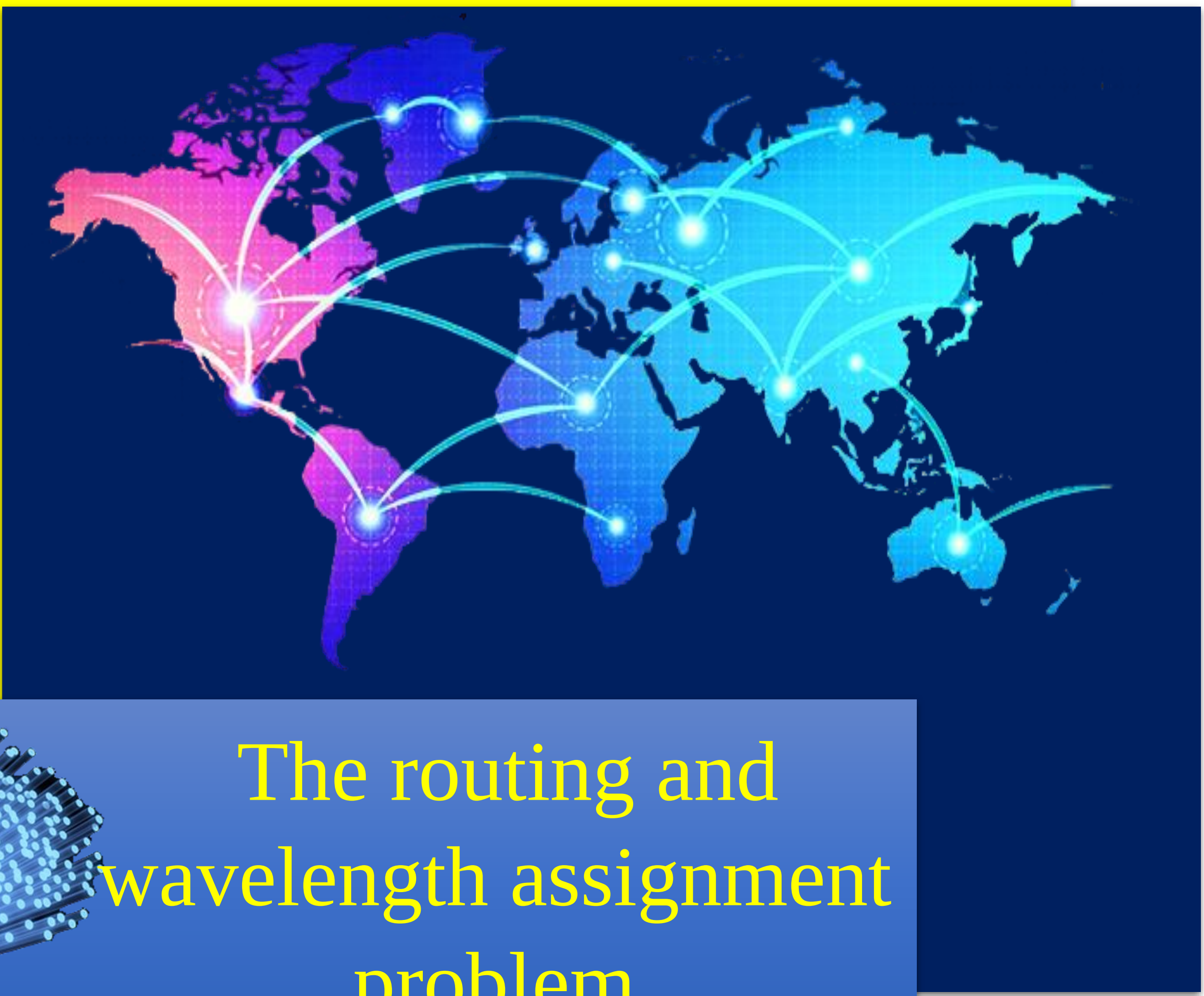
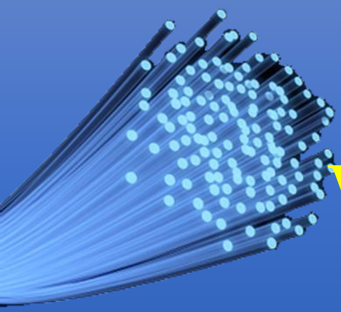
2019/2020





The routing and wavelength assignment problem

Made by

LAHRACH OMAR

MAKMOULI YASSINE

MAZER OMAR

supervisor:

Dr.Ing.KHAROUBI Fouad

Table of Contents

1. Introduction:

The routing and wavelength assignment (RWA) problem is an optical networking problem with the goal of maximizing the number of optical connections.

The general objective of the RWA problem is to maximize the number of established connections. Each connection request must be given a route and wavelength. The wavelength must be consistent for the entire path, unless the usage of wavelength converters is assumed. Two connections requests can share the same optical link, provided a different wavelength is used.

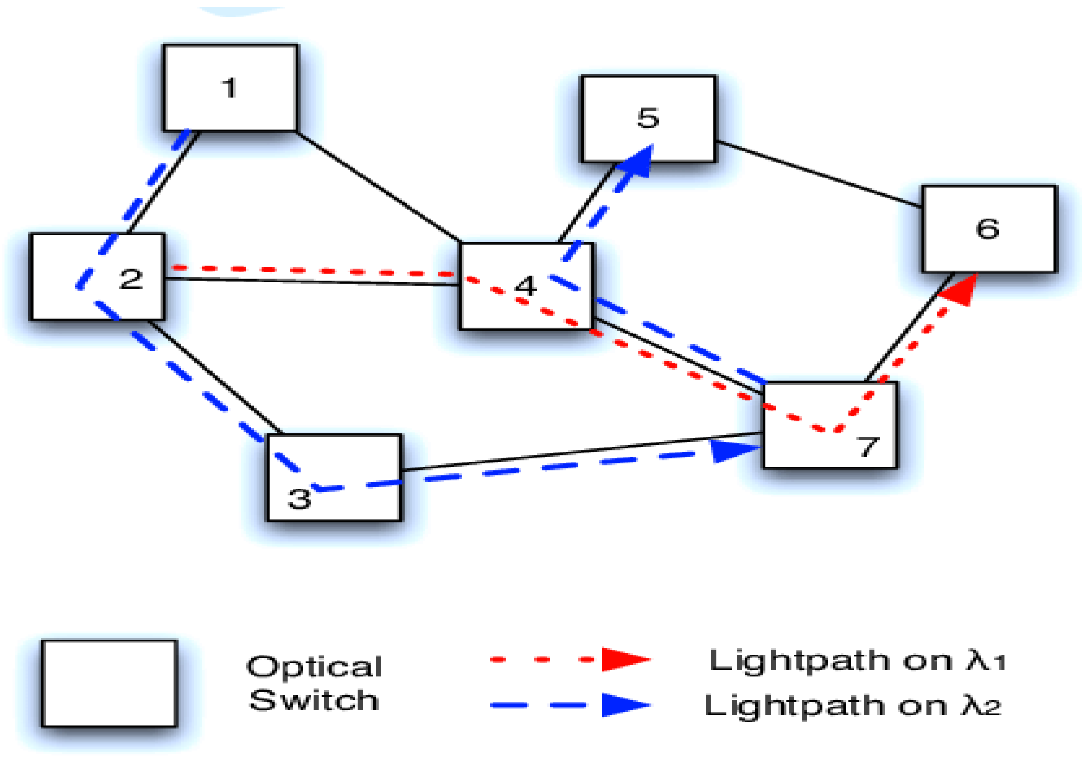


Figure 1

2. Tools:

In order to achieve our application, we were concerned with both simplicity and efficiency so that we can gain time and better result at same time

The following are the factors to be considered:

- The targeted platform
- The elasticity of a language
- The time to production
- The performance

2.1 Python:

Python is a general purpose and high-level programming language. You can use Python for developing desktop GUI applications, websites and web applications. Also, Python, as a high-level programming language, allows you to focus on core functionality of the application by taking care of common programming tasks

Figure 2

2.2 Anaconda Spyder:

Spyder, the Scientific Python Development Environment, is a free integrated development environment (IDE) that is included with Anaconda. It includes editing, interactive testing, debugging, and introspection features.

Spyder is also pre-installed in Anaconda Navigator, which is included in Anaconda

Figure 3

2.3 Qt Designer:

Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. ... Widgets and forms created with Qt Designer integrate seamlessly with programmed code, using Qt's signals and slots mechanism, so that you can easily assign behaviour to graphical elements.

Figure 4

3. flowchart:

