UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

Diploma Thesis

DEVELOPMENT OF AN APPLICATION FOR INPUT/OUTPUT DATA PROCESSING AND VISUALIZATION FOR TRAVELLING SALESMAN PROBLEMS

by

PANAGIOTIS VRAKIDIS

Submitted to fulfill part of the requirements

for the degree of Diploma in Mechanical Engineering

2016

Approved by the Members of the Examination Committee:

First Examiner	Dr. Georgios K.D. Saharidis
(Supervisor)	Assistant Professor, Department of Mechanical Engineering, University of Thessaly
Second Examiner	Dr. Dimitris Pantelis
	Associate Professor, Department of Mechanical Engineering, University of Thessaly
Third Examiner	Dr. George Liberopoulos
	Professor, Department of Mechanical Engineering, University of Thessaly

Acknowledgements

First of all, I would like to thank my supervisor Assistant Professor Dr. Georgios K.D. Saharidis, for the ideal cooperation as well as his valuable guidance during my research for this thesis. I would also like to thank Dimitris Greasidis for the development of the Python code for the calculation of the shortest path distances between nodes. Furthermore, I would like to thank my colleagues Anastasios Orfanidis, Afroditi Temourtzidou and Dimitris Rizopoulos for their support and advice throughout the academic period.

Above all, I am grateful to my mother, Eleni Kappa, for all the sacrifices she has made in order for me to be able to follow my dreams and achieve my goals.

Panagiotis Vrakidis

DEVELOPMENT OF AN APPLICATION FOR INPUT/OUTPUT DATA PROCESSING AND VISUALIZATION FOR TRAVELLING SALESMAN PROBLEMS

PANAGIOTIS VRAKIDIS

University of Thessaly, Department of Mechanical Engineering, 2016

Supervisor: Dr. Georgios K.D. Saharidis, Assistant Professor in Operations Research Methods in Industrial Management

Summary

Transportation has always been important for our lives. Nowadays, there are numerous companies, which provide daily delivery services to people and have to organize their delivery strategy in advance, in order to maximize their profit as well as minimizing the total emissions for environmental purposes.

Finding the optimal tour, contributes to all the above and in this study we will form a complete application, which will enable the user to solve real-life travelling salesman problems. This will be able by only possessing the coordinates of all the nodes in the problem. At first, a Python code will be introduced that will calculate the shortest path distances between the nodes. Furthermore, a TSP algorithm will be constructed in the language of *C*++ and then simulated with the utilization of CPLEX Optimization Studio. Finally, a procedure considering the visualization of the results on the OpenStreetMap through the QGIS platform will be analyzed.

Table of Contents

List of Tables

List of Equations

List of Figures

Acronyms

API	Application Programming Interface
CSV	Comma Separated Variable
GPS	Global Positioning System
JSON	JavaScript Object Notation
NP	Non-deterministic Polynomial-time
OSM	OpenStreetMap
QGIS	Quantum Geographic Information System
UK	United Kingdom
UTF	Universal Transformation Format
WGS	World Geodetic System
XML	Extensible Markup Language

Chapter 1: Vehicle Routing and the Travelling Salesman Problem

1.1 Definition of the problem

The Travelling Salesman Problem (TSP) is the most common problem in the field of Operational Research. However, due to the fact that it is considered to be groundwork for vehicle routing, the approach of this problem can be very challenging. In order to understand the nature of this problem we should first get acquainted with its data. A graph, *G*, that consists of a set of vertices, *V*, as well as a set of edges, *E*, is necessary in order for the problem to have substance. *Cij* is the result of the vehicle travelling along arc (*i,j*) and refers to the cost of the route. Given the above information, a TSP's solution should return the Hamiltonian Cycle of *G* with the minimum total cost (1). By mentioning the term Hamiltonian Cycle, we refer to the cycle that is created when each node in a graph is visited exactly once. To sum it up, the goal is to obtain the optimal tour for each problem.

