quad (33)$$

Constants

$$d_{ij} = \text{distance from demand node } i \in I \text{ to candidate facility site } j \in J \quad (34)$$

$$P = \text{number of facilities to locate} \quad (35)$$

$$h_i = \text{demand at node } i \in I \quad (36)$$

Minimize

$$W \quad (37)$$

Subject to

$$\sum_{j \in J} Y_{ij} = 1 \quad \forall i \in I \quad (38)$$

$$\sum_{j \in J} X_j = P \quad (39)$$

$$Y_{ij} \leq X_j \quad \forall i \in I; j \in J \quad (40)$$

$$W \geq \sum_{j \in J} d_{ij} Y_{ij} \quad \forall i \in I \quad (41)$$

$$X_j \in \{0, 1, \dots, P\} \quad \forall j \in J \quad (42)$$

$$Y_{ij} \geq 0 \quad i \in I; \forall j \in J \quad (43)$$

Optionally this constraint

$$W \geq \sum_{j \in J} d_{ij} Y_{ij} \quad \forall i \in I \quad (41)$$

can be replaced by this

$$W \geq \sum_{j \in J} h_i d_{ij} Y_{ij} \quad \forall i \in I \quad (44)$$

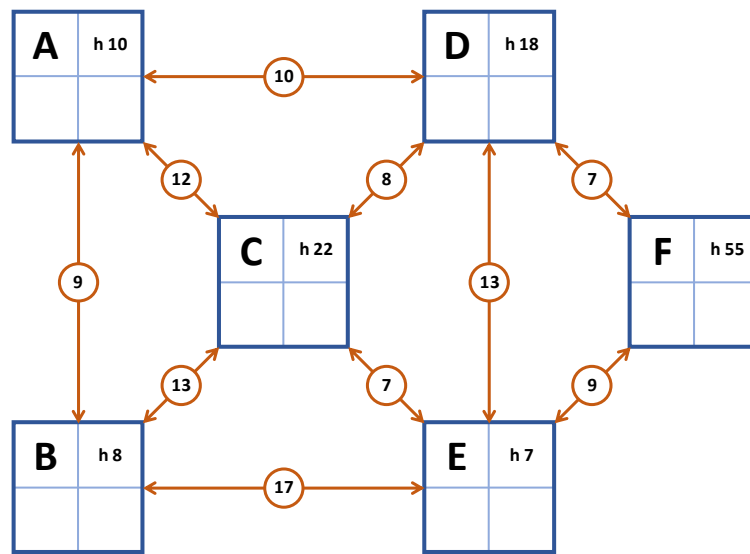
if we want to consider the demand-weighted distance.

The objective function minimizes the maximum distance between a demand node and the closest facility to the node. The first constraint states that all of the demand at node  $i \in I$  must be assigned to a facility at some node  $j \in J$  for all nodes  $i \in I$ . The second constraint stipulates that  $P$  facilities be located. The third constraint declare that demands at node  $i \in I$  cannot be assigned to a facility at node  $j \in J$  unless a facility is located at node  $j \in J$ . The next constraint

state that the maximum distance between a demand node and the nearest facility to the node ( $W$ ) must be greater than the distance between any demand node  $i \in I$  and the facility  $j \in J$  to which it is assigned. The last constraints show at which interval they belong to. Applications of the p-center model appeared on placing ambulances, police stations and fire stations.

### 2.5.3 Example 1

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013), where  $h$  is the node demand.



**Figure 9 A visualization of the Vertex P-Center Model network**

The values within rectangles are the demands of each node that can optionally be taken into account and distances between nodes can be seen in the respective links. The distance matrix then becomes

$$d_{ij} = \begin{bmatrix} 0 & 9 & 12 & 10 & 19 & 17 \\ 9 & 0 & 13 & 21 & 17 & 26 \\ 12 & 13 & 0 & 8 & 7 & 15 \\ 10 & 21 & 8 & 0 & 13 & 7 \\ 19 & 17 & 7 & 13 & 0 & 9 \\ 17 & 26 & 15 & 7 & 9 & 0 \end{bmatrix}$$

with  $P = [1,5]$  facilities to locate. Executing the code, we see that results match the solution of the book, except for  $P = 4$ .

$$X_A, X_B, X_C, X_D, Y_{AA}, Y_{BB}, Y_{CC}, Y_{DD}, Y_{EC}, Y_{FD} = 1, W = 7$$

Further investigation shows that the book and Gurobi's solution for  $P = 4$  have the same objective value.

$$X_A, X_B, X_D, X_E, Y_{AA}, Y_{BB}, Y_{CE}, Y_{DD}, Y_{EE}, Y_{FD} = 1, W = 7$$

### 2.5.3.1 Code

```
#####
# Vertex P-center formulation                                     #
# Chapter 5.2, p.199                                           #
# Network and Discrete Location Models, 2e, Daskin           #
#                                                             #
# Parametric solution                                         #
#                                                             #
#                                                             #
# Code by Foteini Bourou, 2022                                #
#                                                             #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    locations = ['a', 'b', 'c', 'd', 'e', 'f']
    P = 4 # desired Locations
    const_dij = [[ 0, 9, 12, 10, 19, 17],
                  [ 9, 0, 13, 21, 17, 26],
                  [12, 13, 0, 8, 7, 15],
                  [10, 21, 8, 0, 13, 7],
                  [19, 17, 7, 13, 0, 9],
                  [17, 26, 15, 7, 9, 0]]
    #const_hi = [10, 8, 22, 18, 7, 55]
    const_hi = [1, 1, 1, 1, 1, 1]

    #####

    m=gp.Model('p-center')

    # Variables
    var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]

    # Variables
    var_yij = []
    for loci in locations:
        var_yi = [] # temporary List
        for locj in locations:
            var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
        var_yij.append(var_yi)
    var_w = m.addVar(vtype=GRB.CONTINUOUS, name='w')

    # Uncomment the below lines to force the solution of Daskin for P=4
    #m.addConstr(var_xj[0] == 1, 'forced_solution')
```

```
#m.addConstr(var_xj[1] == 1, 'forced_solution')
#m.addConstr(var_xj[3] == 1, 'forced_solution')
#m.addConstr(var_xj[4] == 1, 'forced_solution')

# Constraint
# Sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j = 1')

# Constraint
# Sum(X_j) = P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr == P, f'sum_var_xj = {P}')
```

```
# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij,xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')
```

```
# Constraint
# W - sum(h_i * d_ij * Y_ij) >= 0 // sum over j // for every i
i = 0
for hi,yi,di in zip(const_hi, var_yij, const_dij):
    i += 1
    temp_lin_expr = gp.LinExpr()
    temp_lin_expr.add(var_w, 1.0)
    for yij, dij in zip(yi,di):
        temp_lin_expr.add(yij, -hi*dij)
    m.addConstr(temp_lin_expr >= 0, f'W - h_{i} * sum(d_{i} _j* Y_{i}_j) >= 0')
```

```
# Objective function
m.setObjective(var_w, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)
```

```
if __name__ == '__main__':
    main()
```

### 2.5.4 Example 2

This is an example from (Sitepu, et al., 2019). The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 38 & 17 & 37 & 19 & 33 & 29 & 43 \\ 40 & 0 & 26 & 49 & 29 & 45 & 22 & 48 \\ 16 & 13 & 0 & 40 & 21 & 36 & 36 & 27 \\ 41 & 49 & 40 & 0 & 40 & 12 & 68 & 67 \\ 20 & 20 & 27 & 40 & 0 & 50 & 33 & 25 \\ 37 & 44 & 35 & 12 & 52 & 0 & 64 & 62 \\ 30 & 24 & 40 & 62 & 34 & 58 & 0 & 54 \\ 41 & 38 & 30 & 65 & 24 & 60 & 53 & 0 \end{bmatrix}$$

Running the code, we see that the results from this code do not match the reported optimal solution with  $W = 8512$ .

$$X_1, X_2, X_4, X_5, X_7, X_8 = 1$$

$$Y_{11}, Y_{21}, Y_{32}, Y_{44}, Y_{64}, Y_{88} = 1, Y_{54} = 0.61, Y_{57} = 0.39, Y_{71} = 0.43, Y_{77} = 0.57$$

$$W = 13$$

We also notice that the reported optimal objective function result is abnormally high. Further investigation shows that the objective function reported in (Sitepu, et al., 2019) can be also stated as minimize  $W \geq \sum_{j \in J} d_{ij} Y_{ij} \forall i \in I$ . The sum of all  $d_{ij}$  factors is equal to 2115. If this model were to give an optimal objective function value equal to 8512, then  $Y_{ij} \geq 1$  must be true at least for some  $i, j$ , which of course is not possible as  $Y_{ij} = [0,1]$ . As a reminder, the meaning of  $W$  is the maximum distance between supply and demand nodes.

#### 2.5.4.1 Code

```
#####
# Vertex P-center formulation                                     #
# p.6                                                            #
# Set covering models in optimizing the emergency unit location  #
# of health facility in Palembang, Sitepu et al.                #
#                                                                #
#                                                                #
#                                                                #
# Code by Foteini Bourou, 2022                                   #
#                                                                #
#####
```

```

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set Locations and demand nodes
    locations = ['1 - Ilir Timur II',
                '2 - Kalidoni',
                '3 - Kemuning',
                '4 - Plaju',
                '5 - Sako',
                '6 - Seberang Ulu II',
                '7 - Sematan Borang',
                '8 - Sukarami']

    # const_dij -> Distance between demand node i and candidate j
    const_dij = [[ 0, 38, 17, 37, 19, 33, 29, 43],
                 [40, 0, 26, 49, 29, 45, 22, 48],
                 [16, 13, 0, 40, 21, 36, 36, 27],
                 [41, 49, 40, 0, 40, 12, 68, 67],
                 [20, 20, 27, 40, 0, 50, 33, 25],
                 [37, 44, 35, 12, 52, 0, 64, 62],
                 [30, 24, 40, 62, 34, 58, 0, 54],
                 [41, 38, 30, 65, 24, 60, 53, 0]]

    # Desired Locations
    P = 6

    # const_hi -> Demand at node j
    const_hi = [1, 1, 1, 1, 1, 1, 1, 1]
    #const_hi = [12, 5, 6, 7, 4, 7, 4, 7]

    #####

    m=gp.Model('p-center')

    # Variables
    var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]

    # Variables
    var_yij = []
    for loci in locations:
        var_yi = [] # temporary list
        for locj in locations:
            var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
        var_yij.append(var_yi)
    var_w = m.addVar(vtype=GRB.CONTINUOUS, name='w')

    # Constraint
    # Sum(Y_ij) = 1 // sum over j // for every i
    i = 0
    for yi in var_yij:
        i += 1
        temp_lin_expr = gp.LinExpr()
        for yij in yi:

```

```

        temp_lin_expr.add(yij, 1.0)
        m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j = 1')

# Constraint
# Sum(X_j) = P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr == P, f'sum_var_xj = {P}')
```

```

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij,xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')
```

```

# Constraint
# W - sum(h_i * d_ij * Y_ij) >= 0 // sum over j // for every i
i = 0
for hi,yi,di in zip(const_hi, var_yij, const_dij):
    i += 1
    temp_lin_expr = gp.LinExpr()
    temp_lin_expr.add(var_w, 1.0)
    for yij, dij in zip(yi,di):
        temp_lin_expr.add(yij, -hi*dij)
    m.addConstr(temp_lin_expr >= 0, f'W - h_{i} * sum(d_{i}_j* Y_{i}_j) >= 0')
```

```

# Objective function
m.setObjective(var_w, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```



### 2.5.5 Example 3

This is an example from the Thesis of (Chatzigiannis, 2013). The goal of the example is to place one or two supply centers in some nodes (Attiki, Theva, Lamia, Mesologgi, Xalkida). The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 33.36 & 51.36 & 58.56 & 18.72 \\ 33.36 & 0 & 26.4 & 47.28 & 24.96 \\ 51.36 & 26.4 & 0 & 50.16 & 39.84 \\ 58.56 & 47.28 & 50.16 & 0 & 74.16 \\ 18.72 & 24.96 & 39.84 & 74.16 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [72 \quad 2 \quad 11 \quad 25 \quad 9]$$

Running the code, we see that the results from this code do not match the reported optimal solution.

$$X_{attiki}, X_{mesologgi} = 1$$

$$Y_{attiki\_attiki}, Y_{theva\_attiki}, Y_{lamia\_mesologgi} = 1$$

$$Y_{mesologgi\_attiki} = 0.377, Y_{mesologgi\_mesologgi} = 0.623$$

$$Y_{xalkida\_attiki} = 0.231, Y_{xalkida\_mesologgi} = 0.769$$

$$W = 551.76$$

We also notice that the reported optimal objective function result is abnormally low. Further investigation of (Chatzigiannis, 2013) shows that one of the constraints is erroneously modeled as  $\sum_{i \in I} \sum_{j \in J} Y_{ij} = 1$  instead of  $\sum_{j \in J} Y_{ij} = 1 \forall i \in I$ . This is the reason for this deviation.

#### 2.5.5.1 Code