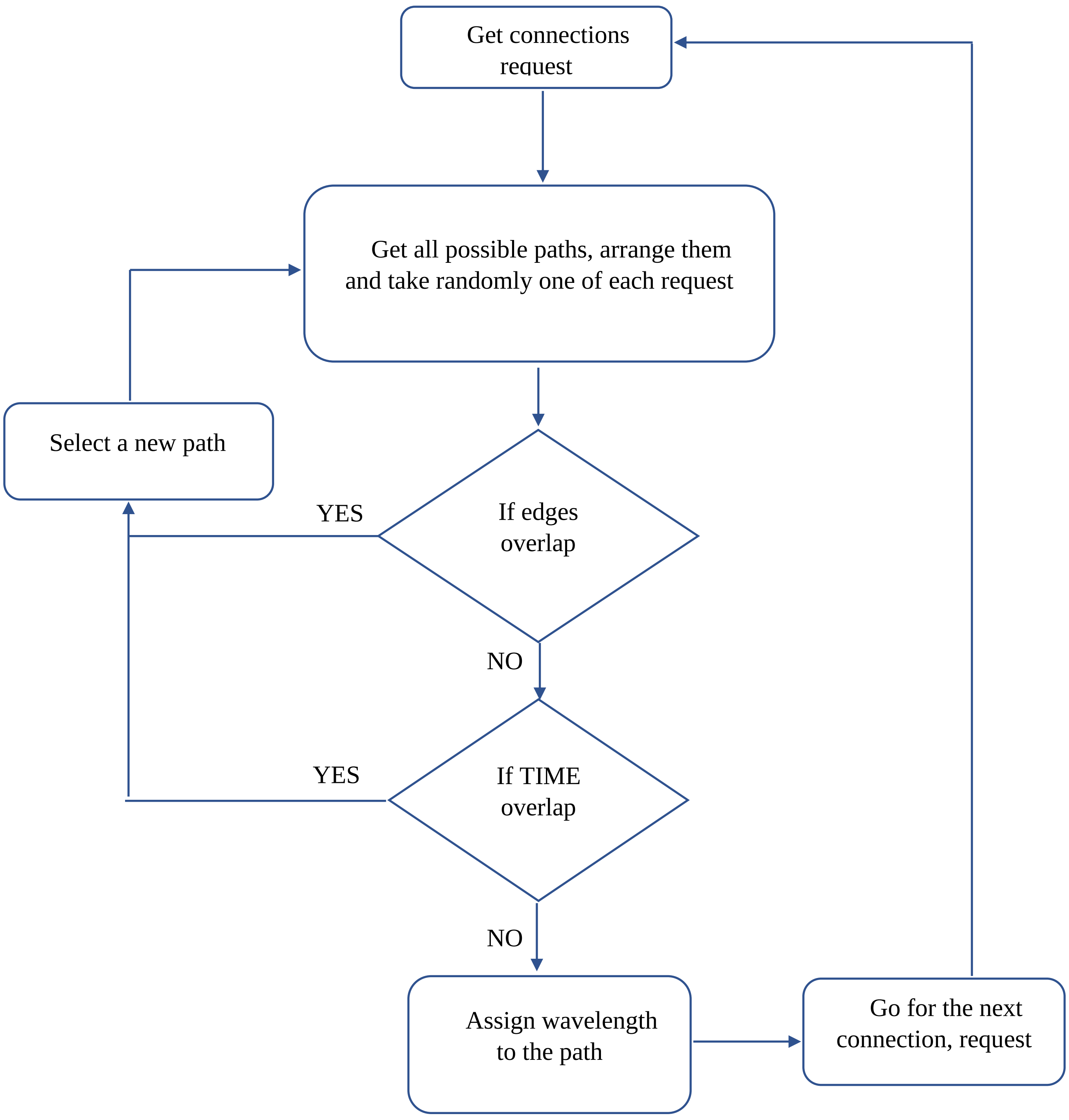

Get connections request
Get all possible paths, arrange them and take randomly one of each request
Select a new path
YES
If edges overlap
NO
YES
If TIME overlap
NO
Assign wavelength to the path
Go for the next connection, request

Figure 5

4. Graphical interfaces:

This picture shows how our application looks like when we run our Python code

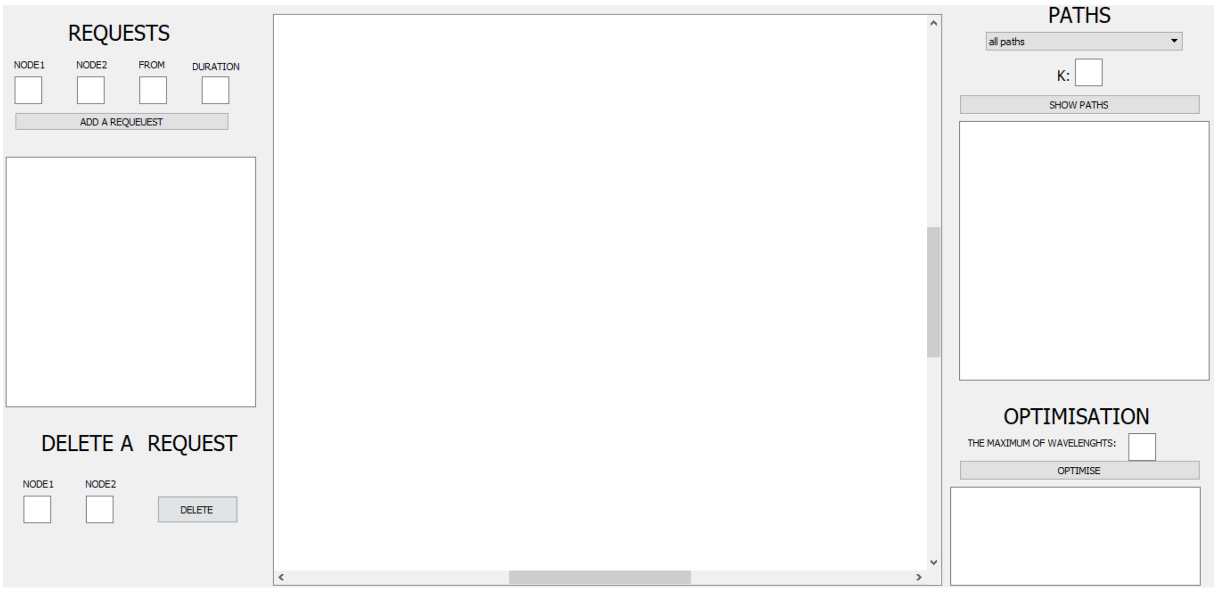
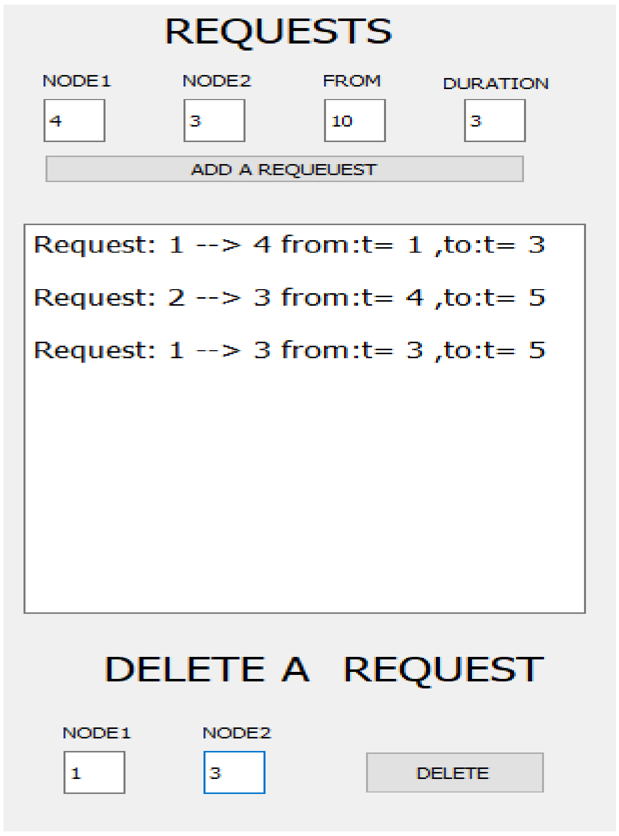


Figure 6

This window divided into three section:

4.1 The left section:

This section is used to enter or delete the connection requests



NODE1	NODE2	FROM	DURATION
4	3	10	3

```
Request: 1 --> 4 from:t= 1 ,to:t= 3
Request: 2 --> 3 from:t= 4 ,to:t= 5
Request: 1 --> 3 from:t= 3 ,to:t= 5
```

NODE1	NODE2
1	3

Figure 7

4.2 The second section:

This section is used to enter the graph manually with click left to insert a node and click right to select a node

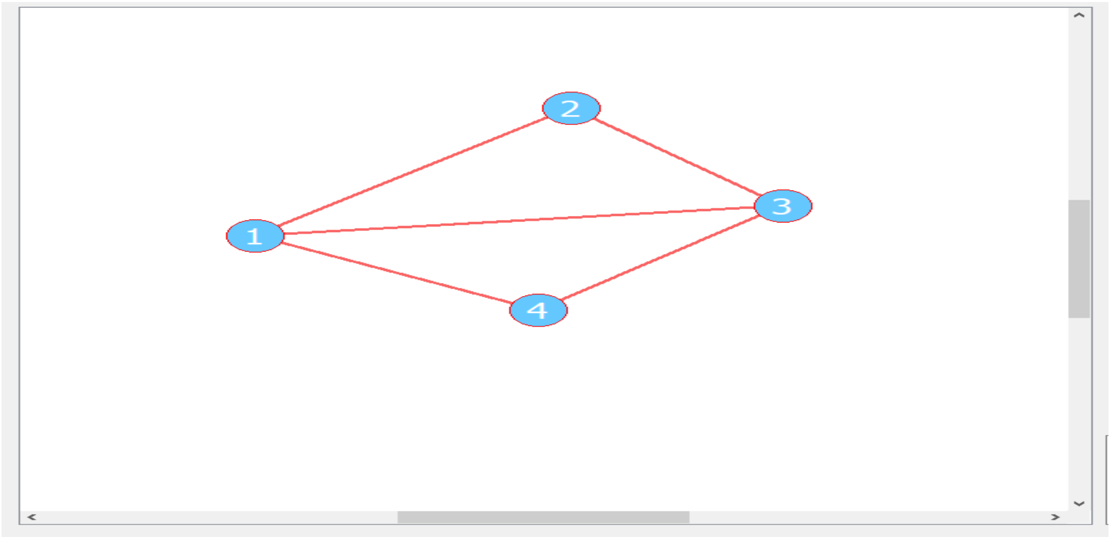
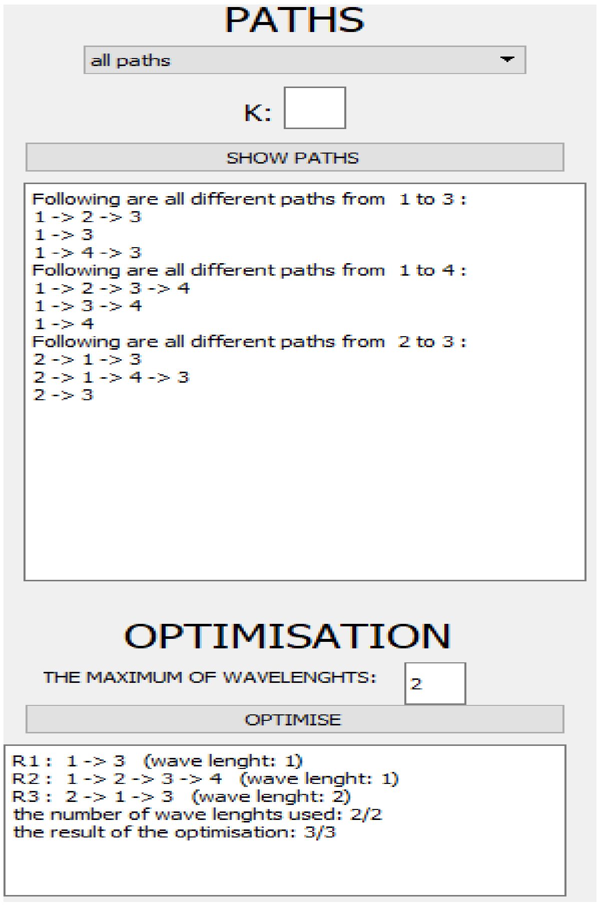