1.2 Historical facts

The first recorded mathematical formulation of the TSP dates back in the 1800s and was accomplished by the Irish mathematician W.R. Hamilton and the British mathematician Thomas Kirkman. As far as the general form of the travelling salesman problem is concerned, it was studied in the 1930s in Vienna, by mathematician Karl Menger and his colleagues (2). The name travelling salesman problem was introduced by Hassler Whitney at Princeton and was officially published in the 1940s. In addition, during the 1970s Padberg and Rinaldi managed to solve examples with up to 2392 cities, using cutting planes and branch and bound. The above was considered to be great progress, but as the research proceeded, we are now able to confront instances with millions of cities that produce a very small optimality gap in the solution-tour.

1.3 Complexity

The travelling salesman problem has been proven to be NP hard, which means that it cannot be solved using Linear Programming Techniques, in polynomial time. Such complexity strengthened the need for approximation algorithms that would be able to produce feasible solutions very close to the optimal one. Branch and bound is a very efficient method in order to deal with the TSP, not to mention the many more numerical methods that have emerged and provide quality solutions.

1.4 Distinctive cases

Numerous restrictions can be applied on the general form of the TSP in order to make it reflect more realistic applications. The most common variations of this type of problems are based on some simple restrictions and are mentioned below:

- Metric TSP: In this case, also known as delta TSP, the edge costs are symmetric and satisfy the triangle inequality. This means that if we have 3 nodes *a*, *b* and *c* the cost for going over from *a* to *c* is lower than visiting node *c* through *b*. It should also be mentioned that the edge costs are calculated by measuring the metric distances between the nodes,

- Euclidean TSP: It is a particular case of the metric TSP due to the fact that the distances in a plane obey the triangle inequality. Moreover, the vertices correspond to points in a d-dimensional space while the cost function is the Euclidean distance. As the input numbers are integers, comparing lengths of tours involves comparing sums of square roots,

- Symmetric TSP: Here, the distance between two nodes in the network of our problem is the same in both directions. Someone can easily realize that the above, results in the cost remaining the same also. When this is not the case and the distance between two points differs according to direction, then we have an asymmetric TSP.

1.5 Applications

The travelling salesman problem has a wide range of real life applications with the contributions in vehicle routing to be the most popular. For instance, the assignment of customers to certain trucks that offer delivery services and to create a delivery strategy for each truck is a case that could be solvable as a TSP. In order however to approach these kind of problems as TSPs, the fleet available should have a fixed number of vehicles and there should not exist time or capacity constraints.

In addition, the order and picking problem in warehouses can also be considered as a TSP. This is very clear to realize, as a vehicle has to collect certain subsets of items which are stored and then ship them to the customer who made the order. The storage locations of the items correspond to the nodes of the graph and the distance between two nodes is given by the time which is required in order to move from one location to the other. As a result, finding a shortest route for the vehicle with minimum pickup time can be solved as a TSP (3).

It is worth mentioning that not only transportation issues are being considered as TSPs. The drilling problem of printed circuit boards is such an example. The ultimate goal is to minimize the travel time of the machine's head. The key thought is to treat the time it takes the drilling head to move from one position to the other like the distance between to cities. Furthermore, computer wiring, x-ray crystallography, overhauling gas turbine engines and mask plotting in printed circuit boards' production are a few more examples that can be dealt with as TSPs.

Chapter 2: Shortest Path using Google's API

2.1 Google API for shortest path distances

2.1.1 Distances

For anyone to be able to solve the travelling salesman problem, the shortest path distances for every possible route are necessary. When someone possesses the above, then he can achieve the ultimate goal, which is no other than the minimization of the total travelling distance and as a result the minimization of the total travelling cost. In order to proceed with the solution process we must first find a way of calculating the shortest path distances and google contributes in that purpose.