```
#####
# Vertex P-center formulation                                     #
# Chapter 6.4.4, p.90                                           #
# Location models within supply chains: analysis and application #
# Ioannis Chatzigiannis, 2013                                   #
#                                                                 #
#                                                                 #
#                                                                 #
# Code by Foteini Bourou, 2022                                   #
#                                                                 #
#####
```

```
import gurobipy as gp
from gurobipy import GRB

def main():
    # Set Locations and demand nodes
    locations = ['attiki', 'theva', 'lamia', 'mesologgi', 'xalkida']

    # const_dij -> Distance between demand node i and candidate j
    const_dij = [[0, 33.36, 51.36, 58.56, 18.72],
                  [33.36, 0, 26.4, 47.28, 24.96],
                  [51.36, 26.4, 0, 50.16, 39.84],
                  [58.56, 47.28, 50.16, 0, 74.16],
                  [18.72, 24.96, 39.84, 74.16, 0]]

    # Desired locations
    P = 2

    # const_hi -> Demand at node j
    const_hi = [72, 2, 11, 25, 9]

    #####

    m=gp.Model('p-center')

    # Variables
    var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]

    # Variables
    var_yij = []
    for loci in locations:
        var_yi = [] # temporary List
        for locj in locations:
            var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
        var_yij.append(var_yi)
    var_w = m.addVar(vtype=GRB.CONTINUOUS, name='w')

    # Constraint
    # Sum(Y_ij) = 1 // sum over j // for every i
    i = 0
    for yi in var_yij:
        i += 1
        temp_lin_expr = gp.LinExpr()
        for yij in yi:
            temp_lin_expr.add(yij, 1.0)
        m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j = 1')

    # Constraint
    # Sum(X_j) = P // sum over j
    temp_lin_expr = gp.LinExpr()
    for xj in var_xj:
        temp_lin_expr.add(xj, 1.0)
    m.addConstr(temp_lin_expr == P, f'sum_var_xj = {P}')

    # Constraint
    # Y_ij - X_j <= 0 // for every i,j
```

```

i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij,xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Constraint
# W - sum(h_i * d_ij * Y_ij) >= 0 // sum over j // for every i
i = 0
for hi,yi,di in zip(const_hi, var_yij, const_dij):
    i += 1
    temp_lin_expr = gp.LinExpr()
    temp_lin_expr.add(var_w, 1.0)
    for yij, dij in zip(yi,di):
        temp_lin_expr.add(yij, -hi*dij)
    m.addConstr(temp_lin_expr >= 0, f'W - h_{i} * sum(d_{i} _j* Y_{i}_j) >= 0')

# Objective function
m.setObjective(var_w, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()

```

### 2.5.6 Example 4

This is an example from the Thesis of (Paraskevopoulou, 2015). The goal of the example is to place one or two supply centers in some nodes (Xalkida, Athina, Theva, Amfissa, Mesologgi, Lamia). The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 18.72 & 24.96 & 63.36 & 74.16 & 25.44 \\ 18.72 & 0 & 33.36 & 62.16 & 58.56 & 51.36 \\ 24.96 & 33.36 & 0 & 45.84 & 47.28 & 26.4 \\ 63.36 & 62.16 & 45.84 & 0 & 40.56 & 24.48 \\ 74.16 & 58.56 & 47.28 & 40.56 & 0 & 50.16 \\ 39.84 & 51.36 & 26.4 & 24.48 & 50.16 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [34 \quad 1140 \quad 3 \quad 4 \quad 16 \quad 17]$$

Running the code, we see that the results from this code do not match the reported optimal solution.

$$X_{athina}, X_{amfissa} = 1$$

$$Y_{xalkida\_athina}, Y_{athina\_athina}, Y_{theva\_athina}, Y_{amfissa\_athina}, Y_{mesologgi\_amfissa} = 1$$

$$Y_{lamia\_athina} = 0.491, Y_{lamia\_amfissa} = 0.509$$

$$W = 648.96$$

We also notice that the reported optimal objective function result is abnormally low. Further investigation of (Paraskevopoulou, 2015) shows that one of the constraints is erroneously modeled as  $\sum_{i \in I} \sum_{j \in J} Y_{ij} = 1$  instead of  $\sum_{j \in J} Y_{ij} = 1 \forall i \in I$ . This is the reason for this deviation.

### 2.5.6.1 Code

```
#####
# Vertex P-center formulation                                     #
# Chapter 8.5.3, p.76                                           #
# Linear Programming Models for Optimal Location Problem       #
# Paraskevopoulou 2015                                         #
#                                                                 #
#                                                                 #
#                                                                 #
# Code by Foteini Bourou, 2022                                  #
#                                                                 #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    locations = ['xalkida', 'athina', 'theva', 'amfissa', 'mesologgi', 'lamia']
    P = 2 # desired Locations
    const_dij = [[0,18.72,24.96,63.36,74.16,25.44],
                 [18.72,0,33.36,62.16,58.56,51.36],
                 [24.96,33.36,0,45.84,47.28,26.4],
                 [63.36,62.16,45.84,0,40.56,24.48],
                 [74.16,58.56,47.28,40.56,0,50.16],
                 [39.84,51.36,26.4,24.48,50.16,0]]
    const_hi = [34,1140,3,4,16,17]

    #####

    m=gp.Model('p-center')
```

```
# Variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]

# Variables
var_yij = []
for loci in locations:
    var_yi = [] # temporary list
    for locj in locations:
        var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
    var_yij.append(var_yi)
var_w = m.addVar(vtype=GRB.CONTINUOUS, name='w')

# Constraint
# Sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j = 1')

# Constraint
# Sum(X_j) = P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr == P, f'sum_var_xj = {P}')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij,xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Constraint
# W - sum(h_i * d_ij * Y_ij) >= 0 // sum over j // for every i
i = 0
for hi,yi,di in zip(const_hi, var_yij, const_dij):
    i += 1
    temp_lin_expr = gp.LinExpr()
    temp_lin_expr.add(var_w, 1.0)
    for yij, dij in zip(yi,di):
        temp_lin_expr.add(yij, -hi*dij)
    m.addConstr(temp_lin_expr >= 0, f'W - h_{i} * sum(d_{i}_j* Y_{i}_j) >= 0')

# Objective function
m.setObjective(var_w, GRB.MINIMIZE)
```

```
m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))

print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```

## 2.6 P-median location model

### 2.6.1 Generic description

The goal of this model is to find the location of  $P$  facilities in such a way that it will be minimized the total demand-weighted distance between each demand node and the nearest facility (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). This minimizes the cost of transportation from supply to demand nodes.

The P-median models are very helpful tools in supply chain, especially in health sector. (McAleer & Naqvi, 1994) used a P-median model to relocate ambulances station in Belfast in Ireland. The purpose was to relocate ambulance stations in order to provide emergency ambulance service to people when they needed it. (McAleer & Naqvi, 1994).

However, as the p-median model is an umbrella method the UFL (2.7 Uncapacitated fixed charge facility location problems) and the CFL problems (2.8 Capacitated fixed charge facility location problems) most of the research surrounding this topic involves mostly methods to solve it (Laporte, Nickel, & da Gama, 2015).

### 2.6.2 Mathematical formulation

The mathematical formulation is the following:

Decision variables

$$X_j = \begin{cases} 1 & \text{if we locate a facility at candidate site } j \in J \\ 0 & \text{if not} \end{cases} \quad (45)$$

$$Y_{ij} = \begin{cases} 1 & \text{if demands at node } i \in I \text{ are served by a facility at node } j \in J \\ 0 & \text{if not} \end{cases} \quad (46)$$

Constants

$$d_{ij} = \text{distance from demand node } i \in I \text{ to candidate facility site } j \in J \quad (47)$$

$$h_i = \text{demand at node } i \in I \quad (48)$$

$$P = \text{number of facilities to locate} \quad (49)$$

Minimize

$$Z = \sum_{i \in I} \sum_{j \in J} h_i d_{ij} Y_{ij} \quad (50)$$

Subject to

$$\sum_{j \in J} Y_{ij} = 1 \quad \forall i \in I \quad (51)$$

$$\sum_{j \in J} X_j = P \quad (52)$$

$$Y_{ij} - X_j \leq 0 \quad \forall i \in I; j \in J \quad (53)$$

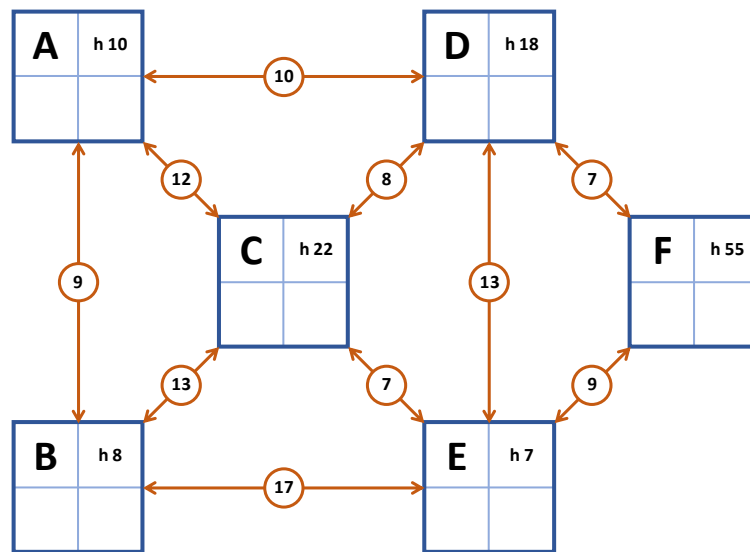
$$X_j \in \{0,1\} \quad \forall j \in J \quad (54)$$

$$Y_{ij} \geq 0 \quad i \in I; \forall j \in J \quad (55)$$

The objective function minimizes the total demand-weighted distance between each demand node  $i \in I$  to be assigned to exactly one facility  $j \in J$ . The last constraints show at which interval they belong to.

### 2.6.3 Example 1

This is an example from the book of (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013), where  $h$  is the node demand.