Figure 8

4.3 The third section:

This section is used to show the paths and assign a wavelength to a specific path



```
Following are all different paths from  1 to 3 :
1 -> 2 -> 3
1 -> 3
1 -> 4 -> 3
Following are all different paths from  1 to 4 :
1 -> 2 -> 3 -> 4
1 -> 3 -> 4
1 -> 4
Following are all different paths from  2 to 3 :
2 -> 1 -> 3
2 -> 1 -> 4 -> 3
2 -> 3
```

THE MAXIMUM OF WAVELENGHTS: 2

```
R1 :  1 -> 3  (wave lenght: 1)
R2 :  1 -> 2 -> 3 -> 4  (wave lenght: 1)
R3 :  2 -> 1 -> 3  (wave lenght: 2)
the number of wave lenghts used: 2/2
the result of the optimisation: 3/3
```

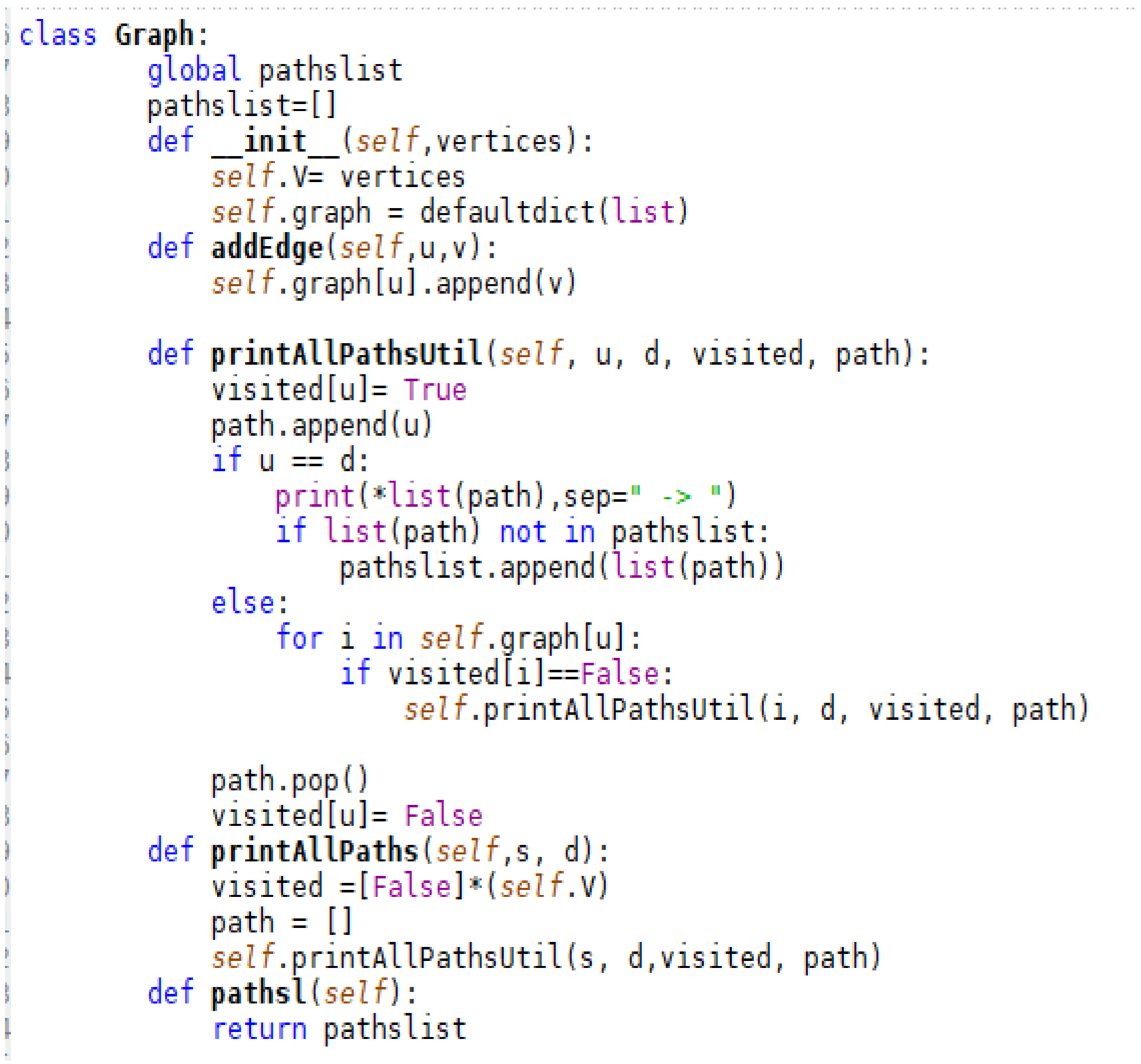
Figure 9

5. The code:

Our code contains a large number of lines so in the pictures bellow we have tried to split the code into small subcodes each one is made to accomplish a certain task so we will try to explain every single piece of code that is relevant to our project.

→ First, we begin by making a class named Graph

This code is very important it is first step to create the graph which is in our project considered to be the "raw material"



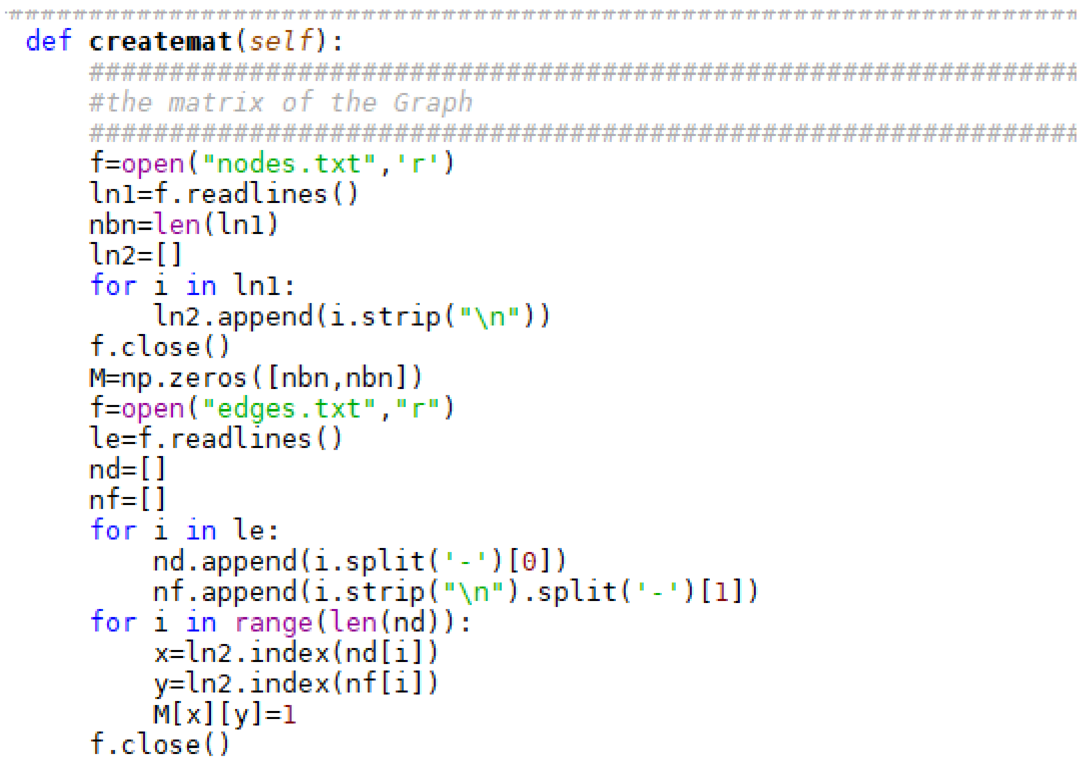
```
class Graph:
        global pathslist
        pathslist=[]
        def __init__(self,vertices):
            self.V= vertices
            self.graph = defaultdict(list)
        def addEdge(self,u,v):
            self.graph[u].append(v)

        def printAllPathsUtil(self, u, d, visited, path):
            visited[u]= True
            path.append(u)
            if u == d:
                print(*list(path),sep=" -> ")
                if list(path) not in pathslist:
                    pathslist.append(list(path))
            else:
                for i in self.graph[u]:
                    if visited[i]==False:
                        self.printAllPathsUtil(i, d, visited, path)

            path.pop()
            visited[u]= False
        def printAllPaths(self,s, d):
            visited =[False]*(self.V)
            path = []
            self.printAllPathsUtil(s, d,visited, path)
        def pathsl(self):
            return pathslist
```

Figure 10

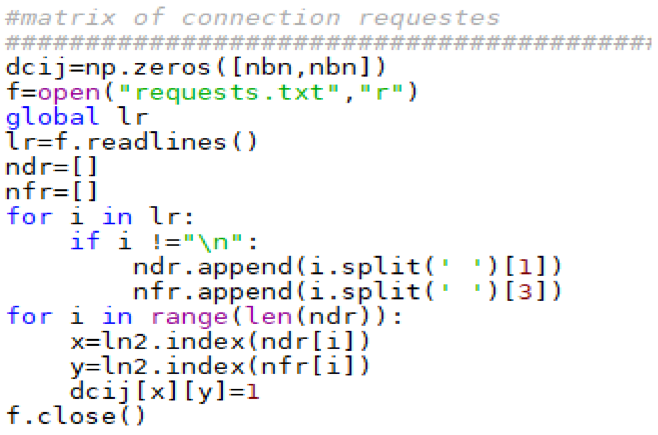
→ This is the function that create graph's matrix:



```
##############################################################################
 def createmat(self):
     ###############################################################
     #the matrix of the Graph
     ###############################################################
     f=open("nodes.txt",'r')
     ln1=f.readlines()
     nbn=len(ln1)
     ln2=[]
     for i in ln1:
         ln2.append(i.strip("\n"))
     f.close()
     M=np.zeros([nbn,nbn])
     f=open("edges.txt","r")
     le=f.readlines()
     nd=[]
     nf=[]
     for i in le:
         nd.append(i.split('-')[0])
         nf.append(i.strip("\n").split('-')[1])
     for i in range(len(nd)):
         x=ln2.index(nd[i])
         y=ln2.index(nf[i])
         M[x][y]=1
     f.close()
```