2.1.2 Google maps distance matrix API

Google offers a service that allows each user to get the shortest path distances of the nodes that exist in their problem. This service in order to be available requires from each user to download the distance matrix API key from the following link:

https://developers.google.com/maps/documentation/distance-matrix/

However, there are some limitations considering the allowed elements of the matrix produced, which are mentioned below:

- 2500 free elements per day,
- 100 elements per second.

The above limitations occur from the standard usage of this API which is free for everyone. Once these steps are completed, we will take advantage of the availability of this API in the Python client and create a code that will send the proper query in order for the distance matrix to be produced.

2.2 Python

2.2.1 Functionality of code

The code that will be presented in the next section was created using the version 2.7.0 of Python along with its setup tools. This code reads a .txt file with the selected nodes of the problem and then creates the links from google maps that correspond to the nodes. Once the links are created, it then reads them from a .txt file in order to calculate the shortest path distances as well as filling a .txt file with the elements produced. To accomplish the above mentioned, one must include in the file named “Python27” which lies in the hard disk (*C*:), not only the Python code, but also the .txt file with the coordinates of the selected nodes. When everything is set, the code runs by opening the command prompt in “Python27”. At first, the command “easy_install simplejson” should be executed and after that, the name of the Python code should be typed in the command prompt. After pressing Enter, it is only a matter of seconds for the matrix to be produced.