**Figure 10 A visualization of the P-median Location Model network**



The goal is to locate  $P$  supply locations. The distance matrix becomes

$$d_{ij} = \begin{bmatrix} 0 & 9 & 12 & 10 & 19 & 17 \\ 9 & 0 & 13 & 21 & 17 & 26 \\ 12 & 13 & 0 & 8 & 7 & 15 \\ 10 & 21 & 8 & 0 & 13 & 7 \\ 19 & 17 & 7 & 13 & 0 & 9 \\ 17 & 26 & 15 & 7 & 9 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [10 \quad 8 \quad 22 \quad 18 \quad 7 \quad 55]$$

The final solution for  $P = 4$  is

$$X_A, X_C, X_D, X_F, Y_{AA}, Y_{BA}, Y_{CC}, Y_{DD}, Y_{EC}, Y_{FF} = 1, Z = 121$$

### 2.6.3.1 Code

```
#####
# Maximum covering model with median formulation, cost minimization #
# Chapter 6.2, p.238 #
# Network and Discrete Location Models, 2e, Daskin #
# # #
# Parametric solution #
# # #
# # #
# Code by Foteini Bourou, 2022 #
# # #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set Locations and demand nodes
    locations = ['a', 'b', 'c', 'd', 'e', 'f']

    # const_dij -> Distance between demand node i and candidate j
    const_dij = [[ 0, 9, 12, 10, 19, 17],
                 [ 9, 0, 13, 21, 17, 26],
                 [12, 13, 0, 8, 7, 15],
                 [10, 21, 8, 0, 13, 7],
                 [19, 17, 7, 13, 0, 9],
                 [17, 26, 15, 7, 9, 0]]

    # Desired Locations
    P = 4

    # const_hi -> Demand at node j
    const_hi = [10, 8, 22, 18, 7, 55]
```

```
#####

m=gp.Model('p-median')

# Variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]

# Variables
var_yij = []
for loci in locations:
    var_yi = [] # temporary list
    for locj in locations:
        var_yi.append(m.addVar(vtype=GRB.BINARY, name=f'y_{loci}_{locj}'))
    var_yij.append(var_yi)

# Constraint
# sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum(y_{i}) = 1')

# Constraint
# sum(X_j) = P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr == P, f'sum(var_xj) = {P}')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Objective
# sum (hi*dij*Yij) // for every j and i
temp_lin_expr = gp.LinExpr()
for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, hi*dij)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
```

```
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))
    print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```

## 2.6.4 Example 2

This is an example based on dataset pmed1.txt from (Beasley, 1990). These models have the purpose of being verifications standards whose solution is the one that any code that solves such problems can be evaluated against the reference solution. Executing the code, we see that results match the reference optimal solution.

### 2.6.4.1 Code

```
#####
# Maximum covering model with median formulation, cost minimization #
# pmed1.txt                                                         #
# OR-Library: Distributing Test Problems by Electronic Mail        #
# J. E. Beasley                                                     #
#                                                                     #
#                                                                     #
# Parametric solution                                              #
#                                                                     #
# Code by Foteini Bourou, 2022                                     #
#                                                                     #
#####

import gurobipy as gp
from gurobipy import GRB

import numpy

def main():
    path_to_input_file = 'beasley_pmed1.txt'
    input_file = open(path_to_input_file, 'r')
    lines = input_file.read().split('\n')
    input_file.close()

    locations_num, paths_num, P = [int(x) for x in lines[0].split()]
    ii, jj, vals = [], [], []
    for line in lines[1:]:
        tokens = [int(x) for x in line.split()]
        if tokens:
            ii.append(tokens[0]-1)
            jj.append(tokens[1]-1)
            vals.append(tokens[2])

    inf = 1e20 # Infinite value
```

```

const_dij = [[inf] * locations_num for i in range(locations_num)]
for i in range(locations_num):
    const_dij[i][i] = 0
for i,j,val in zip(ii,jj,vals):
    const_dij[i][j] = val
    const_dij[j][i] = val

# Floyd - Warshall algorithm
const_dij = list(map(lambda i: list(map(lambda j: j, i)), const_dij))
for k in range(locations_num):
    for i in range(locations_num):
        for j in range(locations_num):
            const_dij[i][j] = min(const_dij[i][j], const_dij[i][k] +
const_dij[k][j])

locations = [str(i) for i in range(1, locations_num+1)]
const_hi = [1 for i in range(1, locations_num+1)]

#####

m=gp.Model('p-median')

# Variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]

# Variables
var_yij = []
for loci in locations:
    var_yi = [] # temporary list
    for locj in locations:
        var_yi.append(m.addVar(vtype=GRB.BINARY, name=f'y_{loci}_{locj}'))
    var_yij.append(var_yi)

# Constraint
# sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum(y_{i}) = 1')

# Constraint
# sum(X_j) = P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr == P, f'sum(var_xj) = {P}')

# Constraint

# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:

```

```

i += 1
for yij, xj in zip(yi, var_xj):
    j += 1
    temp_lin_expr = gp.LinExpr()
    temp_lin_expr.add(yij, 1.0)
    temp_lin_expr.add(xj, -1.0)
    m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Objective
# sum (hi*dij*yij) // for every j and i
temp_lin_expr = gp.LinExpr()
for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, hi*dij)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))
print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()

```

### 2.6.5 Example 3

This is an example from (Sitepu, et al., 2019). The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 38 & 17 & 37 & 19 & 33 & 29 & 43 \\ 40 & 0 & 26 & 49 & 29 & 45 & 22 & 48 \\ 16 & 13 & 0 & 40 & 21 & 36 & 36 & 27 \\ 41 & 49 & 40 & 0 & 40 & 12 & 68 & 67 \\ 20 & 20 & 27 & 40 & 0 & 50 & 33 & 25 \\ 37 & 44 & 35 & 12 & 52 & 0 & 64 & 62 \\ 30 & 24 & 40 & 62 & 34 & 58 & 0 & 54 \\ 41 & 38 & 30 & 65 & 24 & 60 & 53 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [12 \ 5 \ 6 \ 7 \ 4 \ 7 \ 4 \ 7]$$

Running the code, we see that the results from this code do not match the reported optimal solution with  $Z = 8512$ .

$$X_1, X_2, X_4, X_6, X_7, X_8 = 1$$

$$Y_{11}, Y_{22}, Y_{32}, Y_{44}, Y_{52}, Y_{66}, Y_{77}, Y_{88} = 1$$

$$Z = 158$$

We also notice that the reported optimal objective function result is abnormally high. Further investigation shows that the objective function reported in (Sitepu, et al., 2019) can be also stated as minimize  $W \geq \sum_{j \in J} d_{ij} Y_{ij} \forall i \in I$ . The sum of all  $d_{ij}$  factors is equal to 2115. If this model were to give an optimal objective function value equal to 8512, then  $Y_{ij} \geq 1$  must be true at least for some  $i, j$ , which of course is not possible as  $Y_{ij} = [0,1]$ . As a reminder, the meaning of  $Z$  is the maximum weighted distance between supply and demand nodes.

### 2.6.5.1 Code

```
#####
# Maximum covering model with median formulation, cost minimization #
# p.6 #
# Set covering models in optimizing the emergency unit location #
# of health facility in Palembang, Sitepu et al. #
# #
# #
# #
# Code by Foteini Bourou, 2022 #
# #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set locations and demand nodes
    locations = ['1 - Ilir Timur II',
                '2 - Kalidoni',
                '3 - Kemuning',
                '4 - Plaju',
                '5 - Sako',
                '6 - Seberang Ulu II',
                '7 - Sematan Borang',
                '8 - Sukarami']

    # const_dij -> Distance between demand node i and candidate j
    const_dij = [[ 0, 38, 17, 37, 19, 33, 29, 43],
                 [40, 0, 26, 49, 29, 45, 22, 48],
                 [16, 13, 0, 40, 21, 36, 36, 27],
                 [41, 49, 40, 0, 40, 12, 68, 67],
                 [20, 20, 27, 40, 0, 50, 33, 25],
                 [37, 44, 35, 12, 52, 0, 64, 62],
                 [30, 24, 40, 62, 34, 58, 0, 54],
                 [41, 38, 30, 65, 24, 60, 53, 0]]

    # Desired locations
    P = 6

    # const_hi -> Demand at node j
    const_hi = [12, 5, 6, 7, 4, 7, 4, 7]
    #####
```

```

m=gp.Model('p-median')

# Variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]

# Variables
var_yij = []
for loci in locations:
    var_yi = [] # temporary list
    for locj in locations:
        var_yi.append(m.addVar(vtype=GRB.BINARY, name=f'y_{loci}_{locj}'))
    var_yij.append(var_yi)

# Constraint
# sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum(y_{i}) = 1')

# Constraint
# sum(X_j) = P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr == P, f'sum(var_xj) = {P}')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Objective
# sum (hi*dij*Yij) // for every j and i
temp_lin_expr = gp.LinExpr()
for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, hi*dij)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():

```

```

if v.X:
    print('%s %g' % (v.VarName, v.X))
print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()

```

### 2.6.6 Example 4

This is an example from the Thesis of (Chatzigiannis, 2013). The goal of the example is to place one or two supply centers in some nodes (Attiki, Theva, Lamia, Mesologgi, Xalkida). The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 33.36 & 51.36 & 58.56 & 18.72 \\ 33.36 & 0 & 26.4 & 47.28 & 24.96 \\ 51.36 & 26.4 & 0 & 50.16 & 39.84 \\ 58.56 & 47.28 & 50.16 & 0 & 74.16 \\ 18.72 & 24.96 & 39.84 & 74.16 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [72 \quad 2 \quad 11 \quad 25 \quad 9]$$

Running the code, we see that the results from this code do not match the reported optimal solution.

$$X_{attiki}, X_{mesologgi} = 1$$

$$Y_{attiki\_attiki}, Y_{theva\_attiki}, Y_{lamia\_mesologgi}, Y_{mesologgi\_mesologgi}, Y_{xalkida\_attiki} = 1$$

$$Z = 786.96$$

We also notice that the reported optimal objective function result  $Z = 50$  is abnormally low. Further investigation of (Chatzigiannis, 2013) shows that one of the constraints is erroneously modeled as  $\sum_{i \in I} \sum_{j \in J} Y_{ij} = 1$  instead of  $\sum_{j \in J} Y_{ij} = 1 \forall i \in I$ . This is the reason for this deviation.

#### 2.6.6.1 Code

```

#####
# Maximum covering model with median formulation, cost minimization #
# Chapter 6.4.1, p.85 #
# Location models within supply chains: analysis and application #
# Ioannis Chatzigiannis, 2013 #
# #
# #
# #

```



```
# Code by Foteini Bourou, 2022 #
# #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set locations and demand nodes
    locations = ['attiki', 'theva', 'lamia', 'mesologgi', 'xalkida']

    # const_dij -> Distance between demand node i and candidate j
    const_dij = [[0, 33.36, 51.36, 58.56, 18.72],
                  [33.36, 0, 26.4, 47.28, 24.96],
                  [51.36, 26.4, 0, 50.16, 39.84],
                  [58.56, 47.28, 50.16, 0, 74.16],
                  [18.72, 24.96, 39.84, 74.16, 0]]

    # Desired locations
    P = 2

    # const_hi -> Demand at node j
    const_hi = [72, 2, 11, 25, 9]

    #####

    m=gp.Model('p-median')

    # Variables
    var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]

    # Variables
    var_yij = []
    for loci in locations:
        var_yi = [] # temporary list
        for locj in locations:
            var_yi.append(m.addVar(vtype=GRB.BINARY, name=f'y_{loci}_{locj}'))
        var_yij.append(var_yi)

    # Constraint
    # sum(Y_ij) = 1 // sum over j // for every i
    i = 0
    for yi in var_yij:
        i += 1
        temp_lin_expr = gp.LinExpr()
        for yij in yi:
            temp_lin_expr.add(yij, 1.0)
        m.addConstr(temp_lin_expr == 1, f'sum(y_{i}) = 1')

    # Constraint
    # sum(X_j) = P // sum over j
    temp_lin_expr = gp.LinExpr()
    for xj in var_xj:
        temp_lin_expr.add(xj, 1.0)
    m.addConstr(temp_lin_expr == P, f'sum(var_xj) = {P}')
```

```
# Constraint

# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Objective
# sum (hi*dij*yij) // for every j and i
temp_lin_expr = gp.LinExpr()
for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, hi*dij)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        print('%s %g' % (v.VarName, v.X))
print('Obj: %g' % m.ObjVal)

if __name__ == '__main__':
    main()
```

### 2.6.7 Example 5

This is an example from the Thesis of (Paraskevopoulou, 2015). The goal of the example is to place one or two supply centers in some nodes (Xalkida, Athina, Theva, Amfissa, Mesologgi, Lamia). The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 18.72 & 24.96 & 63.36 & 74.16 & 25.44 \\ 18.72 & 0 & 33.36 & 62.16 & 58.56 & 51.36 \\ 24.96 & 33.36 & 0 & 45.84 & 47.28 & 26.4 \\ 63.36 & 62.16 & 45.84 & 0 & 40.56 & 24.48 \\ 74.16 & 58.56 & 47.28 & 40.56 & 0 & 50.16 \\ 39.84 & 51.36 & 26.4 & 24.48 & 50.16 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [34 \quad 1140 \quad 3 \quad 4 \quad 16 \quad 17]$$

Running the code, we see that the results from this code do not match the reported optimal solution.

$$X_{athina}, X_{lamia} = 1$$

$$Y_{xalkida\_athina}, Y_{athina\_athina}, Y_{theva\_lamia}, Y_{amfissa\_lamia}, Y_{mesologgi\_lamia}, Y_{lamia\_lamia} = 1$$

$$Z = 1616.16$$

We also notice that the reported optimal objective function result  $Z = 75$  is abnormally low. Further investigation of (Paraskevopoulou, 2015) shows that one of the constraints is erroneously modeled as  $\sum_{i \in I} \sum_{j \in J} Y_{ij} = 1$  instead of  $\sum_{j \in J} Y_{ij} = 1 \forall i \in I$ . This is the reason for this deviation.

### 2.6.7.1 Code