Figure 11

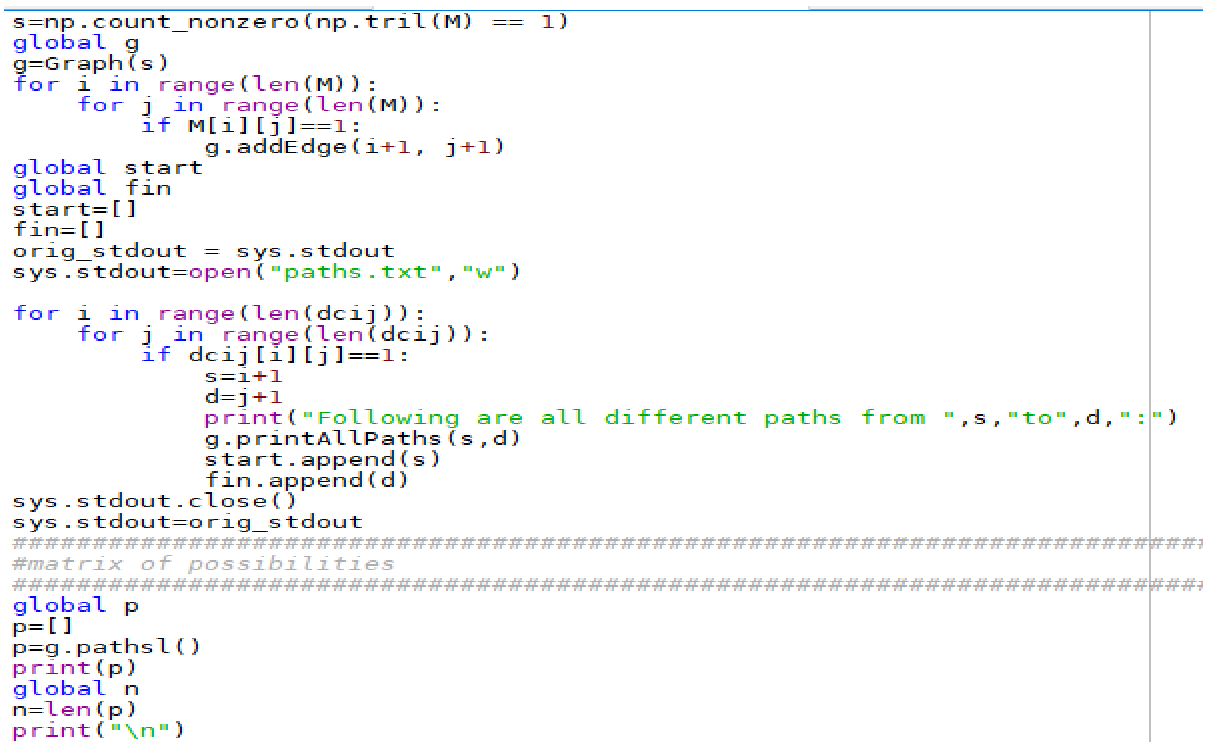
→ This one is the code that create the matrix of connections:



```
#matrix of connection requestes
########################################
dcij=np.zeros([nbn,nbn])
f=open("requests.txt","r")
global lr
lr=f.readlines()
ndr=[]
nfr=[]
for i in lr:
    if i !="\n":
        ndr.append(i.split(' ')[1])
        nfr.append(i.split(' ')[3])
for i in range(len(ndr)):
    x=ln2.index(ndr[i])
    y=ln2.index(nfr[i])
    dcij[x][y]=1
f.close()
```

Figure 12

→ And this code is for generating all path possibilities of each request and put them in a matrix:

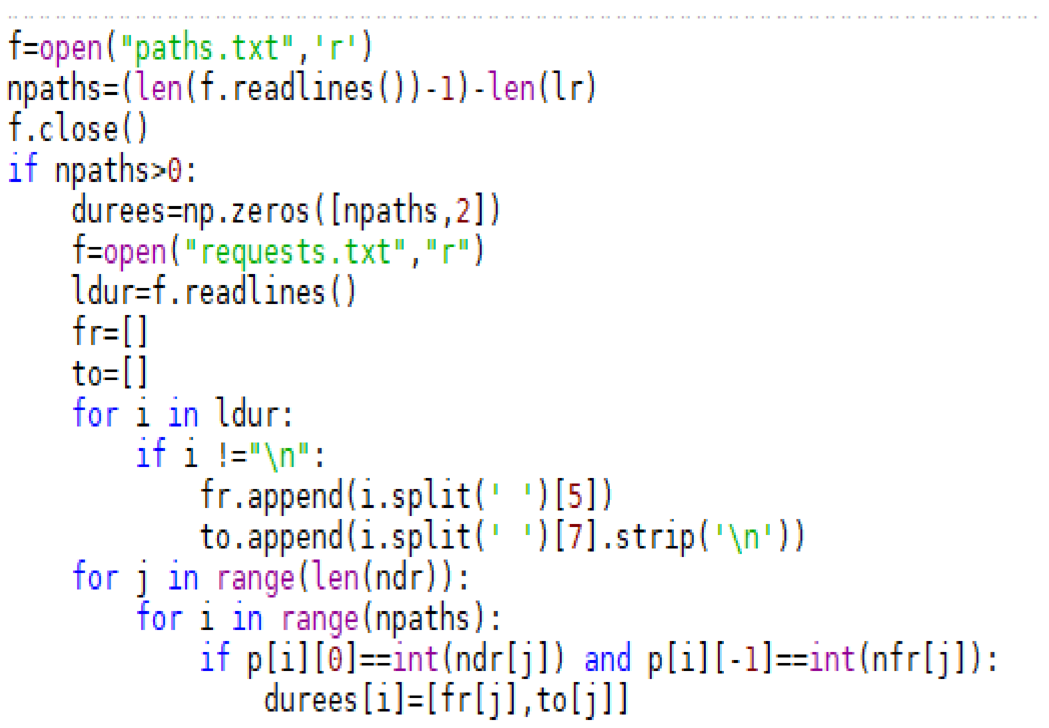


```
s=np.count_nonzero(np.tril(M) == 1)
global g
g=Graph(s)
for i in range(len(M)):
    for j in range(len(M)):
        if M[i][j]==1:
            g.addEdge(i+1, j+1)
global start
global fin
start=[]
fin=[]
orig_stdout = sys.stdout
sys.stdout=open("paths.txt","w")

for i in range(len(dcij)):
    for j in range(len(dcij)):
        if dcij[i][j]==1:
            s=i+1
            d=j+1
            print("Following are all different paths from ",s,"to",d,":")
            g.printAllPaths(s,d)
            start.append(s)
            fin.append(d)
sys.stdout.close()
sys.stdout=orig_stdout
###########################################################################
#matrix of possibilities
###########################################################################
global p
p=[]
p=g.pathsl()
print(p)
global n
n=len(p)
print("\n")
```

Figure 13

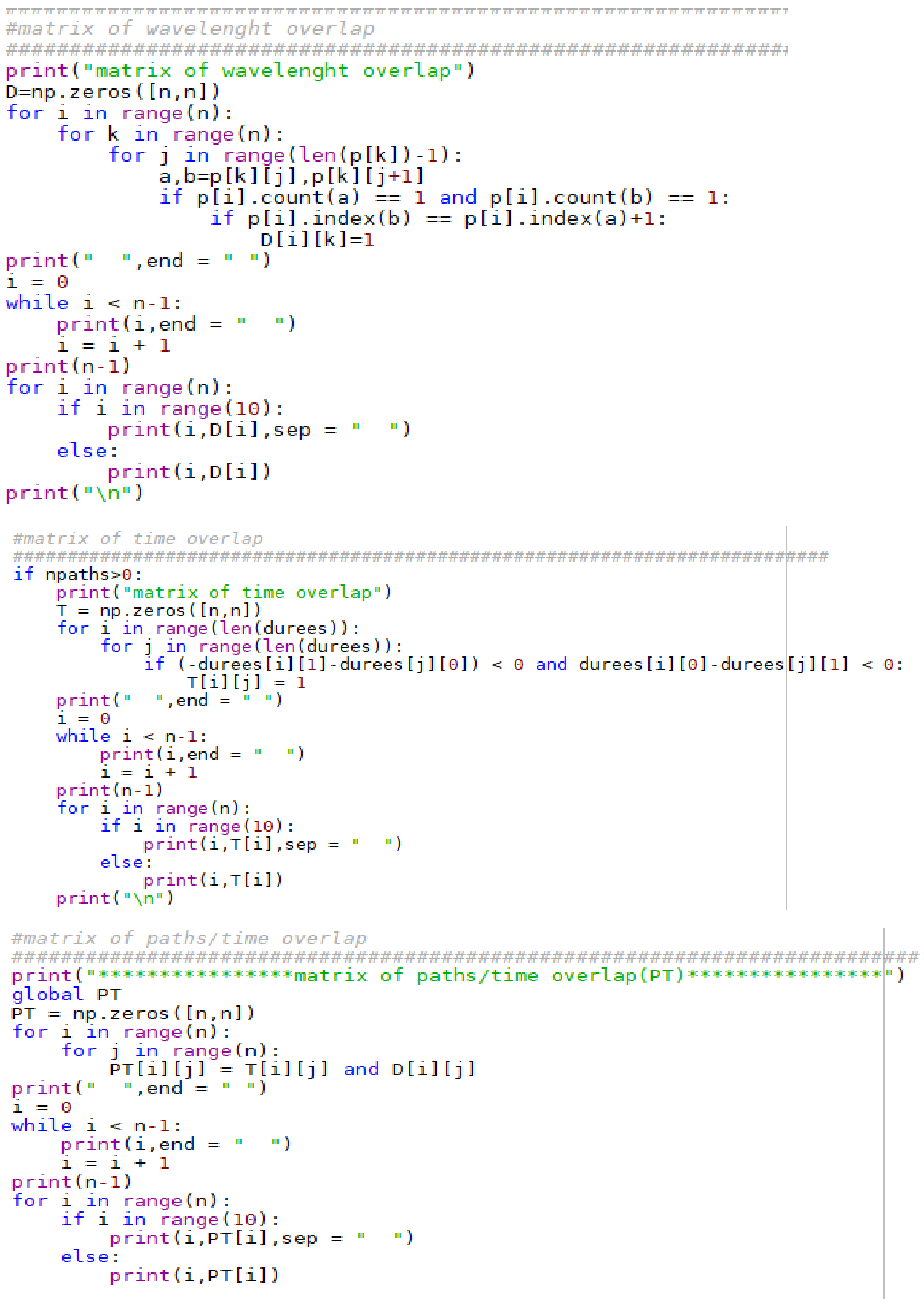
→ This code is for adding parameter of time



```
f=open("paths.txt",'r')
npaths=(len(f.readlines())-1)-len(lr)
f.close()
if npaths>0:
    durees=np.zeros([npaths,2])
    f=open("requests.txt","r")
    ldur=f.readlines()
    fr=[]
    to=[]
    for i in ldur:
        if i !="\n":
            fr.append(i.split(' ')[5])
            to.append(i.split(' ')[7].strip('\n'))
    for j in range(len(ndr)):
        for i in range(npaths):
            if p[i][0]==int(ndr[j]) and p[i][-1]==int(nfr[j]):
                durees[i]=[fr[j],to[j]]
```