2.2.2 Python code

The code written in Python is presented below:

```python
import simplejson, urllib
import time
#from urllib.request import urlopen

#file = open('RESULTS.DAT','r')
file2 = open("DISTANCES.DAT", "w")
file3 = open("DURATIONS.DAT","w")
file4=open('DATA.DAT','r')
file5=open('LINKS.DAT','w')

def file_len(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
    return (i+1)


def ori(first,last):
  file5.write ("https://maps.googleapis.com/maps/api/distancematrix/json?uni
ts=metric&origins=")
  file5.write (str(DATAX[first])+","+str(DATAY[first]))
  for i in range(first+1,last):
    file5.write ("|"+str(DATAX[i])+","+str(DATAY[i]))
  file5.write ("&destinations=")

def dest(first,last):
  file5.write (str(DATAX[first])+","+str(DATAY[first]))
  for i in range(first+1,last):
```

```
  file5.write ("|"+str(DATAX[i])+","+str(DATAY[i]))
 file5.write ("&key=AIzaSyA0-pYqwOleHFQALXR4x8WtfM0cNn0Ywfk")
 file5.write ('\n')


def geocode(adress,ori,ele,a,b,distan,durati):

  url = adress
  result = simplejson.load(urllib.urlopen(url))
  for i in range(0,ori):
    for x in range(0,ele):
     dist = result['rows'][i]['elements'][x]['distance']
     dura = result['rows'][i]['elements'][x]['duration']
     "print (dist['text'])"
     distan[10*a+i][10*b+x]=dist['text']
     durati[10*a+i][10*b+x]=dura['text']
     "print (" ")"
     "print dura['text']"
     "print (" ")"
     "print (" ")"
     "file2.write (dist['text'])"
     "file2.write (",")"
     " file3.write (dura['text'])"
     "file3.write (",")"
    "file2.write('\n')"
    "file3.write('\n')"
  #print "value is : " + str(dist['value'])

DATAX=[]
DATAY=[]



for line in file4:
 x=line
 y=x[9:17]
 DATAY.append(float(y))
 DATAX.append(float(x[0:8]))

n=file_len('DATA.DAT')

distan=[["0" for x in range(n)] for x in range(n)]
durati=[["0" for x in range(n)] for x in range(n)]

for x in range(5):

 if (x==0):
  for f in range(0,5):
   if f==0:
    ori(0,10)
    dest(0,10)
   elif f==1:
    ori(0,10)
    dest(10,20)
   elif f==2:
    ori(0,10)
    dest(20,30)
   elif f==3:
    ori(0,10)
    dest(30,40)
   elif f==4:
    ori(0,10)
    dest(40,n)
  #file1.write ('\n')
 elif x==1:
    for f in range(0,5):
      if f==0:
        ori(10,20)
        dest(0,10)
      elif f==1:
        ori(10,20)
        dest(10,20)
            elif f==2:
              ori(10,20)
              dest(20,30)
            elif f==3:
              ori(10,20)
              dest(30,40)
            elif f==4:
              ori(10,20)
              dest(40,n)
          #file1.write ('\n')
        elif x==2:
          for f in range(0,5):
            if f==0:
              ori(20,30)
              dest(0,10)
            elif f==1:
              ori(20,30)
              dest(10,20)
            elif f==2:
              ori(20,30)
              dest(20,30)
            elif f==3:
              ori(20,30)
              dest(30,40)
            elif f==4:
              ori(20,30)
              dest(40,n)
          #file1.write ('\n')
        elif x==3:
          for f in range(0,5):
            if f==0:
              ori(30,40)
              dest(0,10)
            elif f==1:
              ori(30,40)
              dest(10,20)
            elif f==2:
              ori(30,40)
              dest(20,30)
            elif f==3:
              ori(30,40)
              dest(30,40)
            elif f==4:
              ori(30,40)
              dest(40,n)
          #file1.write ('\n')
        elif x==4:
          for f in range(0,5):
            if f==0:
              ori(40,n)
              dest(0,10)
            elif f==1:
              ori(40,n)
              dest(10,20)
            elif f==2:
              ori(40,n)
              dest(20,30)
            elif f==3:
```

```
    ori(40,n)
    dest(30,40)
   elif f==4:
    ori(40,n)
    dest(40,n)
  #file1.write ('\n')
file5.close

file5=open('LINKS.DAT','r')



y=file_len('LINKS.DAT')

links=[]
for x in range(y):
 links.append(str(file5.readline()))

a=0
b=0

if __name__ == '__main__':
  for i in range(1,y+1):
   time.sleep(2)
   if (i%5==0 and i!=y):
    geocode(links[i-1],10,(n-40),a,b,distan,durati)
   elif (i>20 and i<y):
    geocode(links[i-1],(n-40),10,a,b,distan,durati)
   elif (i==y):
    geocode(links[i-1],(n-40),(n-40),a,b,distan,durati)
   else:
    geocode(links[i-1],10,10,a,b,distan,durati)
   a=int(i/5)
   b=i%5
   print (" ")

for x in range(n):
 for i in range(n):
  file2.write (distan[x][i])
  file2.write (",")
  file3.write (durati[x][i])
  file3.write (",")
 file2.write('\n')
 file3.write('\n')
```

It should be highlighted that the code above runs for 50 nodes and that the inserted coordinates must have strictly a five number decimal place.

Chapter 3: Mathematical Model

3.1 Problem description

The TSP model that will be formed, concerns a real-life instance of a courier company named “Geniki Taxidromiki”. The facility of the company that the problem involves is situated at 56 Lambraki str. in the city of Volos. The goal is to structure a mathematical model able to minimize the total travelling distance covered by the company’s van in order to satisfy the daily demand.

3.2 The TSP mathematical model

The model that will be presented is a mixed integer-linear programming one, due to the decision variables that it includes. Prior to the equations of the formulation, all the other essential parameters will also be mentioned. By referring to the term parameters, all the indexes, data and sets as well as the decision variables that this model is based on will be defined.

Indexes

i, j: the nodes of the problem’s network.

Data

n: the number of customers.

m: starting point.

D_{ij}: the distance in kilometers between nodes i and j, where i, $j = 1,\dots, n+m$. Therefore, D_{ij} represents the distances from customer to customer and from the starting point to customers.

Sets

$[1,\dots, n+m]$: the set of nodes including all customers as well as the starting point.

$[1,\dots, n]$: the set of customers.

$(n+m-1, n+\mathrm{m}]$: the set of the starting point.