```
#####
# Maximum covering model with median formulation, cost minimization #
# Chapter 8.5.1, p.72 #
# Linear Programming Models for Optimal Location Problem #
# Paraskevopoulou 2015 #
# #
# #
# #
# Code by Foteini Bourou, 2022 #
# #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    # Set locations and demand nodes
    locations = ['xalkida', 'athina', 'theva', 'amfissa', 'mesologgi', 'lamia']

    # const_dij -> Distance between demand node i and candidate j
    const_dij = [[0,18.72,24.96,63.36,74.16,25.44],
                  [18.72,0,33.36,62.16,58.56,51.36],
                  [24.96,33.36,0,45.84,47.28,26.4],
                  [63.36,62.16,45.84,0,40.56,24.48],
                  [74.16,58.56,47.28,40.56,0,50.16],
                  [39.84,51.36,26.4,24.48,50.16,0]]

    # Desired Locations
    P = 1

    # const_hi -> Demand at node j
    const_hi = [34,1140,3,4,16,17]
```

```
#####

m=gp.Model('p-median')

# Variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in locations]

# Variables
var_yij = []
for loci in locations:
    var_yi = [] # temporary list
    for locj in locations:
        var_yi.append(m.addVar(vtype=GRB.BINARY, name=f'y_{loci}_{locj}'))
    var_yij.append(var_yi)

# Constraint
# sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum(y_{i}) = 1')

# Constraint
# sum(X_j) = P // sum over j
temp_lin_expr = gp.LinExpr()
for xj in var_xj:
    temp_lin_expr.add(xj, 1.0)
m.addConstr(temp_lin_expr == P, f'sum(var_xj) = {P}')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Objective
# sum (hi*dij*Yij) // for every j and i
temp_lin_expr = gp.LinExpr()
for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, hi*dij)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
```

```
for v in m.getVars():  
    if v.X:  
        print('%s %g' % (v.VarName, v.X))  
    print('Obj: %g' % m.ObjVal)  
  
if __name__ == '__main__':  
    main()
```

## 2.7 Uncapacitated fixed charge facility location problems

### 2.7.1 Generic description

In this model, there is a set of demand nodes and a set of candidate facility locations. The number of facilities to be located, contrary to previous models, is not defined contrary. When a facility is located automatically there is a cost of setting-up. Moreover, there is a cost for every transfer of product from a candidate facility location to a demand node. The goal is to find the optimal facility locations and the optimal ways of transferring, so that it will be achieved the minimization of the total cost and there will be covered all demand nodes (Daskin, Snyder, & Berger, Facility Location in Supply Chain Design, 2005).

Uncapacitated location problems have been studied through distinct views like clustering analysis, communication networks and bank account perspective. Clustering analysis is used in many areas such as biology, medicine, marketing research and statistics (Guha & Khuller, 1998), (Cornuejols, Nemhauser, & Wolsey, 1990).

### 2.7.2 Mathematical formulation

The mathematical formulation is the following:

Decision variables

$$X_j = \begin{cases} 1 & \text{if we locate a facility at candidate site } j \in J \\ 0 & \text{if not} \end{cases} \quad (56)$$

$$Y_{ij} = \text{fraction of demand at node } i \in I \text{ that is served by a facility at node } j \in J \quad (57)$$

Constants

$$f_j = \text{fixed cost of locating at candidate site } j \in J \quad (58)$$

$$h_i = \text{demand at node } i \in I \quad (59)$$

$$d_{ij} = \text{distance from demand node } i \in I \text{ to candidate facility site } j \in J \quad (60)$$

$$a = \text{cost per unit distance per unit demand} \quad (61)$$

Minimize

$$Z = \sum_{j \in J} f_j X_j + a \sum_{i \in I} \sum_{j \in J} h_i d_{ij} Y_{ij} \quad (62)$$

Subject to

$$\sum_{j \in J} Y_{ij} = 1 \quad \forall i \in I \quad (63)$$

$$Y_{ij} - X_j \leq 0 \quad \forall i \in I; j \in J \quad (64)$$

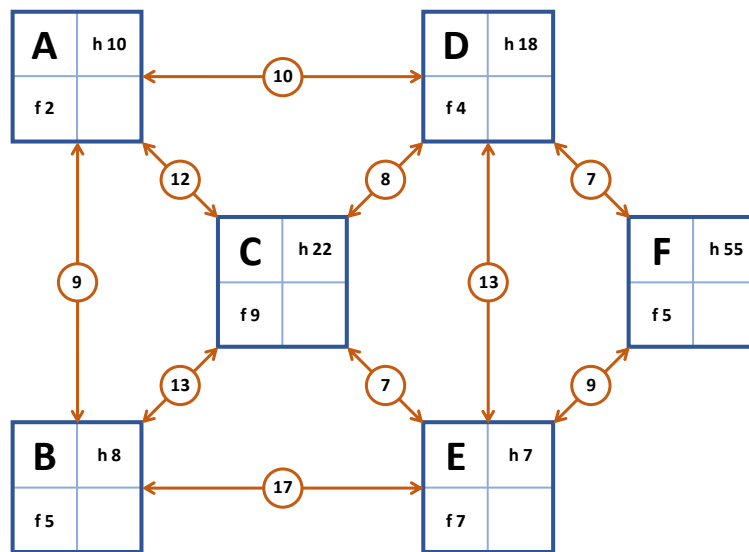
$$X_j \in \{0,1\} \quad \forall j \in J \quad (65)$$

$$Y_{ij} \geq 0 \quad i \in I; \forall j \in J \quad (66)$$

The objective function minimizes the total cost which is the sum of the fixed facility costs and the total demand-weighted distance multiplied by the cost per unit distance per unit demand. The first constraint declares that each demand node  $i \in I$  be served. The second constraint state that demands at node  $i \in I$  cannot be assigned to a facility at candidate site  $j \in J$  unless we locate a facility at node  $j \in J$ . The last constraints show at which interval they belong to.

### 2.7.3 Example 1

This is an example from (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013), where  $h$  is the node demand, and  $f$  the fixed warehouse cost.



**Figure 11 A visualization of the Uncapacitated Fixed Charge Facility Location Model network**

The goal of the example is to place supply centers in some nodes. The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 9 & 12 & 10 & 19 & 17 \\ 9 & 0 & 13 & 21 & 17 & 26 \\ 12 & 13 & 0 & 8 & 7 & 15 \\ 10 & 21 & 8 & 0 & 13 & 7 \\ 19 & 17 & 7 & 13 & 0 & 9 \\ 17 & 26 & 15 & 7 & 9 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [10 \quad 8 \quad 22 \quad 18 \quad 7 \quad 55]$$

the cost to establish a warehouse is

$$f_j = [2 \quad 5 \quad 9 \quad 4 \quad 7 \quad 5]$$

and the cost per unit distance per unit demand is  $a = 21$ . The results of this model are

$$X_A, X_B, X_C, X_D, X_E, X_F, Y_{AA}, Y_{BB}, Y_{CC}, Y_{DD}, Y_{EE}, Y_{FF} = 1, Z = 32$$

### 2.7.3.1 Code

```
#####
# Uncapacitated fixed charge facility location problem #
# Chapter 7.2, p.297 #
# Network and Discrete Location Models, 2e, Daskin #
# #
# #
# #
# Code by Foteini Bourou, 2022 #
# #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    locations = ['a', 'b', 'c', 'd', 'e', 'f']
    client_demands_cost_per_warehouse = [[ 0, 9, 12, 10, 19, 17],
                                           [ 9, 0, 13, 21, 17, 26],
                                           [12, 13, 0, 8, 7, 15],
                                           [10, 21, 8, 0, 13, 7],
                                           [19, 17, 7, 13, 0, 9],
                                           [17, 26, 15, 7, 9, 0]]

    client_demands = [10, 8, 22, 18, 7, 55]
    warehouse_fixed_costs = [ 2, 5, 9, 4, 7, 5]
    cost_per_unit_distance_per_unit_demand = 21

    #####
```



```

m=gp.Model('UFLP')

# Set Locations and demand nodes
warehouses = locations
clients = locations

# const_dij -> Distance between demand node i and candidate j
const_dij = client_demands_cost_per_warehouse

# const_hi -> Demand at node i
const_hi = client_demands

# const_fj -> fixed cost of locating at candidate site j
const_fj = warehouse_fixed_costs

# const_a -> cost per unit distance per unit demand
const_a = cost_per_unit_distance_per_unit_demand

# Add variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in warehouses]
var_yij = []
for loci in clients:
    var_yi = [] # temporary list
    for locj in warehouses:
        var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
        var_yij.append(var_yi)

# Constraint
# sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j = 1')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Objective function
# Minimize sum (hi*dij*Yij) + sum(fj*Xj) // for every i and j
temp_lin_expr = gp.LinExpr()
for fj, xj in zip(const_fj, var_xj):
    temp_lin_expr.add(xj, fj)

```

```

for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, const_a*dij*hi)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        if v.VarName.startswith('x'):
            print('%s %d' % (v.VarName, v.X))
        if v.VarName.startswith('y'):
            print('%s %f' % (v.VarName, v.X))

print('Obj: %f' % m.ObjVal)

if __name__ == '__main__':
    main()

```

## 2.7.4 Example 2

This is an example based on dataset cap71.txt from (Beasley, 1990). These models have the purpose of being verifications standards whose solution is the one that any code that solves such problems can be evaluated against the reference solution. Executing the code, we see that results match the reference optimal solution.

### 2.7.4.1 Code

```

#####
# Uncapacitated fixed charge facility location problem #
# cap71.txt #
# OR-Library: Distributing Test Problems by Electronic Mail #
# J. E. Beasley #
# #
# #
# Code by Foteini Bourou, 2022 #
# #
# #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    path_to_input_file = 'beasley_cap71.txt'
    input_file = open(path_to_input_file, 'r')
    lines = input_file.read().split('\n')
    input_file.close()

    warehouse_locations_num = int(lines[0].split()[0])

```

```

warehouse_locations = [str(i) for i in range(1, warehouse_locations_num + 1)]
warehouse_fixed_costs = [float(line.split()[1]) for line in
lines[1:warehouse_locations_num + 1]]
client_locations_num = int(lines[0].split()[1])
client_locations = [str(i) for i in range(1, client_locations_num + 1)]
client_demands = []
client_demands_cost_per_warehouse = []

lnwcs = warehouse_locations_num # one line before
line_number_where_clients_start
current_line = lnwcs
for i in range(client_locations_num):
    current_line += 1
    client_demand = float(lines[current_line].split()[0])
    client_demands.append(client_demand)
    read_demand_cost_per_warehouse = []
    while len(read_demand_cost_per_warehouse) < warehouse_locations_num:
        current_line += 1
        numbers_in_line = [float(token) for token in lines[current_line].split()]
        read_demand_cost_per_warehouse.extend(numbers_in_line)
        read_demand_cost_per_warehouse = [cost/client_demand for cost in
read_demand_cost_per_warehouse]
        client_demands_cost_per_warehouse.append(read_demand_cost_per_warehouse)

cost_per_unit_distance_per_unit_demand = 1

#####

m=gp.Model('UFLP')