Figure 14

→ This part of code is dedicated for making the matrix of edges and time overlap and the combination between them



```
###################################################################
#matrix of wavelenght overlap
###################################################################
print("matrix of wavelenght overlap")
D=np.zeros([n,n])
for i in range(n):
    for k in range(n):
        for j in range(len(p[k])-1):
            a,b=p[k][j],p[k][j+1]
            if p[i].count(a) == 1 and p[i].count(b) == 1:
                if p[i].index(b) == p[i].index(a)+1:
                    D[i][k]=1
print("  ",end = " ")
i = 0
while i < n-1:
    print(i,end = "  ")
    i = i + 1
print(n-1)
for i in range(n):
    if i in range(10):
        print(i,D[i],sep = "  ")
    else:
        print(i,D[i])
print("\n")
```

```
#matrix of time overlap
##############################################################################
if npaths>0:
    print("matrix of time overlap")
    T = np.zeros([n,n])
    for i in range(len(durees)):
        for j in range(len(durees)):
            if (-durees[i][1]-durees[j][0]) < 0 and durees[i][0]-durees[j][1] < 0:
                T[i][j] = 1
    print("  ",end = " ")
    i = 0
    while i < n-1:
        print(i,end = "  ")
        i = i + 1
    print(n-1)
    for i in range(n):
        if i in range(10):
            print(i,T[i],sep = "  ")
        else:
            print(i,T[i])
    print("\n")
```

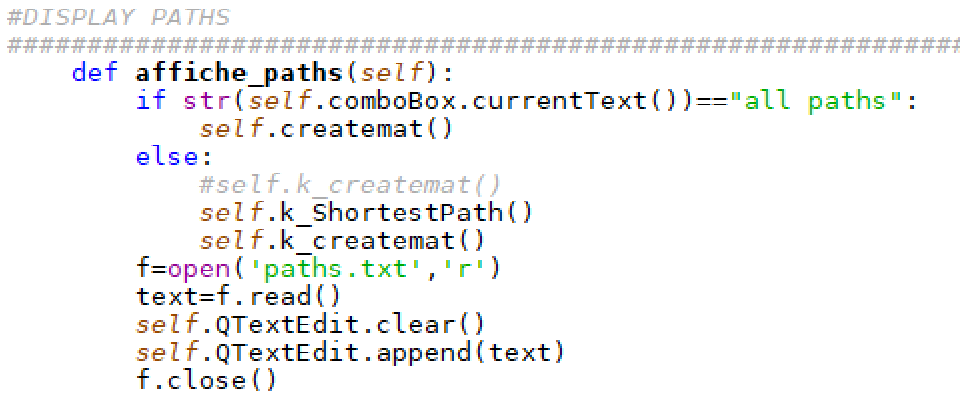
```
#matrix of paths/time overlap
################################################################################
print("****************matrix of paths/time overlap(PT)****************")
global PT
PT = np.zeros([n,n])
for i in range(n):
    for j in range(n):
        PT[i][j] = T[i][j] and D[i][j]
print("  ",end = " ")
i = 0
while i < n-1:
    print(i,end = "  ")
    i = i + 1
print(n-1)
for i in range(n):
    if i in range(10):
        print(i,PT[i],sep = "  ")
    else:
        print(i,PT[i])
```

Figure 15

In the source code we add subcodes to save the matrix of graph and requests in a separate file but in seek of brevity we won't show them in this report for further information please take a look at our source code

Now we move to the part of code that show how we made it possible to visualize the results

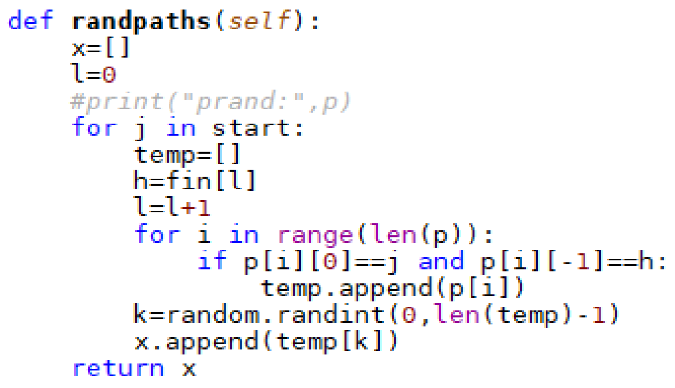
→ This function is for display all paths



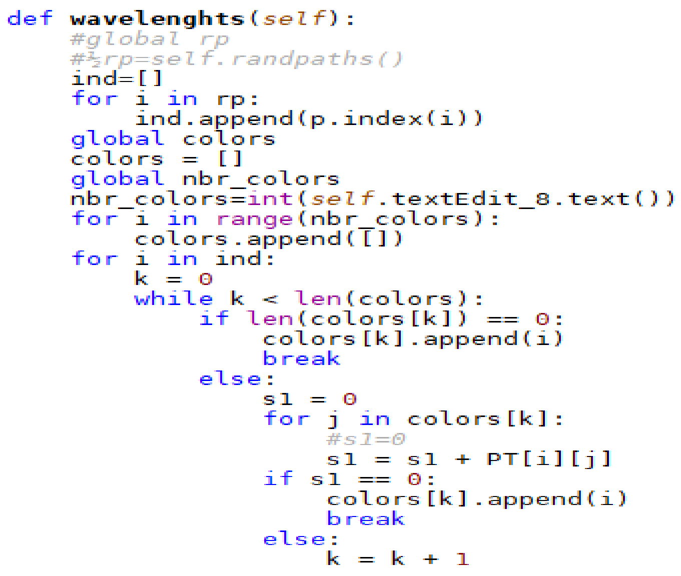
```
#DISPLAY PATHS
##############################################################
    def affiche_paths(self):
        if str(self.comboBox.currentText())=="all paths":
            self.createmat()
        else:
            #self.k_createmat()
            self.k_ShortestPath()
            self.k_createmat()
        f=open('paths.txt','r')
        text=f.read()
        self.QTextEdit.clear()
        self.QTextEdit.append(text)
        f.close()
```