Decision Variables

X_{ij}: this is a binary variable which indicates whether the vehicle travels along arc (i,j) or not. When $X_{ij} = 1$, then the vehicle covers the distance from node i to node j, otherwise $X_{ij} = 0$. Where $i, j = 1,\dots, n+m$.

u_i: non-negative integer variable, which indicates the turn that a certain node is visited. This variable is needed in order to structure the sub-tour elimination constraints, which play a vital role to the solution's feasibility (4). Where $i = 1,\dots, n+m$.

Mathematical Formulation

The following constraints combined with the objective function, form a complete model of the travelling salesman problem.

- Constraints:

$$\sum_{i=1}^{n+m} X_{ij} = 1 \qquad \forall\ j = 1,..,n \quad (1)$$

$$\sum_{j=1}^{n+m} X_{ij} = 1 \qquad \forall\ i = 1,..,n \quad (2)$$

$$\sum_{j=1}^{n} X_{ij} = 1 \qquad \forall\ i = n+m \quad (3)$$

$$\sum_{i=1}^{n} X_{ij} = 1 \qquad \forall j = n+m \quad (4)$$

$$u_i - u_j + 1 \le (n+m-1)\left(1 - X_{ij}\right) \qquad \forall\, i,j \ne n+m \quad (5)$$

$$u_{n+m} = 1 \qquad (5.1)$$

$$2 \le u_i \le n+m \qquad \forall\, i \ne n+m \quad (5.2)$$

$$X_{ii} = 0 \qquad i = 1, \dots, n+m \quad (6)$$

$$X_{ij} \in \{0,1\} \qquad i,j = 1, \dots, n+m \quad (7)$$

$$u_i \in Z^+ \qquad i = 1, \dots, n+m \quad (8)$$

- Objective Function:

$$Minimize\ z = \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} D_{ij} X_{ij} \qquad (9)$$

Explanation of constraints and objective function of TSP model

Constraints (1) ensure that each customer is visited exactly once. This gets clearer if the approach is strictly mathematical, as the equation formed allows only one *Xij* to get the value 1 for each *j* (customers).

Constraints (2) show that from every node of a customer, the vehicle will depart for another node. This is very crucial in order to get a feasible solution due to the fact that they act as flow conservation constraints.

Constraints (3) indicate that the vehicle will serve exactly one customer-node when departing from the starting point. These constraints in other words force the vehicle to start its tour from the starting point.

Constraints (4) on the other hand force the vehicle to return to the starting point in order to complete its tour.

Constraints (5) belong to the sub-tour elimination family. When an arc (*i*,*j*) is covered, the vehicle should visit node j immediately after node i. As a result the creation of sub-tours is avoided.

Constraints (5.1) show that the turn which corresponds to the starting point must be equal to one, as the problem must begin its solution from this node.

Constraints (5.2) set the range of values for the turns of every customer. In order for the solution to be feasible, the vehicle cannot start or end its tour from a customer-node.

Constraints (6) ensure that the vehicle cannot leave a node in order to visit the same node.

Constraints (7) and (8) refer to the binary and non-negativity of the decision variables respectively.

The objective function (9) represents the total travelling distance covered by the vehicle in order to satisfy all of the customers' demands. Through the minimization of the total travelling distance, the lowest-cost tour is achieved, which is the ultimate goal of the TSP.

3.3 Coding of TSP

3.3.1 Modifications for two different approaches of the same problem

Both codes read a .txt file that includes the shortest path distances of all the nodes in the problem. The .txt file is a product of the Python code, which exploits the services of the Google Distance Matrix API. In order for the user to receive an optimal solution, the last row and column of the matrix should refer to the starting point of the problem. To achieve the above, the .txt file that the Python code reads should have the coordinates of the starting point inserted last. From this point forward we will refer to the codes as "Fixed-Matrix Code" and "Sub-Matrix Code" respectively. The first one requires to give values to *n* (customers) and *m* (starting point = 1) at the beginning of the code, while the second one requires an input value only for *n* (all nodes). The main difference between the two codes is that with the "Fixed-Matrix" the user can run an example for *n*+*m* specific nodes, while with the "Sub-Matrix" he can run multiple examples, by every time selecting a different combination of *n* nodes from a .txt file that contains the distances of a larger whole of nodes. The above leads to the conclusion that the "Sub-Matrix Code" can be very time-saving, due to the fact that the user in order to run a variety of examples should only run the Python code once.