# Set Locations and demand nodes
warehouses = warehouse_locations
clients = client_locations

# const_dij -> Distance between demand node i and candidate j
const_dij = client_demands_cost_per_warehouse

# const_hi -> Demand at node i
const_hi = client_demands

# const_fj -> fixed cost of locating at candidate site j
const_fj = warehouse_fixed_costs

# const_a -> cost per unit distance per unit demand
const_a = cost_per_unit_distance_per_unit_demand

# Add variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in warehouses]
var_yij = []
for loci in clients:
    var_yi = [] # temporary list
    for locj in warehouses:
        var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
    var_yij.append(var_yi)

```

```
# Constraint
# sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j = 1')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Objective function
# Minimize sum (hi*dij*Yij) + sum(fj*Xj) // for every i and j
temp_lin_expr = gp.LinExpr()
for fj, xj in zip(const_fj, var_xj):
    temp_lin_expr.add(xj, fj)

for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, const_a*dij*hi)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        if v.VarName.startswith('x'):
            print('%s %d' % (v.VarName, v.X))
        if v.VarName.startswith('y'):
            print('%s %f' % (v.VarName, v.X))

print('Obj: %f' % m.ObjVal)

if __name__ == '__main__':
    main()
```

### 2.7.5 Example 3

This is an example from the Thesis of (Chatzigiannis, 2013). The goal of the example is to place supply centers in some nodes (Attiki, Theva, Lamia, Mesologgi, Xalkida). The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 33.36 & 51.36 & 58.56 & 18.72 \\ 33.36 & 0 & 26.4 & 47.28 & 24.96 \\ 51.36 & 26.4 & 0 & 50.16 & 39.84 \\ 58.56 & 47.28 & 50.16 & 0 & 74.16 \\ 18.72 & 24.96 & 39.84 & 74.16 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [72 \quad 2 \quad 11 \quad 25 \quad 9]$$

the cost to establish a warehouse is

$$f_j = [1960000 \quad 1624000 \quad 1636000 \quad 1612000 \quad 1708000]$$

and the cost per unit distance per unit demand is  $a = 1$ . Running the code, we see that the results from this code partially match the reported optimal solution.

$$X_{mesologgi} = 1$$

$$Y_{attiki\_mesologgi}, Y_{theva\_mesologgi}, Y_{lamia\_mesologgi} = 1$$

$$Y_{mesologgi\_mesologgi}, Y_{xalkida\_mesologgi} = 1$$

$$Z = 1617530$$

The calculated optimal objective function value of 1617530.08 is very close to the reported value of 1612095. Further investigation of (Chatzigiannis, 2013) shows that one of the constraints is erroneously modeled as  $\sum_{i \in I} \sum_{j \in J} Y_{ij} = 1$  instead of  $\sum_{j \in J} Y_{ij} = 1 \forall i \in I$ . This is the reason for this deviation.

#### 2.7.5.1 Code

```
#####
# Uncapacitated fixed charge facility location problem #
# Chapter 6.4.2, p.86 #
# Location models within supply chains: analysis and application #
# Ioannis Chatzigiannis, 2013 #
# #
# #
# #
# Code by Foteini Bourou, 2022 #
```

```
#
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    locations = ['attiki', 'theva', 'lamia', 'mesologgi', 'xalkida']
    client_demands_cost_per_warehouse = [[0, 33.36, 51.36, 58.56, 18.72],
                                           [33.36, 0, 26.4, 47.28, 24.96],
                                           [51.36, 26.4, 0, 50.16, 39.84],
                                           [58.56, 47.28, 50.16, 0, 74.16],
                                           [18.72, 24.96, 39.84, 74.16, 0]]

    client_demands = [72, 2, 11, 25, 9]
    warehouse_fixed_costs = [1960000, 1624000, 1636000, 1612000, 1708000]
    cost_per_unit_distance_per_unit_demand = 1

    #####

    m=gp.Model('UFLP')

    # Set Locations and demand nodes
    warehouses = locations
    clients = locations

    # const_dij -> Distance between demand node i and candidate j
    const_dij = client_demands_cost_per_warehouse

    # const_hi -> Demand at node i
    const_hi = client_demands

    # const_fj -> fixed cost of locating at candidate site j
    const_fj = warehouse_fixed_costs

    # const_a -> cost per unit distance per unit demand
    const_a = cost_per_unit_distance_per_unit_demand

    # Add variables
    var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in warehouses]
    var_yij = []
    for loci in clients:
        var_yi = [] # temporary list
        for locj in warehouses:
            var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
        var_yij.append(var_yi)

    # Constraint
    # sum(Y_ij) = 1 // sum over j // for every i
    i = 0
    for yi in var_yij:
        i += 1
        temp_lin_expr = gp.LinExpr()
        for yij in yi:
            temp_lin_expr.add(yij, 1.0)
```

```

m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_{j} = 1')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Objective function
# Minimize sum (hi*dij*Yij) + sum(fj*Xj) // for every i and j
temp_lin_expr = gp.LinExpr()
for fj, xj in zip(const_fj, var_xj):
    temp_lin_expr.add(xj, fj)

for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, const_a*dij*hi)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        if v.VarName.startswith('x'):
            print('%s %d' % (v.VarName, v.X))
        if v.VarName.startswith('y'):
            print('%s %f' % (v.VarName, v.X))

print('Obj: %f' % m.ObjVal)

if __name__ == '__main__':
    main()

```

## 2.7.6 Example 4

This is an example from the Thesis of (Paraskevopoulou, 2015). The goal of the example is to place one or two supply centers in some nodes (Xalkida, Athina, Theva, Amfissa, Mesologgi, Lamia). The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 18.72 & 24.96 & 63.36 & 74.16 & 25.44 \\ 18.72 & 0 & 33.36 & 62.16 & 58.56 & 51.36 \\ 24.96 & 33.36 & 0 & 45.84 & 47.28 & 26.4 \\ 63.36 & 62.16 & 45.84 & 0 & 40.56 & 24.48 \\ 74.16 & 58.56 & 47.28 & 40.56 & 0 & 50.16 \\ 39.84 & 51.36 & 26.4 & 24.48 & 50.16 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [34 \quad 1140 \quad 3 \quad 4 \quad 16 \quad 17]$$

the cost to establish a warehouse is

$$f_j = [495000 \quad 540000 \quad 483300 \quad 482250 \quad 482250 \quad 484800]$$

and the cost per unit distance per unit demand is  $a = 1$ . Running the code, we see that the results from this code partially match the reported optimal solution.

$$X_{xalkida} = 1$$

$$Y_{xalkida\_xalkida}, Y_{athina\_xalkida}, Y_{theva\_xalkida} = 1$$

$$Y_{amfissa\_xalkida}, Y_{mesologgi\_xalkida}, Y_{lamia\_xalkida} = 1$$

$$Z = 518533$$

The calculated optimal objective function value of 518533 is not so very close to the reported value of 482388. Further investigation of (Paraskevopoulou, 2015) shows that one of the constraints is erroneously modeled as  $\sum_{i \in I} \sum_{j \in J} Y_{ij} = 1$  instead of  $\sum_{j \in J} Y_{ij} = 1 \forall i \in I$ . This is the reason for this deviation.

### 2.7.6.1 Code

```
#####
# Uncapacitated fixed charge facility location problem #
# Chapter 8.5.2, p.74 #
# Linear Programming Models for Optimal Location Problem #
# Paraskevopoulou 2015 #
# #
# #
# #
# Code by Foteini Bourou, 2022 #
# #
#####

import gurobipy as gp
from gurobipy import GRB
```



```
def main():
    locations = ['xalkida', 'athina', 'theva', 'amfissa', 'mesologgi', 'lamia']
    client_demands_cost_per_warehouse = [[0,18.72,24.96,63.36,74.16,25.44],
                                           [18.72,0,33.36,62.16,58.56,51.36],
                                           [24.96,33.36,0,45.84,47.28,26.4],
                                           [63.36,62.16,45.84,0,40.56,24.48],
                                           [74.16,58.56,47.28,40.56,0,50.16],
                                           [39.84,51.36,26.4,24.48,50.16,0]]

    client_demands = [34,1140,3,4,16,17]
    warehouse_fixed_costs = [495000, 540000, 483300, 482250, 482250, 484800]
    cost_per_unit_distance_per_unit_demand = 1

    #####

    m=gp.Model('UFLP')

    # Set Locations and demand nodes
    warehouses = locations
    clients = locations

    # const_dij -> Distance between demand node i and candidate j
    const_dij = client_demands_cost_per_warehouse

    # const_hi -> Demand at node i
    const_hi = client_demands

    # const_fj -> fixed cost of locating at candidate site j
    const_fj = warehouse_fixed_costs

    # const_a -> cost per unit distance per unit demand
    const_a = cost_per_unit_distance_per_unit_demand

    # Add variables
    var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in warehouses]
    var_yij = []
    for loci in clients:
        var_yi = [] # temporary list
        for locj in warehouses:
            var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
        var_yij.append(var_yi)

    #Constraint
    #sum(Y_ij) = 1 // sum over j // for every i
    i = 0
    for yi in var_yij:
        i += 1
        temp_lin_expr = gp.LinExpr()
        for yij in yi:
            temp_lin_expr.add(yij, 1.0)
        m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j) = 1')

    #Constraint
    # Y_ij - X_j <= 0 // for every i,j
    i = 0
    j = 0
```

```
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Objective function
# Minimize sum (hi*dij*Yij) + sum(fj*Xj) // for every i and j
temp_lin_expr = gp.LinExpr()
for fj, xj in zip(const_fj, var_xj):
    temp_lin_expr.add(xj, fj)

for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, const_a*dij*hi)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        if v.VarName.startswith('x'):
            print('%s %d' % (v.VarName, v.X))
        if v.VarName.startswith('y'):
            print('%s %f' % (v.VarName, v.X))

print('Obj: %f' % m.ObjVal)

if __name__ == '__main__':
    main()
```

## 2.8 Capacitated fixed charge facility location problems

### 2.8.1 Generic description

This model is similar to the previous model. The only difference is that demand at demand nodes cannot be covered only by one facility. Consequently, the demand of some demand nodes will be covered by more than one facility locations (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013). This type of problem is mostly used in cases of locating plants and warehouses. (Li, Yan, & Ren, 2011).

### 2.8.2 Mathematical formulation

The mathematical formulation is the following:

Decision variables

$$X_j = \begin{cases} 1 & \text{if we locate a facility at candidate site } j \in J \\ 0 & \text{if not} \end{cases} \quad (67)$$

$$Y_{ij} = \text{fraction of demand at node } i \in I \text{ that is served by a facility at node } j \in J \quad (68)$$

Constants

$$k_j = \text{capacity of a facility at candidate site } j \in J \text{ if a facility is located there} \quad (69)$$

$$f_j = \text{fixed cost of locating at candidate site } j \in J \quad (70)$$

$$h_i = \text{demand at node } i \in I \quad (71)$$

$$d_{ij} = \text{distance from demand node } i \in I \text{ to candidate facility site } j \in J \quad (72)$$

$$a = \text{cost per unit distance per unit demand} \quad (73)$$

Minimize

$$z = \sum_{j \in J} f_j X_j + a \sum_{i \in I} \sum_{j \in J} h_i d_{ij} Y_{ij} \quad (74)$$

Subject to

$$\sum_{j \in J} Y_{ij} = 1 \quad \forall i \in I \quad (75)$$

$$Y_{ij} - X_j \leq 0 \quad \forall i \in I; j \in J \quad (76)$$

$$\sum_{i \in I} h_i Y_{ij} \leq k_j X_j \quad \forall j \in J \quad (77)$$

$$X_j \in \{0,1\} \quad \forall j \in J \quad (78)$$

$$Y_{ij} \geq 0 \quad i \in I; \forall j \in J \quad (79)$$

The mathematical formulation is the same with the previous model. The only difference is the extra constraint which will ensure that demands at node  $i \in I$  are not assigned to a facility at candidate location  $j \in J$  if we have not selected candidate location  $j \in J$ .

### 2.8.3 Example 1

This is an example from (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013), where  $h$  is the node demand,  $f$  the fixed warehouse cost, and  $k$  the warehouse capacity.

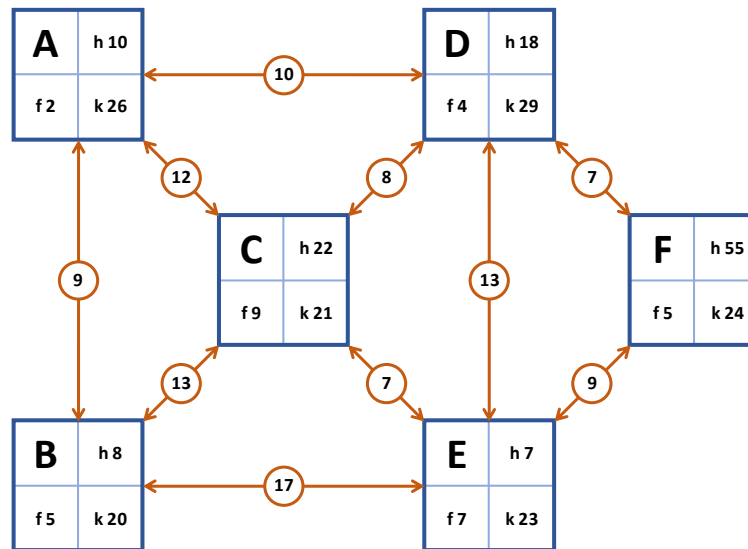


Figure 12 A visualization of the Capacitated Fixed Charge Facility Location Model network

The goal of the example is to place supply centers in some nodes. The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 9 & 12 & 10 & 19 & 17 \\ 9 & 0 & 13 & 21 & 17 & 26 \\ 12 & 13 & 0 & 8 & 7 & 15 \\ 10 & 21 & 8 & 0 & 13 & 7 \\ 19 & 17 & 7 & 13 & 0 & 9 \\ 17 & 26 & 15 & 7 & 9 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [10 \quad 8 \quad 22 \quad 18 \quad 7 \quad 55]$$

the cost to establish a warehouse is

$$f_j = [2 \quad 5 \quad 9 \quad 4 \quad 7 \quad 5]$$

the warehouse capacity vector is

$$k_j = [26 \quad 20 \quad 21 \quad 29 \quad 23 \quad 24]$$

and the cost per unit distance per unit demand is  $a = 21$ . The results of this model are

$$Y_{AA}, Y_{BB}, Y_{DD}, Y_{EE} = 1$$

$$Y_{CA} = 0.045, Y_{CC} = 0.955$$

$$Y_{FA} = 0.073, Y_{FD} = 0.2, Y_{FE} = 0.291, Y_{FF} = 0.436$$

$$Z = 333$$

### 2.8.3.1 Code