Figure 16

→ And those two functions are for choosing random paths to make the tests



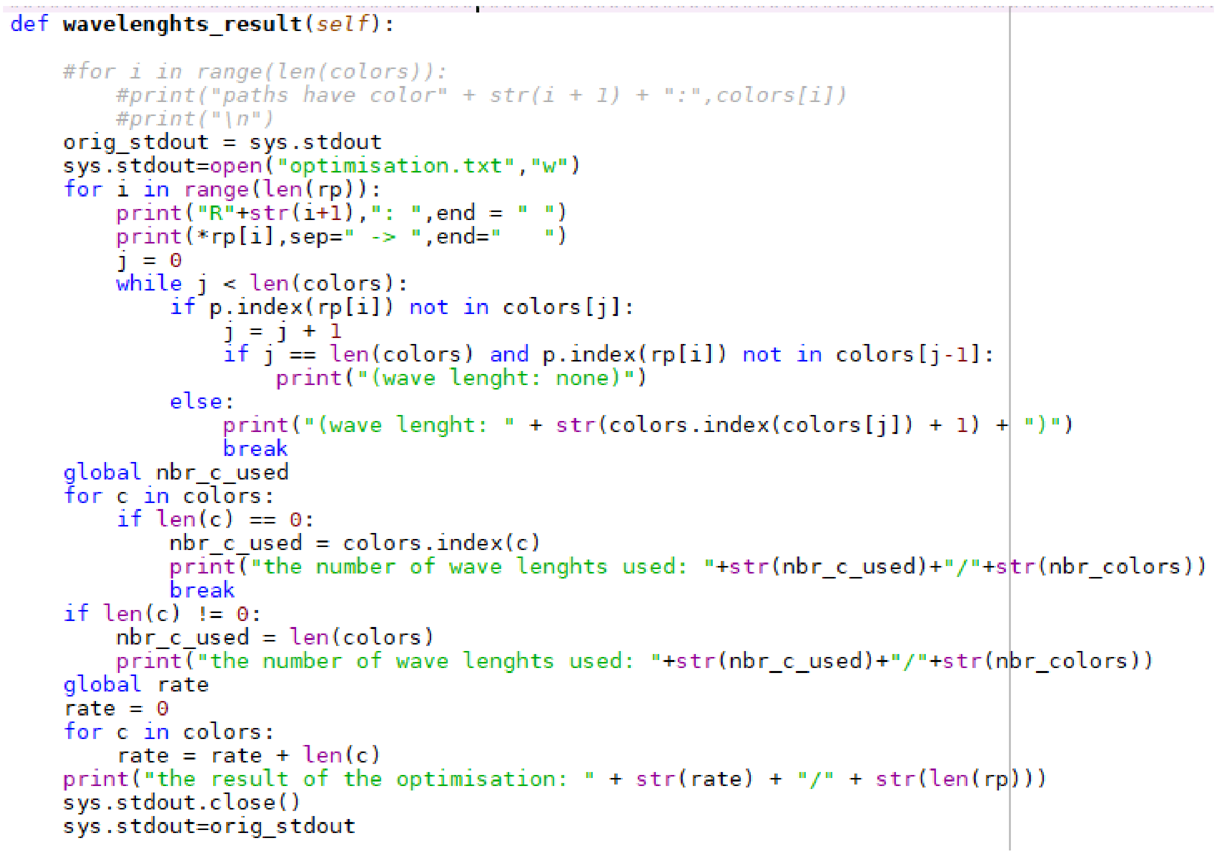
```
def randpaths(self):
    x=[]
    l=0
    #print("prand:",p)
    for j in start:
        temp=[]
        h=fin[l]
        l=l+1
        for i in range(len(p)):
            if p[i][0]==j and p[i][-1]==h:
                temp.append(p[i])
        k=random.randint(0,len(temp)-1)
        x.append(temp[k])
    return x
```



```
def wavelenghts(self):
    #global rp
    #½rp=self.randpaths()
    ind=[]
    for i in rp:
        ind.append(p.index(i))
    global colors
    colors = []
    global nbr_colors
    nbr_colors=int(self.textEdit_8.text())
    for i in range(nbr_colors):
        colors.append([])
    for i in ind:
        k = 0
        while k < len(colors):
            if len(colors[k]) == 0:
                colors[k].append(i)
                break
            else:
                s1 = 0
                for j in colors[k]:
                    #s1=0
                    s1 = s1 + PT[i][j]
                if s1 == 0:
                    colors[k].append(i)
                    break
                else:
                    k = k + 1
```

Figure 17

→ Now in order to give each path a colour we used this function

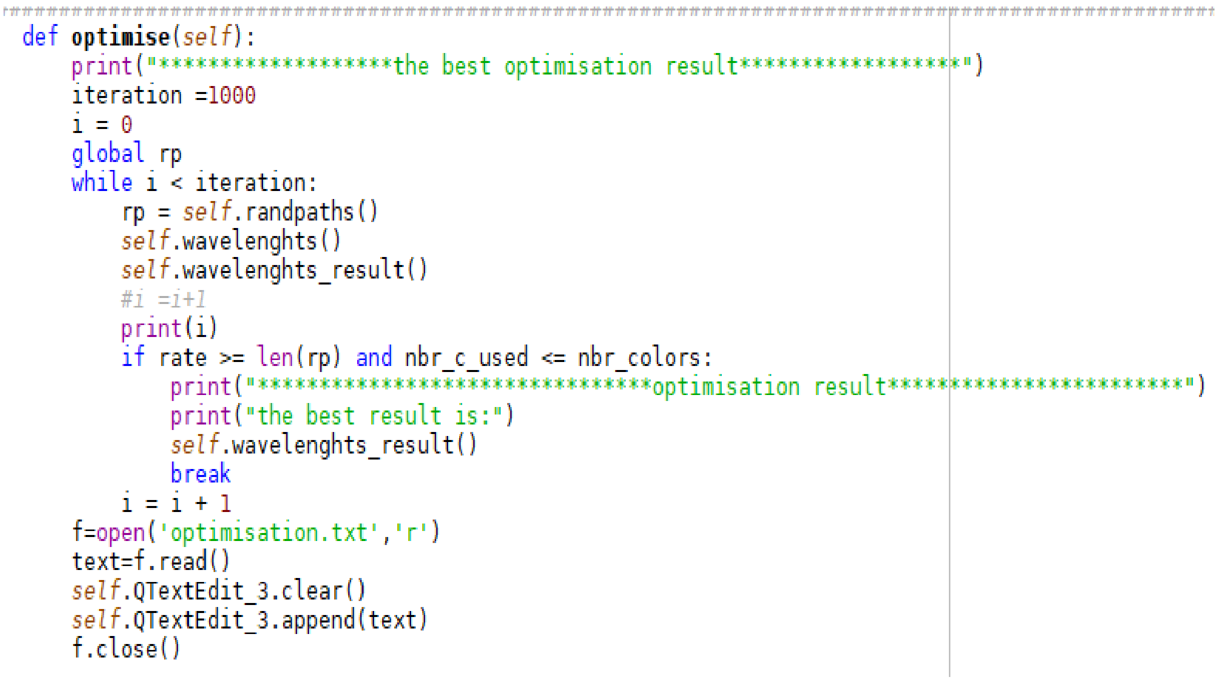


```
def wavelenghts_result(self):

    #for i in range(len(colors)):
        #print("paths have color" + str(i + 1) + ":",colors[i])
        #print("\n")
    orig_stdout = sys.stdout
    sys.stdout=open("optimisation.txt","w")
    for i in range(len(rp)):
        print("R"+str(i+1),": ",end = " ")
        print(*rp[i],sep=" -> ",end="    ")
        j = 0
        while j < len(colors):
            if p.index(rp[i]) not in colors[j]:
                j = j + 1
                if j == len(colors) and p.index(rp[i]) not in colors[j-1]:
                    print("(wave lenght: none)")
            else:
                print("(wave lenght: " + str(colors.index(colors[j]) + 1) + ")")
                break
    global nbr_c_used
    for c in colors:
        if len(c) == 0:
            nbr_c_used = colors.index(c)
            print("the number of wave lenghts used: "+str(nbr_c_used)+"/"+str(nbr_colors))
            break
    if len(c) != 0:
        nbr_c_used = len(colors)
        print("the number of wave lenghts used: "+str(nbr_c_used)+"/"+str(nbr_colors))
    global rate
    rate = 0
    for c in colors:
        rate = rate + len(c)
    print("the result of the optimisation: " + str(rate) + "/" + str(len(rp)))
    sys.stdout.close()
    sys.stdout=orig_stdout
```

Figure 18

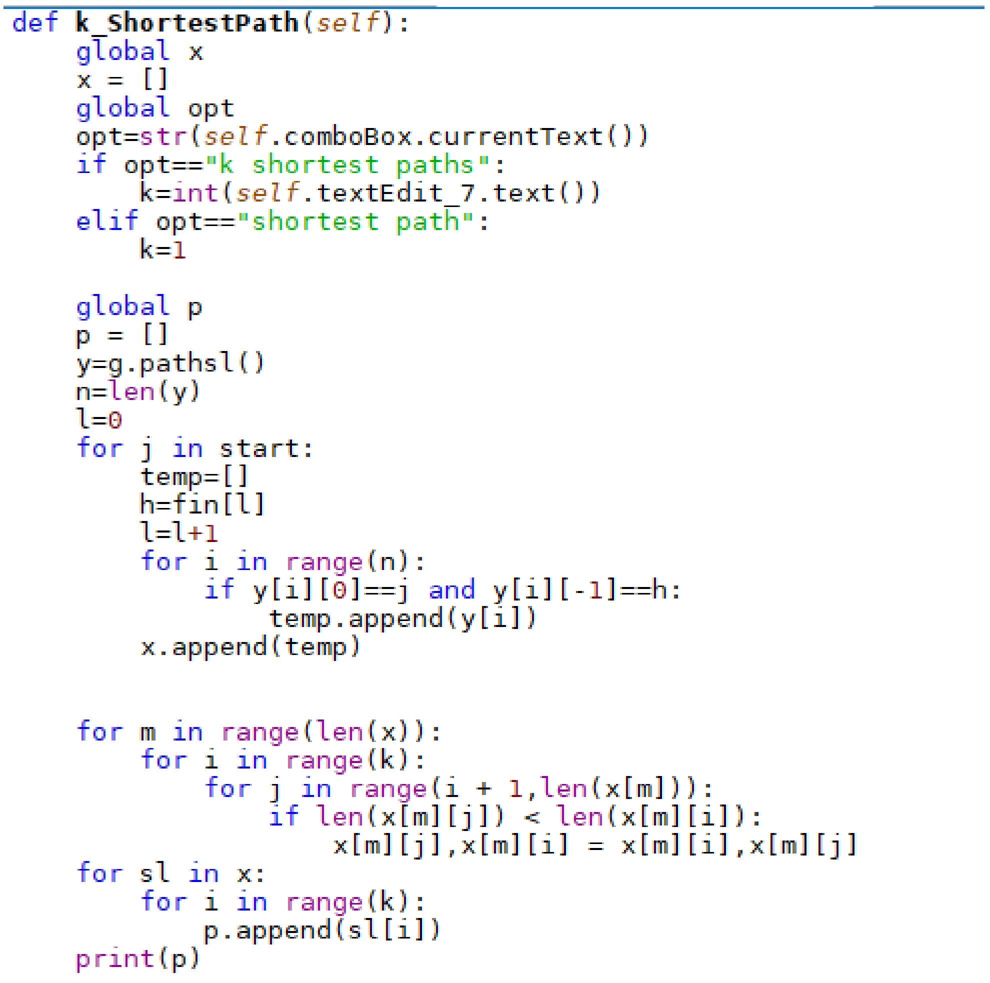
→ We get the optimised results by this function that use 1000 iterations before displaying the exact paths for each request



```
def optimise(self):
    print("********************the best optimisation result******************")
    iteration =1000
    i = 0
    global rp
    while i < iteration:
        rp = self.randpaths()
        self.wavelenghts()
        self.wavelenghts_result()
        #i =i+1
        print(i)
        if rate >= len(rp) and nbr_c_used <= nbr_colors:
            print("*********************************optimisation result***************************")
            print("the best result is:")
            self.wavelenghts_result()
            break
        i = i + 1
    f=open('optimisation.txt','r')
    text=f.read()
    self.QTextEdit_3.clear()
    self.QTextEdit_3.append(text)
    f.close()
```