3.3.2 Fixed-Matrix Code

The "Fixed-Matrix Code" written in *C++* is presented below:

```
/* Travelling Salesman Problem */

#include <ilcplex/ilocplex.h>
ILOSTLBEGIN

#include <vector>
#include <fstream>
#include <stdlib.h>
using std::vector;


```

```cpp
int main() {

    // Data

    int             i,j;                                // i,j: pointer of nodes for customers and starting point
    const int       imax=30;                            // max nodes (customer and starting point)
    const int       jmax=30;                            // max nodes (customer and starting point)
    const int       n=29;                               // n: number of customers
    const int       m=1;                                // m: starting point

    float           D[imax][jmax];                      // Table of the distances from customer to customer and from starting point to customers defined in km


//----------------------------------------------------------------------------------------------------------------------------------------------------------------------

    // Welcoming Message
    cout<<"--------------------------------------------------------------------"<<endl;
    cout<<"|                          TSP Solver                         |"<<endl;
    cout<<"--------------------------------------------------------------------"<<endl;
    cout<<endl;
    cout<<"<------> Plan your products' distribution in an optimal way <------>"<<endl;
    cout<<endl;
    cout<<"Designed by:"<<endl;
    cout<<"Panagiotis Vrakidis"<<endl;
    cout<<endl;
    cout<<"Press Enter to continue."<<endl;
    cin.get();
//-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

    // Initialization:Set table of distances equal to zero//
    for (i=0;i<n+m;i++){
        for (j=0;j<n+m;j++){
            D[i][j]=0;
        }
    }

//---------------------------------------------------------------------------------------------------------------------------------------------------------------------

    // Import of Data

    // Import of Distance Data
    ifstream ifs_1;
    ifs_1.open("DATA.txt");

    for (int i=0;i<n+m;i++){
        for (int j=0;j<n+m;j++){
            ifs_1 >> D[i][j];
        }
    }

//----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

    // Building Model

    IloEnv env;

    try {
        IloModel model (env);

        typedef IloArray<IloNumVarArray> IloNumVarMatrix2x2;
        typedef IloArray<IloNumVarMatrix2x2> IloNumVarMatrix3x3;
        typedef IloArray<IloNumVarMatrix3x3> IloNumVarMatrix4x4;

        typedef IloArray<IloRangeArray> IloRangeMatrix2x2;
        typedef IloArray<IloRangeMatrix2x2> IloRangeMatrix3x3;
        typedef IloArray<IloRangeMatrix3x3> IloRangeMatrix4x4;

        IloCplex cplex(env);
//----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

    // Decision Variables

    // Xij: Binary variable which indicates if a route is covered or not
            IloNumVarMatrix2x2 Xij(env,0);
                for (i=0;i<n+m;i++){
                    IloNumVarArray Xj(env,0);
                    for (j=0;j<n+m;j++){
                        char Path[70];
                        sprintf(Path,"Xij(i%d,j%d)",i,j);
                        IloNumVar X(env,0,1,ILOBOOL,Path);
                        Xj.add(X);
                    }
                    Xij.add(Xj);
                }

    //ui: Non-negative integer variable which indicates the turn that each node is visited

                IloNumVarArray ui(env,0);
                for (i=0;i<n+m;i++){
                          char Turn[70];
                          sprintf(Turn,"ui(i%d)",i);
                          IloNumVar u(env,1,n+m,ILOINT,Turn);
                          ui.add(u);
                      }

      //---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

          // Constraints

          // Constraints (1): Each customer is visited exactly once
              IloRangeArray Sum2_Xj(env,0);
              for (j=0;j<n;j++){
                  IloExpr expr(env,0);
                  for (i=0;i<n+m;i++){
                              expr+=Xij[i][j];
                  }
                  int LB=1, UB=1;
                  IloRange Sum2_X(env,LB,expr,UB);
                  expr.end();
                  model.add(Sum2_X);
                  Sum2_Xj.add(Sum2_X);
              }


          // Constraints (2): From every node of a customer, the vehicle will depart for another node.
              IloRangeArray Sum5_Xi(env,0);
              for (i=0;i<n;i++){
                  IloExpr expr(env,0);
                  for (j=0;j<n+m;j++){
                              expr+=Xij[i][j];
                          }
                  int LB=1, UB=1;
                  IloRange Sum5_X(env,LB,expr,UB);
                  expr.end();
                  model.add(Sum5_X);
                  Sum5_Xi.add(Sum5_X);
              }


          //Constraints (3): The vehicle will serve exactly one customer node when departing from the starting point.
              IloRangeArray Sum6_Xi(env,0);
              for (i=n;i<n+m;i++){
                  IloExpr expr(env,0);
                  for (j=0;j<n;j++){
                      expr+=Xij[i][j];
                  }
                  int LB=1, UB=1;
                  IloRange Sum6_X(env,LB,expr,UB);