```
#####
# Capacitated fixed charge facility location problem #
# Chapter 7.3, p.326 #
# Network and Discrete Location Models, 2e, Daskin #
# #
# Parametric solution #
# #
# #
# #
# Code by Foteini Bourou, 2022 #
# #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    locations = ['a', 'b', 'c', 'd', 'e', 'f']
    client_demands_cost_per_warehouse = [[ 0, 9, 12, 10, 19, 17],
                                           [ 9, 0, 13, 21, 17, 26],
```

```

[12, 13, 0, 8, 7, 15],
[10, 21, 8, 0, 13, 7],
[19, 17, 7, 13, 0, 9],
[17, 26, 15, 7, 9, 0]]

client_demands = [10, 8, 22, 18, 7, 55]
warehouse_fixed_costs = [ 2, 5, 9, 4, 7, 5]
warehouse_capacities = [ 26, 20, 21, 29, 23, 24]
cost_per_unit_distance_per_unit_demand = 21

#####

m=gp.Model('CFLP')

# Set locations and demand nodes
warehouses = locations
clients = locations

# const_dij -> Distance between demand node i and candidate j
const_dij = client_demands_cost_per_warehouse

# const_hi -> Demand at node i
const_hi = client_demands

# const_fj -> fixed cost of locating at candidate site j
const_fj = warehouse_fixed_costs

# const_kj -> capacity of facility at site j
const_kj = warehouse_capacities

# const_a -> cost per unit distance per unit demand
const_a = 1

# Add variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in warehouses]
var_yij = []
for loci in clients:
    var_yi = [] # temporary list
    for locj in warehouses:
        var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
    var_yij.append(var_yi)

# Constraint
# sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j) = 1')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0

```

```

for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Constraint
# sum(h_i * Y_ij) - k_j * X_j <= 0 // sum over i // for every j
for j in range(len(var_yij[0])):
    temp_lin_expr = gp.LinExpr()
    for i in range(len(var_xj)):
        temp_lin_expr.add(var_yij[i][j], const_hi[i])
        temp_lin_expr.add(var_xj[j], -const_kj[j])
    m.addConstr(temp_lin_expr <= 0, f'h_{i}*y_{i}_{j} - k_{j}*x_{j} <= 0')

# Objective function
# Minimize sum (hi*dij*Yij) + sum(fj*Xj) // for every i and j
temp_lin_expr = gp.LinExpr()
for fj, xj in zip(const_fj, var_xj):
    temp_lin_expr.add(xj, fj)

for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, const_a*dij*hi)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        if v.VarName.startswith('x'):
            print('%s %d' % (v.VarName, v.X))
        if v.VarName.startswith('y'):
            print('%s %f' % (v.VarName, v.X))

print('Obj: %f' % m.ObjVal)

if __name__ == '__main__':
    main()

```

## 2.8.4 Example 2

This is an example based on dataset cap71.txt from (Beasley, 1990). These models have the purpose of being verifications standards whose solution is the one that any code that solves such problems can be evaluated against the reference solution. Executing the code, we see that results match the reference optimal solution.

### 2.8.4.1 Code

```
#####
# Capacitated fixed charge facility location problem #
# cap71.txt #
# OR-Library: Distributing Test Problems by Electronic Mail #
# J. E. Beasley #
# #
# #
# Parametric solution #
# #
# Code by Foteini Bourou, 2022 #
# #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    path_to_input_file = 'beasley_cap71.txt'
    input_file = open(path_to_input_file, 'r')
    lines = input_file.read().split('\n')
    input_file.close()

    warehouse_locations_num = int(lines[0].split()[0])
    warehouse_locations = [str(i) for i in range(1, warehouse_locations_num + 1)]
    warehouse_capacities = [float(line.split()[0]) for line in
lines[1:warehouse_locations_num + 1]]
    warehouse_fixed_costs = [float(line.split()[1]) for line in
lines[1:warehouse_locations_num + 1]]
    client_locations_num = int(lines[0].split()[1])
    client_locations = [str(i) for i in range(1, client_locations_num + 1)]
    client_demands = []
    client_demands_cost_per_warehouse = []

    lnwcs = warehouse_locations_num # one line before
line_number_where_clients_start
    current_line = lnwcs
    for i in range(client_locations_num):
        current_line += 1
        client_demand = float(lines[current_line].split()[0])
        client_demands.append(client_demand)
        read_demand_cost_per_warehouse = []
        while len(read_demand_cost_per_warehouse) < warehouse_locations_num:
            current_line += 1
            numbers_in_line = [float(token) for token in lines[current_line].split()]
            read_demand_cost_per_warehouse.extend(numbers_in_line)
            read_demand_cost_per_warehouse = [cost/client_demand for cost in
read_demand_cost_per_warehouse]
            client_demands_cost_per_warehouse.append(read_demand_cost_per_warehouse)

#####

m=gp.Model('CFLP')
```



```
# Set Locations and demand nodes
warehouses = warehouse_locations
clients = client_locations

# const_dij -> Distance between demand node i and candidate j
const_dij = client_demands_cost_per_warehouse

# const_hi -> Demand at node i
const_hi = client_demands

# const_fj -> fixed cost of locating at candidate site j
const_fj = warehouse_fixed_costs

# const_kj -> capacity of facility at site j
const_kj = warehouse_capacities

# const_a -> cost per unit distance per unit demand
const_a = 1

# Add variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in warehouses]
var_yij = []
for loci in clients:
    var_yi = [] # temporary list
    for locj in warehouses:
        var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
        var_yij.append(var_yi)

# Constraint
# sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j = 1')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Constraint
# sum(h_i * Y_ij) - k_j * X_j <= 0 // sum over i // for every j
for j in range(len(var_yij[0])):
    temp_lin_expr = gp.LinExpr()
```

```

for i in range(len(var_xj)):
    temp_lin_expr.add(var_yij[i][j], const_hi[i])
    temp_lin_expr.add(var_xj[j], -const_kj[j])
    m.addConstr(temp_lin_expr <= 0, f'_h_{i}*y_{i}_{j} - k_{j}*x_{j} <= 0')

# Objective function
# Minimize sum (hi*dij*Yij) + sum(fj*Xj) // for every i and j
temp_lin_expr = gp.LinExpr()
for fj, xj in zip(const_fj, var_xj):
    temp_lin_expr.add(xj, fj)

for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, const_a*dij*hi)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        if v.VarName.startswith('x'):
            print('%s %d' % (v.VarName, v.X))
        if v.VarName.startswith('y'):
            print('%s %f' % (v.VarName, v.X))

print('Obj: %f' % m.ObjVal)

if __name__ == '__main__':
    main()

```

### 2.8.5 Example 3

This is an example from the Thesis of (Chatzigiannis, 2013). The goal of the example is to place supply centers in some nodes (Attiki, Theva, Lamia, Mesologgi, Xalkida). The distance matrix is

$$d_{ij} = \begin{bmatrix} 0 & 33.36 & 51.36 & 58.56 & 18.72 \\ 33.36 & 0 & 26.4 & 47.28 & 24.96 \\ 51.36 & 26.4 & 0 & 50.16 & 39.84 \\ 58.56 & 47.28 & 50.16 & 0 & 74.16 \\ 18.72 & 24.96 & 39.84 & 74.16 & 0 \end{bmatrix}$$

and the demand vector is

$$h_i = [72 \quad 2 \quad 11 \quad 25 \quad 9]$$

and the cost to establish a warehouse is

$$f_j = [1960000 \quad 1624000 \quad 1636000 \quad 1612000 \quad 1708000]$$

and the capacity of the warehouses would be

$$f_j = [80 \quad 100 \quad 100 \quad 100 \quad 80]$$

and the cost per unit distance per unit demand is  $a = 1$ . Running the code, we see that the results from this code do not match the reported optimal solution.

$$X_{theva}, X_{mesologgi} = 1$$

$$Y_{attiki\_theva}, Y_{theva\_theva}, Y_{lamia\_theva}, Y_{mesologgi\_mesologgi}, Y_{xalkida\_theva} = 1$$

$$Z = 3238917$$

We also notice that the reported optimal objective function of 50 result is abnormally low, especially in comparison to this code's calculated value of 3238917. Further investigation of (Chatzigiannis, 2013) shows that one of the constraints is erroneously modeled as  $\sum_{i \in I} \sum_{j \in J} Y_{ij} = 1$  instead of  $\sum_{j \in J} Y_{ij} = 1 \forall i \in I$ . This is the reason for this deviation.

### 2.8.5.1 Code

```
#####
# Capacitated fixed charge facility location problem #
# Chapter 6.4.3, p.88 #
# Location models within supply chains: analysis and application #
# Ioannis Chatzigiannis, 2013 #
# #
# #
# #
# Code by Foteini Bourou, 2022 #
# #
#####

import gurobipy as gp
from gurobipy import GRB

def main():
    locations = ['attiki', 'theva', 'lamia', 'mesologgi', 'xalkida']
    client_demands_cost_per_warehouse = [[0, 33.36, 51.36, 58.56, 18.72],
                                           [33.36, 0, 26.4, 47.28, 24.96],
                                           [51.36, 26.4, 0, 50.16, 39.84],
                                           [58.56, 47.28, 50.16, 0, 74.16],
                                           [18.72, 24.96, 39.84, 74.16, 0]]

    client_demands = [72, 2, 11, 25, 9]
    warehouse_fixed_costs = [1960000, 1624000, 1636000, 1612000, 1708000]
    warehouse_capacities = [80, 100, 100, 100, 80]
    cost_per_unit_distance_per_unit_demand = 1