Figure 19

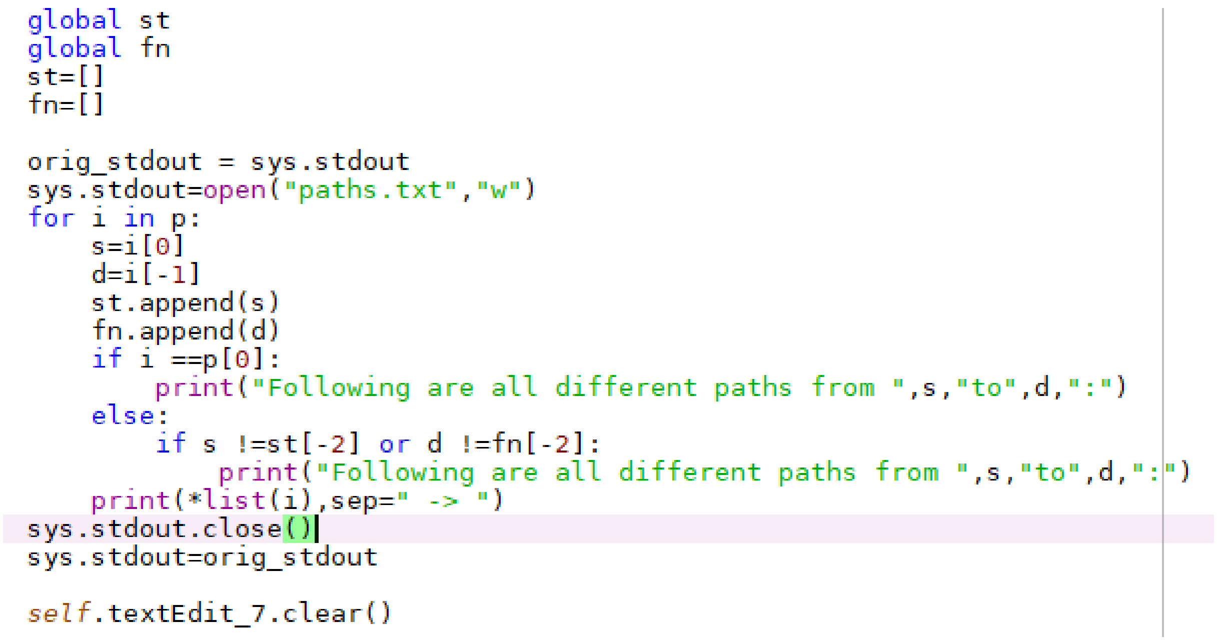
→This piece of code is for the option k_ShortestPath



```python
def k_ShortestPath(self):
    global x
    x = []
    global opt
    opt=str(self.comboBox.currentText())
    if opt=="k shortest paths":
        k=int(self.textEdit_7.text())
    elif opt=="shortest path":
        k=1

    global p
    p = []
    y=g.pathsl()
    n=len(y)
    l=0
    for j in start:
        temp=[]
        h=fin[l]
        l=l+1
        for i in range(n):
            if y[i][0]==j and y[i][-1]==h:
                temp.append(y[i])
        x.append(temp)


    for m in range(len(x)):
        for i in range(k):
            for j in range(i + 1,len(x[m])):
                if len(x[m][j]) < len(x[m][i]):
                    x[m][j],x[m][i] = x[m][i],x[m][j]
    for sl in x:
        for i in range(k):
            p.append(sl[i])
    print(p)
```



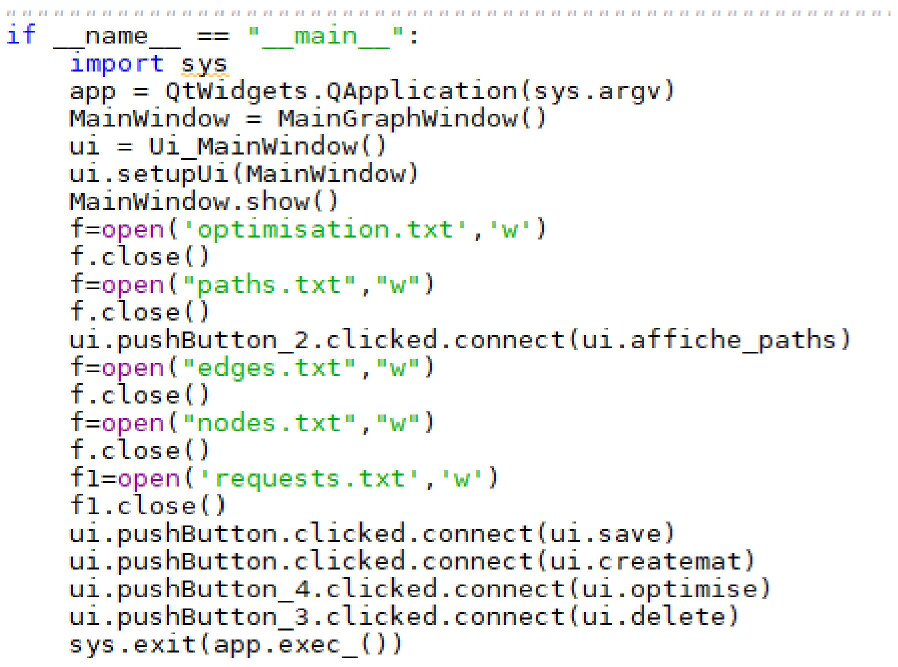
```python
global st
global fn
st=[]
fn=[]

orig_stdout = sys.stdout
sys.stdout=open("paths.txt","w")
for i in p:
    s=i[0]
    d=i[-1]
    st.append(s)
    fn.append(d)
    if i ==p[0]:
        print("Following are all different paths from ",s,"to",d,":")
    else:
        if s !=st[-2] or d !=fn[-2]:
            print("Following are all different paths from ",s,"to",d,":")
    print(*list(i),sep=" -> ")
sys.stdout.close()
sys.stdout=orig_stdout

self.textEdit_7.clear()
```

Figure 20

Finally, we should make a link between all the parts of our code and what we will have in the Graphical user interface



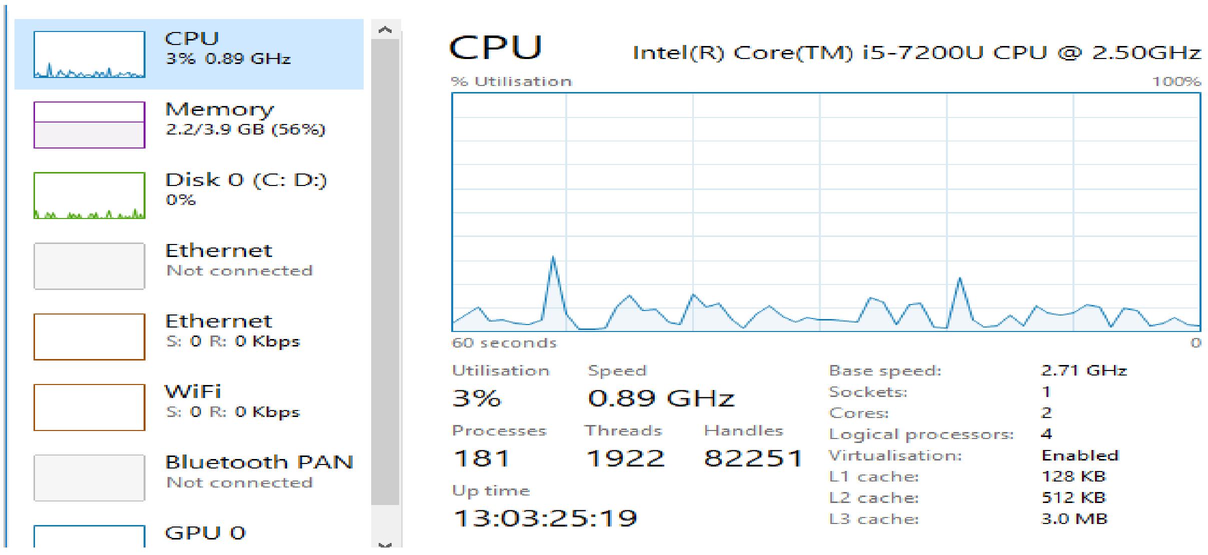
```
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = MainGraphWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    f=open('optimisation.txt','w')
    f.close()
    f=open("paths.txt","w")
    f.close()
    ui.pushButton_2.clicked.connect(ui.affiche_paths)
    f=open("edges.txt","w")
    f.close()
    f=open("nodes.txt","w")
    f.close()
    f1=open('requests.txt','w')
    f1.close()
    ui.pushButton.clicked.connect(ui.save)
    ui.pushButton.clicked.connect(ui.createmat)
    ui.pushButton_4.clicked.connect(ui.optimise)
    ui.pushButton_3.clicked.connect(ui.delete)
    sys.exit(app.exec_())
```