                  expr.end();
                  model.add(Sum6_X);
                  Sum6_Xi.add(Sum6_X);
              }

          //Constraints (4): The vehicle must return to the starting point.

              IloRangeArray Sum7_Xj(env,0);
              for (j=n;j<n+m;j++){
                  IloExpr expr(env,0);
                  for (i=0;i<n;i++){
                              expr+=Xij[i][j];
                          }
                  int LB=1, UB=1;
                  IloRange Sum7_X(env,LB,expr,UB);
                  expr.end();
                  model.add(Sum7_X);
                  Sum7_Xj.add(Sum7_X);
              }

          //Constraints (5): Sub-tour elimination
              IloRangeMatrix2x2 Subtourij(env,0);
              for(i=0;i<n;i++){
                  IloRangeArray Subtourj(env,0);
                      for(j=0;j<n;j++){
                          if (i!=j){
                              IloExpr expr(env,0);
                              expr+=ui[i]-ui[j]+1-(n+m-1)*(1-Xij[i][j]);
                              char Subtours[70];
                              sprintf(Subtours,"Subtour(i%d,j%d)",i,j);
                              float LB=-IloInfinity,UB=0;
                              IloRange Subtour(env,LB,expr,UB,Subtours);
                              expr.end();
                              model.add(Subtour);
                              Subtourj.add(Subtour);
                          }
                      }
                      Subtourij.add(Subtourj);
                  }

          // Constraints (5.1)
              IloRangeArray Sub_ui(env,0);
                  for (i=n;i<n+m;i++){
                      IloExpr expr(env,0);
                          expr+=ui[i];
                          int LB=1, UB=1;
                              IloRange Sub_u(env,LB,expr,UB);
                              expr.end();
                              model.add(Sub_u);
                              Sub_ui.add(Sub_u);
                          }
          // Constraints (5.2)
              IloRangeArray Sub1_ui(env,0);
                  for (i=0;i<n;i++){
                      IloExpr expr(env,0);
                          expr+=ui[i];
                          int LB=2, UB=n+m;
                              IloRange Sub1_u(env,LB,expr,UB);
                              expr.end();
                              model.add(Sub1_u);
                              Sub1_ui.add(Sub1_u);
                          }


          // Constraints (6): The vehicle cannot leave a node in order to visit the same node.
                  IloRangeArray Zero1_Xi(env,0);
                      for (i=0;i<n+m;i++){
                          IloExpr expr(env,0);
                          expr+=Xij[i][i];
                          int LB=0, UB=0;
                              IloRange Zero1_X(env,LB,expr,UB);
                              expr.end();
                              model.add(Zero1_X);
                              Zero1_Xi.add(Zero1_X);
                          }





      //-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

          // Objective Function: Minimization of the total travelling distance.
              IloExpr expr_obj(env);
                      for (i=0;i<n+m;i++){
                          for (j=0;j<n+m;j++){
                              expr_obj+=D[i][j]*Xij[i][j];
                          }
                      }

              model.add(IloMinimize(env,expr_obj));
              expr_obj.end();
```

```
       //-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

           // Solve
            cplex.extract(model);
            cplex.exportModel("onoma.lp");


           // Print Results
               if (!cplex.solve ()){
                   env.error()<<"Failed to optimize LP."<<endl;
                   throw(-1);
               }

               env.out()<<"Solution status = " <<cplex.getStatus()<<endl;
               env.out()<<"Solution value = " <<cplex.getObjValue()<<endl;

               cplex.solve();

           // Print Xij
               cout<<"The optimal routes are:"<<endl;
               for (i=0;i<n+m;i++){
                   for (j=0;j<n+m;j++){
                               int g = cplex.getValue(Xij[i][j]);
                                   if(g!=0)cout<<"Xij"<<"("<<i<<","<<j<<")"<<"="<<g<<endl;
                   }
               }
               cout<<endl;

           // Print ui
               cout<<"The turn of each service is:"<<endl;
               for (i=0;i<n+m;i++){
                       int g = cplex.getValue(ui[i]);
                       if(g!=0) cout<<"ui"<<"("<<i<<")"<<"="<<g<<endl;
               }
                   cout<<endl;

          }
               catch ( IloException& e){
                   cerr <<"concert exception caught:"<<e<<endl;
               }
               catch (...){
                   cerr <<"Unknown exception caught"<<endl;
               }
       //-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

           // End of env
               env.end();

           // Thank you Message
               cout<<"<------> Thank you for using TSP Solver! <------>"<<endl;

               system("pause");
               return 0;

         }
        // End main
```