    #####

    m=gp.Model('CFLP')
```

```
# Set Locations and demand nodes
warehouses = locations
clients = locations

# const_dij -> Distance between demand node i and candidate j
const_dij = client_demands_cost_per_warehouse

# const_hi -> Demand at node i
const_hi = client_demands

# const_fj -> fixed cost of locating at candidate site j
const_fj = warehouse_fixed_costs

# const_kj -> capacity of facility at site j
const_kj = warehouse_capacities

# const_a -> cost per unit distance per unit demand
const_a = 1

# Add variables
var_xj = [m.addVar(vtype=GRB.BINARY, name=f'x_{loc}') for loc in warehouses]
var_yij = []
for loci in clients:
    var_yi = [] # temporary list
    for locj in warehouses:
        var_yi.append(m.addVar(lb=0, ub=1, vtype=GRB.CONTINUOUS,
name=f'y_{loci}_{locj}'))
        var_yij.append(var_yi)

# Constraint
# sum(Y_ij) = 1 // sum over j // for every i
i = 0
for yi in var_yij:
    i += 1
    temp_lin_expr = gp.LinExpr()
    for yij in yi:
        temp_lin_expr.add(yij, 1.0)
    m.addConstr(temp_lin_expr == 1, f'sum_y_{i}_j = 1')

# Constraint
# Y_ij - X_j <= 0 // for every i,j
i = 0
j = 0
for yi in var_yij:
    i += 1
    for yij, xj in zip(yi, var_xj):
        j += 1
        temp_lin_expr = gp.LinExpr()
        temp_lin_expr.add(yij, 1.0)
        temp_lin_expr.add(xj, -1.0)
        m.addConstr(temp_lin_expr <= 0, f'y_{i}_{j} - x_{j} <= 0')

# Constraint
# sum(h_i * Y_ij) - k_j * X_j <= 0 // sum over i // for every j
for j in range(len(var_yij[0])):
```

```
temp_lin_expr = gp.LinExpr()
for i in range(len(var_xj)):
    temp_lin_expr.add(var_yij[i][j], const_hi[i])
temp_lin_expr.add(var_xj[j], -const_kj[j])
m.addConstr(temp_lin_expr <= 0, f'_h_{i}*y_{i}_{j} - k_{j}*x_{j} <= 0')

# Objective function
# Minimize sum (hi*dij*Yij) + sum(fj*Xj) // for every i and j
temp_lin_expr = gp.LinExpr()
for fj, xj in zip(const_fj, var_xj):
    temp_lin_expr.add(xj, fj)

for hi, di, yi in zip(const_hi, const_dij, var_yij):
    for dij, yij in zip(di, yi):
        temp_lin_expr.add(yij, const_a*dij*hi)
m.setObjective(temp_lin_expr, GRB.MINIMIZE)

m.optimize()
print('#####')
for v in m.getVars():
    if v.X:
        if v.VarName.startswith('x'):
            print('%s %d' % (v.VarName, v.X))
        if v.VarName.startswith('y'):
            print('%s %f' % (v.VarName, v.X))

print('Obj: %f' % m.ObjVal)