Figure 21

6. The Complexity of our application in terms of memory and consumed time:

6.1 Before launching the application

6.1.1 The CPU status

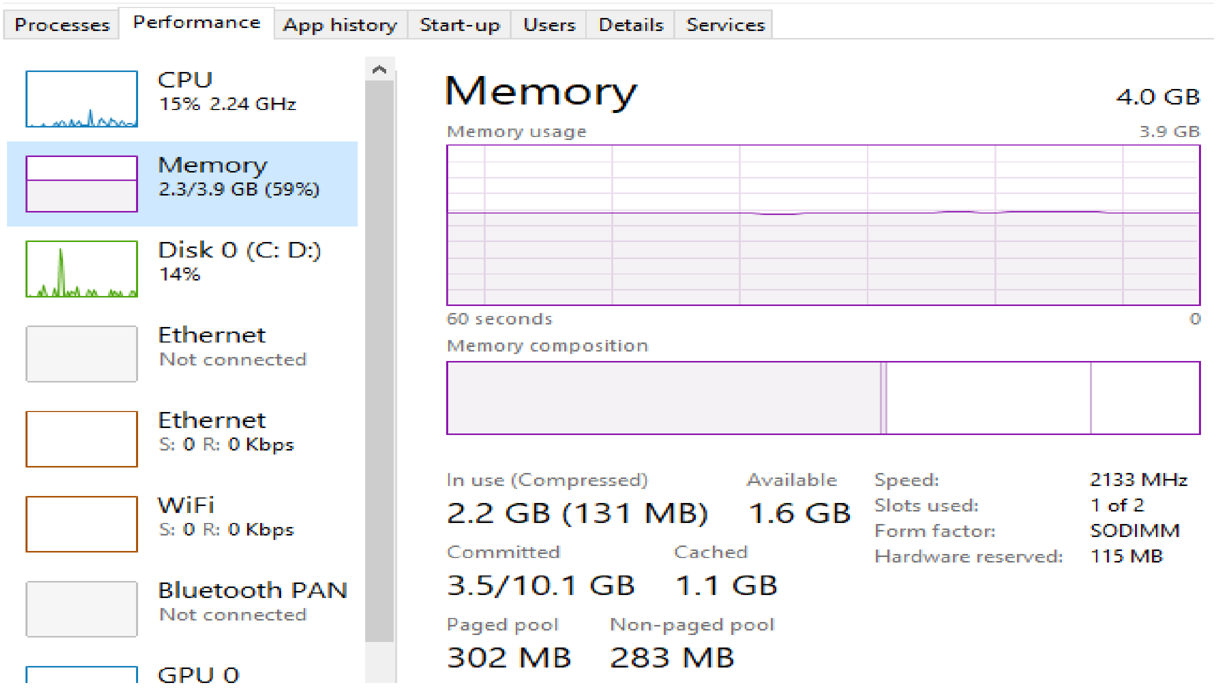


Utilisation	3%
Speed	0.89 GHz
Processes	181
Threads	1922
Handles	82251
Up time	13:03:25:19
Base speed:	2.71 GHz
Sockets:	1
Cores:	2
Logical processors:	4
Virtualisation:	Enabled
L1 cache:	128 KB
L2 cache:	512 KB
L3 cache:	3.0 MB

Figure 22

The 3% was reserved for basic system process

6.1.2 Memory status



Processes | Performance | App history | Start-up | Users | Details | Services

CPU 15% 2.24 GHz
Memory 2.3/3.9 GB (59%)
Disk 0 (C: D:) 14%
Ethernet Not connected
Ethernet S: 0 R: 0 Kbps
WiFi S: 0 R: 0 Kbps
Bluetooth PAN Not connected
GPU 0

Memory 4.0 GB
Memory usage 3.9 GB
60 seconds 0
Memory composition

In use (Compressed) 2.2 GB (131 MB)
Available 1.6 GB
Committed 3.5/10.1 GB
Cached 1.1 GB
Paged pool 302 MB
Non-paged pool 283 MB

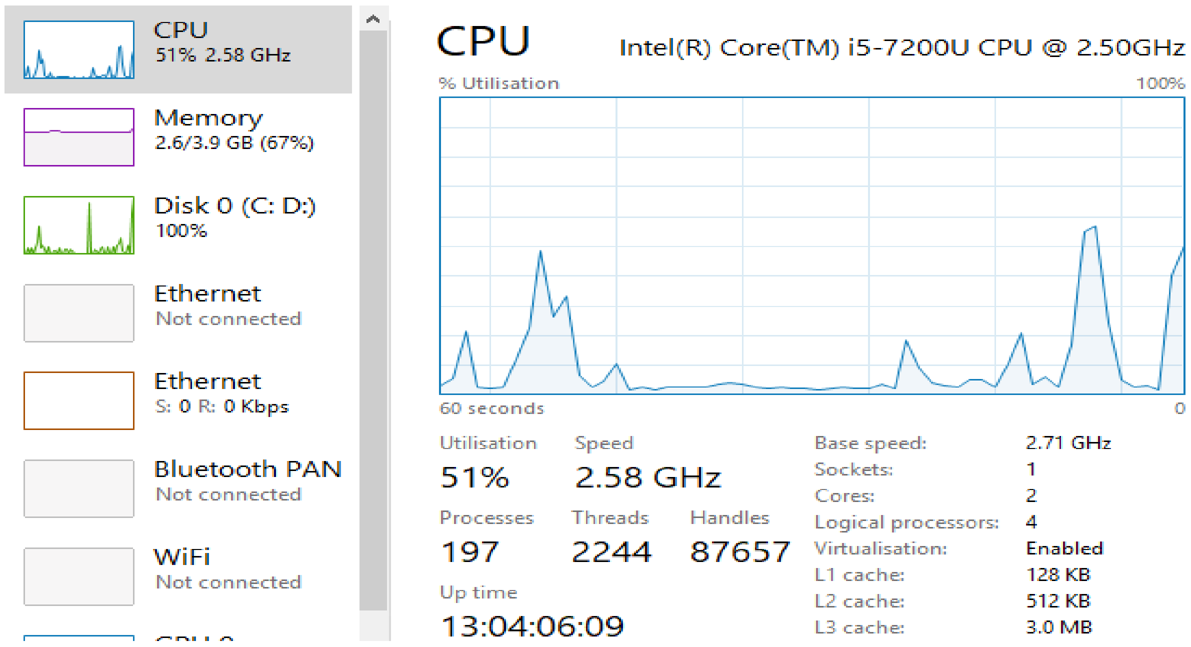
Speed:	2133 MHz
Slots used:	1 of 2
Form factor:	SODIMM
Hardware reserved:	115 MB

Figure 23

The 2.2 GB of the memory used only by windows and anti-virus program

6.2 After launching the application

6.2.1 The CPU status



CPU 51% 2.58 GHz
Memory 2.6/3.9 GB (67%)
Disk 0 (C: D:) 100%
Ethernet Not connected
Ethernet S: 0 R: 0 Kbps
Bluetooth PAN Not connected
WiFi Not connected

CPU Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
% Utilisation 100%
60 seconds 0

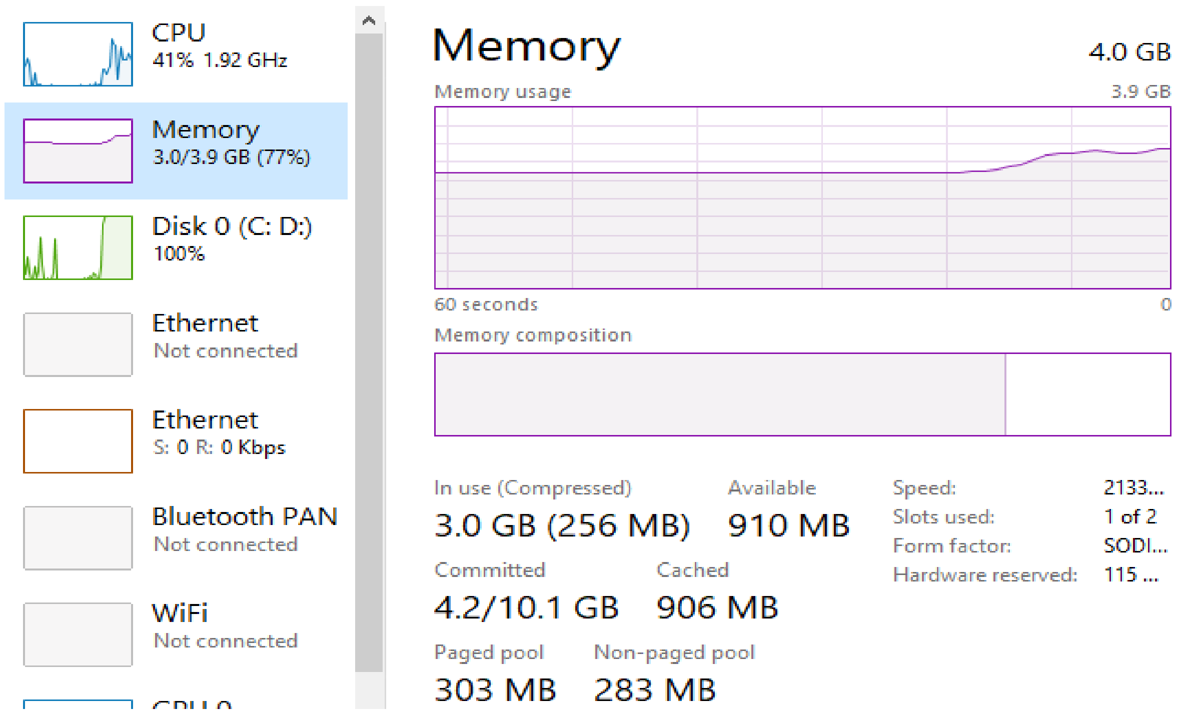
Utilisation 51%
Speed 2.58 GHz
Processes 197
Threads 2244
Handles 87657
Up time 13:04:06:09

Base speed:	2.71 GHz
Sockets:	1
Cores:	2
Logical processors:	4
Virtualisation:	Enabled
L1 cache:	128 KB
L2 cache:	512 KB
L3 cache:	3.0 MB

Figure 24

As we can see the processor status raises from 3% to 51%

6.2.2 Memory status



CPU
41% 1.92 GHz

Memory
3.0/3.9 GB (77%)

Disk 0 (C: D:)
100%

Ethernet
Not connected

Ethernet
S: 0 R: 0 Kbps

Bluetooth PAN
Not connected

WiFi
Not connected

Memory 4.0 GB

Memory usage 3.9 GB

60 seconds 0

Memory composition

In use (Compressed)
3.0 GB (256 MB)

Available
910 MB

Committed
4.2/10.1 GB

Cached
906 MB

Paged pool
303 MB

Non-paged pool
283 MB

Speed:	2133...
Slots used:	1 of 2
Form factor:	SODI...
Hardware reserved:	115 ...

Figure 25

The memory raised by 0.8 GB

7. Conclusion:

The routing and wavelength assignment RWA application gives us a simple and reliable way to find the paths representing each request and provides us with optimal results after verifying all conditions of shared edges and duration of each request

With the use of graphical interface, we can easily entre the graph and all our requests and choose the option of assignment of wavelength and with only two colors we can visualize the paths

In this project we learnt how to work in group, how to divide tasks and how to trust each other