if __name__ == '__main__':
    main()
```

### 3. Computer programming of location models

All of the location problems mentioned in 2. Mathematical models of location problems are mathematical optimization problems. These problems can be solved through the use of well known optimization techniques in the field of integer linear programming (Winston & Goldberg, 2004). A generic description of a computer code that solves these models would be:

- The input part, where the problem description data is provided to the code like variables and constraints
- The object generation that corresponds to the data input where the input data is converted to objects of classes of the code. For example, an integer decision variable with lower bound 0 and upper bound 3 named 'allocated\_warehouses' will be converted to the corresponding object of a class decision variable. This object will have the necessary attributes, e.g. integer type, lower and upper bound.
- The solution procedure where an appropriate algorithm will solve the problem according to its specification. For example, if it is a linear programming problem, it can be solved with simplex (Winston & Goldberg, 2004). If it is an integer linear programming problem then it can be solved with the branch-and-cut method (Padberg & Rinaldi, 1991).
- The output procedure where the final results of the optimization procedure are communicated to the analyst.

#### 3.1 The choice of Python

Python is a high-level, interpreted, general-purpose, multi-paradigm programming language. It supports paradigms like object-oriented programming and it is easy to read and write.

It was deemed as the most suitable programming language to use for this thesis as I was unable to write any code when I started it. Hence, a language easy to read and write was a significant benefit, but Python is also easy to learn.

An extra benefit is the ecosystem of Python. For an absolute beginner in programming, this comprises:

- Easy to access documentation
- Abundance of learning resources, like YouTube video tutorials
- Abundance of on-line answered questions in StackOverflow
- Availability of Integrated Development Environments (IDE) of different complexity
- Easy printing
- Understandable code
- Easy to execute
- Available libraries
- Anaconda distribution

For this Thesis, Python 3.9.5 was used, as part of the respective Anaconda distribution. The IDE used was Geany 1.37.1, which was chosen due to its simplicity. Its simplicity makes it a perfect choice for an absolute beginner.

### **3.2 The choice of Gurobi**

Python has an extensive library of free-to-use code and this includes PuLP, a module that can be used for mathematical optimization problems. However, Gurobi produces the namesake optimization solver which can be used with Python.

This solver was deemed as a better choice as this Thesis does not deal with how to solve mixed-integer linear programming problems from the mathematical point of view, it rather focuses on a more practical approach. For this reason, a ready to use, commercial, reliable and highly capable solver is a better choice that allows to save time. Gurobi is also free for academic use and the license provided lasts for a year.

### **3.3 Object-oriented programming**

The main constraint in using Gurobi is that its Python API works with an object-oriented approach. Since Python is largely an object-oriented language and that object-oriented programming paradigm is a rather good idea, this constraint is actually very well suited to this endeavor.

Throughout the Examples in 2. Mathematical models of location problems there is usage of Gurobi's classes. These classes are relatively simple and easy to understand and use: a model, variables and constraints that are added to the model, and finally linear expressions, which come in rather handy when implementing loops. Linear expressions are objects that can be gradually formed from other linear expressions and are finally used in defining constraints. This capability in particular is what enables a problem-independent algorithm design.

Finally, Gurobi has two ways to specify the mathematical model, one is with object-oriented programming – the one used in this Thesis – and the other is through direct matrix formulation – which was not used in this Thesis at all.

### **3.4 Coding remarks**

The code presented in this Thesis is on a rather basic level. Initially all code formulations were simplistic and largely based on code found on Gurobi tutorials. The major issue with this

approach is that as I gained experience, I was able to reformulate these problems into a more parametric form of code where the problem data input, namely the decision variables and related constants, e.g., cost, were defined separately from the model building part of the code. This allows for an easy reconfiguration of the code files to solve another problem, of different size and constraints.



## **4. Conclusion and suggestions for further research**

This Thesis proved to be a worthwhile journey into coding, mathematical optimization and its applications to supply chain network problems and location modelling. There were many obstacles to overcome, which can be fitted into the following generic categories: location modelling theories, mathematical modelling, coding.

These three categories created a predicament where all three had to share the amount of time available for the completion of this Thesis and were causing trouble to the other. So during this Thesis I had to balance the time allocated in these problems so that the end result is satisfactory. The main cause of obstacles and time delay can be attributed to the coding efforts, as my programming skills were at the absolute beginner's level. For this reason, the final remarks will be split into two parts, the first will involve coding and the second location modelling theories and mathematical modelling.

### **4.1 Code improvement**

While the code delivers the expected results, there are quite significant margins for improvement which will be listed below.

#### **4.1.1 Continuation of code**

Despite the simplicity of the code, there are clear indications that this code can be turned into a more advanced code that will handle location problems in a better way. Perhaps, in a future Thesis, someone with better programming skills will improve on this code.

Maybe such a code can also be part of the lecture material in a supply chain management course of related topic.

#### **4.1.2 Open source**

The code is very simple but it also produces accurate results and is easy to reconfigure and reuse. I believe that such codes, no matter how simple they are, should be available on a webpage where people share coding efforts. Such an example can be found at (Radopoulos, 2021).

#### **4.1.3 File-based input**

A more experienced programmer could take these code samples and combine them into a more advanced code that will accept data input in the form of a text file that contains directives.

#### 4.1.4 Output visualization

The current output of all these codes is merely a print on the terminal of the values of the decision variables and the objective function after optimization. A better programmer might be able to output code results into a spreadsheet file (Radopoulos, 2021) or even maybe include some plot visualization (Buchanan, 2022).

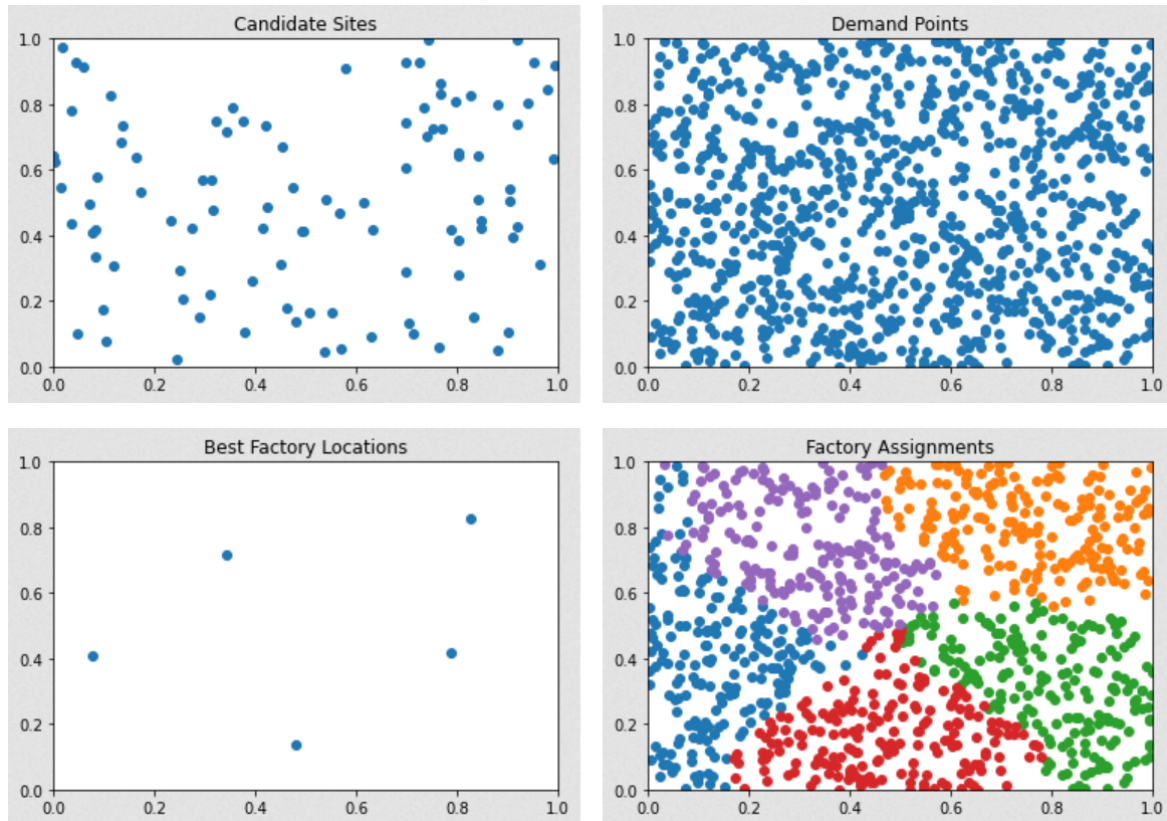


Figure 13 Capacitated facility location visualization with matplotlib, (Buchanan, 2022)

#### 4.1.5 Automated testing

A good programming practice is to have tests that ensure the quality or outcome of the code. This allows for an accelerated development of the code since it can be tested rapidly and thoroughly against known scenarios (Radopoulos, 2021).

### 4.2 Mathematical modelling of location problems

The mathematical models used in this Thesis are only a subset of what is available in the literature. These problems have their own benefits and drawbacks and some lacking features have been observed in them.

#### **4.2.1 Lack of verifiable models**

In several parts of this Thesis, errors in published models were detected. A typographic error was found in (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013), wrong results in (Sitepu, et al., 2019), wrong formulations in (Chatzigiannis, 2013) and (Paraskevopoulou, 2015) and finally some models had no results in (Daskin, Network and Discrete Location: Models, Algorithms, and Applications, Second Edition, 2013).

It is essential for any published method to provide the exact results of sample models and even better the code that was used to produce these results. Else, it could simply be that people that will rely on these publications will simply be led to erroneous results of futile efforts to match these results.

#### **4.2.2 Lack of a holistic approach**

Each of these models have been neglecting something. Some have been neglecting cost, or demand or availability. Apparently, all of them are neglecting routing. It would be best if locations models followed a more holistic approach by encompassing a number of real-world parameters that have been neglected. So the first suggestion is to include all of the traditional parameters in SCM, like availability, running costs, transportation costs and methods, demand variation, distances, storage costs, availability of skilled personnel and others.

#### **4.2.3 Lack of modern routing impact**

There isn't much depth in these models' ability to take into account modern routing, or transportation capabilities in general. Nowadays, we all have in the palm of our hands a computer more powerful than the original Beowulf at NASA (Sterling, et al., 1995). That computer also has the ability to track its location and communicate with other computer or electronic equipment. Taking this thought a step further, we can use the data available from mapping companies to combine location modelling with routing problems. This will allow for a better modelling of the cost associated with transporting goods as well as the delivery time impact.

#### **4.2.4 Load balancing**

Taking inspiration from the load balancing of parallel computing codes and modern computer architecture, we can insert into location algorithms load balancing aspects. That is, if located facilities can operate with dissimilar demand coverage and help each other out when needed. A

similar balancing process is also employed by electric power companies, since not all electric power plants can change their output levels fast enough to compensate for e.g. noon's power surge due to cooking.

#### **4.2.5 Digital twin**

Today we can simulate structures in real-time and keep track of their operational history with electronic means and calculate the remaining capabilities of them. A similar technique might also be applicable in location problems, or even better in resource allocation problems that can be solved with the same models. If we assume that we do not place warehouses but e.g. ambulances, we can keep track of these ambulances through a digital twin and forecast their expected reliability. This forecasted reliability can then be inserted as a parameter in location modelling.

## References

- Beasley, J. E. (1990, November). OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41, 1069–1072. doi:10.1057/jors.1990.166
- Bennett, V. L., Eaton, D. J., & Church, R. L. (1982, January). Selecting sites for rural health workers. *Social Science & Medicine*, 16, 63–72. doi:10.1016/0277-9536(82)90424-5
- Buchanan, A. L. (2022, June). Location Models. *Location Models*. GitHub. Retrieved from [https://github.com/AustinLBuchanan/Location\\_Models](https://github.com/AustinLBuchanan/Location_Models)
- Caprara, A., Kroon, L., Monaci, M., Peeters, M., & Toth, P. (2007). *Passenger railway optimization* (Vol. 14). (C. Barnhart, & G. Laporte, Eds.) Elsevier. doi:[https://doi.org/10.1016/S0927-0507\(06\)14003-7](https://doi.org/10.1016/S0927-0507(06)14003-7)
- Ceria, S., Nobili, P., & Sassano, A. (1998, April). A Lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81, 215–228. doi:10.1007/BF01581106
- Chatzigiannis, I. (2013). Μοντέλα χωροθέτησης εγκαταστάσεων στα πλαίσια εφοδιαστικών αλυσίδων: ανάλυση και εφαρμογή [Master thesis, Aristotle University of Thessaloniki].
- Chopra, S. (2019). *Supply chain management: strategy, planning and operation* (Seventh edition ed.). New, York, NY: Pearson Education.
- Chu, S. C., & Chu, L. (2000, November). A modeling framework for hospital location and service allocation. *International Transactions in Operational Research*, 7, 539–568. doi:10.1111/j.1475-3995.2000.tb00216.x
- Church, R. L., Niblett, M. R., & Gerrard, R. A. (2015). *Modeling the Potential for Critical Habitat*. (H. A. Eiselt, & V. Marianov, Eds.) Cham: Springer International Publishing. doi:10.1007/978-3-319-20282-2\_6
- Church, R. L., Stoms, D. M., & Davis, F. W. (1996). Reserve selection as a maximal covering location problem. *Biological Conservation*, 76, 105–112. doi:10.1016/0006-3207(95)00102-6

- Cornuejols, G., Nemhauser, G. L., & Wolsey, L. A. (1990). *The uncapacitated facility location problem*. (P. B. Mirchandani, & R. L. Francis, Eds.) New York: Wiley.
- Daskin, M. S. (2008, June). What you should know about location modeling. *Naval Research Logistics*, 55, 283–294. doi:10.1002/nav.20284
- Daskin, M. S. (2013, September). *Network and Discrete Location: Models, Algorithms, and Applications, Second Edition*. Hoboken, New, Jersey: John Wiley & Sons, Inc. doi:10.1002/9781118537015
- Daskin, M. S., & Stern, E. H. (1981, May). A Hierarchical Objective Set Covering Model for Emergency Medical Service Vehicle Deployment. *Transportation Science*, 15, 137–152. doi:10.1287/trsc.15.2.137
- Daskin, M. S., Snyder, L. V., & Berger, R. T. (2005). *Facility Location in Supply Chain Design*. (A. Langevin, & D. Riopel, Eds.) New York: Springer-Verlag. doi:10.1007/0-387-24977-X\_2
- Eaton, D. J., Daskin, M. S., Simmons, D., Bulloch, B., & Jansma, G. (1985, February). Determining Emergency Medical Service Vehicle Deployment in Austin, Texas. *Interfaces*, 15, 96–108. doi:10.1287/inte.15.1.96
- Eaton, D. J., Sánchez, U. H., Lantigua, R. R., & Morgan, J. (1986, February). Determining Ambulance Deployment in Santo Domingo, Dominican Republic. *Journal of the Operational Research Society*, 37, 113–126. doi:10.1057/jors.1986.21
- Griffin, P. M., Scherrer, C. R., & Swann, J. L. (2008, July). Optimization of community health center locations and service offerings with statistical need estimation. *IIE Transactions*, 40, 880–892. doi:10.1080/07408170802165864
- Guha, S., & Khuller, S. (1998, January). Greedy strikes back: improved facility location algorithms. *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms* (pp. 649–657). USA: Society for Industrial and Applied Mathematics.
- Hakimi, S. L. (1965, June). Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems. *Operations Research*, 13, 462–475. doi:10.1287/opre.13.3.462

- Hale, T., & Moberg, C. R. (2005, March). Improving supply chain disaster preparedness: A decision process for secure site location. *International Journal of Physical Distribution & Logistics Management*, 35, 195–207. doi:10.1108/09600030510594576
- Hugos, M. H. (2018). *Essentials of supply chain management* (Fourth edition ed.). Hoboken, New, Jersey: John Wiley & Sons, Inc.
- Indriasari, V., Mahmud, A. R., Ahmad, N., & Shariff, A. R. (2010, February). Maximal service area problem for optimal siting of emergency facilities. *International Journal of Geographical Information Science*, 24, 213–230. doi:10.1080/13658810802549162
- Karakostas, P., & Sifaleras, A. (2021). Recent Trends in Sustainable Supply-Chain Optimization. In M. Fathi, E. Zio, & P. M. Pardalos (Eds.), *Handbook of Smart Energy Systems* (pp. 1–23). Cham: Springer International Publishing. doi:10.1007/978-3-030-72322-4\_181-1
- Labbé, M., & Laporte, G. (1986). Maximizing user convenience and postal service efficiency in post box location. *Belgian Journal of Operations Research, Statistics and Computer Science*, 26, 21–35. Retrieved from <https://hal.archives-ouvertes.fr/hal-01255666>
- Laporte, G., Nickel, S., & da Gama, F. S. (Eds.). (2015). *Location science*. Springer International Publishing. doi:10.1007/978-3-319-13111-5
- Li, H., Yan, J., & Ren, M. (2011). *Bender's Algorithm for Facility Location Problem with Uncertain Demand* (Vol. 231). (M. Dai, Ed.) Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-23993-9\_18
- McAleer, W. E., & Naqvi, I. A. (1994, June). The relocation of ambulance stations: A successful case study. *European Journal of Operational Research*, 75, 582–588. doi:10.1016/0377-2217(94)90298-4
- Padberg, M., & Rinaldi, G. (1991, March). A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Review*, 33, 60–100. doi:10.1137/1033004
- Paraskevopoulou, F. (2015). Μοντέλα γραμμικού προγραμματισμού για το πρόβλημα της βέλτιστης χωροθέτησης [Master thesis, Aristotle University of Thessaloniki].



- Plane, D. R., & Hendrick, T. E. (1977, August). Mathematical Programming and the Location of Fire Companies for the Denver Fire Department. *Operations Research*, 25, 563–578. doi:10.1287/opre.25.4.563
- Radopoulos, A. (2021, September). *Development of a decision support system based on Fuzzy PROMETHEE*. Tech. rep., University of Macedonia. Retrieved from <https://dspace.lib.uom.gr/handle/2159/25850>
- Ratick, S. J., Osleeb, J. P., & Hozumi, D. (2009, June). Application and extension of the Moore and ReVelle Hierarchical Maximal Covering Model. *Socio-Economic Planning Sciences*, 43, 92–101. doi:10.1016/j.seps.2008.02.011
- Sitepu, R., Puspita, F. M., Romelda, S., Fikri, A., Susanto, B., & Kaban, H. (2019, July). Set covering models in optimizing the emergency unit location of health facility in Palembang. *Journal of Physics: Conference Series*, 1282, 012008. doi:10.1088/1742-6596/1282/1/012008
- Skouri, K., Sifaleras, A., & Konstantaras, I. (2018). Open Problems in Green Supply Chain Modeling and Optimization with Carbon Emission Targets. In P. M. Pardalos, & A. Migdalas (Eds.), *Open Problems in Optimization and Data Analysis* (pp. 83–90). Cham: Springer International Publishing. doi:10.1007/978-3-319-99142-9\_6
- Stanton, D. (2018). *Supply chain management*. Hoboken, NJ: John Wiley & Sons, Inc.
- Sterling, T. L., Savarese, D., Becker, D. J., Dorband, J. E., Ranawake, U. A., & Packer, C. V. (1995). BEOWULF: A Parallel Workstation for Scientific Computation. In P. Banerjee (Ed.), *Proceedings of the 1995 International Conference on Parallel Processing, Urbana-Champaign, Illinois, USA, August 14-18, 1995. Volume I: Architecture* (pp. 11–14). CRC Press.
- Storbeck, J. E. (1988). The spatial structuring of central places. *Geographical Analysis*, 20, 93–110. doi:<https://doi.org/10.1111/j.1538-4632.1988.tb00169.x>
- Swersey, A. J. (1994). *Chapter 6 The deployment of police, fire, and emergency medical units* (Vol. 6). Elsevier. doi:[https://doi.org/10.1016/S0927-0507\(05\)80087-8](https://doi.org/10.1016/S0927-0507(05)80087-8)



- Toregas, C., Swain, R., ReVelle, C., & Bergman, L. (1971, October). The Location of Emergency Service Facilities. *Operations Research*, 19, 1363–1373. doi:10.1287/opre.19.6.1363
- van Hoesel, C. P., Goossens, J. H., & Kroon, L. G. (2004, January). *Optimising halting station of passenger railway lines*. Tech. rep., Maastricht University, Maastricht Research School of Economics of Technology and Organization (METEOR). doi:10.26481/umamet.2004016
- Verter, V., & Lapierre, S. D. (2002). Location of Preventive Health Care Facilities. *Annals of Operations Research*, 110, 123–132. doi:10.1023/A:1020767501233
- Winston, W. L., & Goldberg, J. B. (2004). *Operations research: applications and algorithms* (4 ed.). Belmont, CA: Thomson/Brooks/Cole.

Author's Statement:

I hereby expressly declare that, according to the article 8 of Law 1559/1986, this dissertation is solely the product of my personal work, does not infringe any intellectual property, personality and personal data rights of third parties, does not contain works/contributions from third parties for which the permission of the authors/beneficiaries is required, is not the product of partial or total plagiarism, and that the sources used are limited to the literature references alone and meet the rules of scientific citations.