3.3.3 Sub-Matrix Code

The “Sub-Matrix Code” written in *C++* is presented below:

```cpp
/* Travelling Salesman Problem */

#include <ilcplex/ilocplex.h>
ILOSTLBEGIN

#include <vector>
#include <fstream>
#include <stdlib.h>
using std::vector;


int main() {

    // Data

    int             i,j;                        // i,j: pointer of nodes for customers,starting point
    const int       imax=100;                   // max nodes (customer and starting point)
    const int       jmax=100;                   // max nodes (customer and starting point)
    const int       n=20;                       // n: number of nodes for problem
    int             l,m,p,k;                    //l,m,p,k: pointer for the creation of the problems' submatrix

    float           D[imax][jmax];              // Table of the distances from customer to customer and from starting point to customers defined in km
    float           A[n][n];                    // Submatrix including the distances of the selected nodes for the TSP

//--------------------------------------------------------------------------------------------------------------------------------------------------------------------

    // Welcoming Message
    cout<<"-------------------------------------------------------------------"<<endl;
    cout<<"|                         TSP Solver                               |"<<endl;
    cout<<"-------------------------------------------------------------------"<<endl;
    cout<<endl;
    cout<<"<-------> Plan your products' distribution in an optimal way <------>"<<endl;
    cout<<endl;
    cout<<"Designed by:"<<endl;
    cout<<"Panagiotis Vrakidis"<<endl;
    cout<<endl;
    cout<<"Press Enter to continue."<<endl;
    cin.get();
//---------------------------------------------------------------------------------------------------------------------------------------------------------------------

    // Initialization:Set table of distances equal to zero//
    for (i=0;i<imax;i++){
        for (j=0;j<jmax;j++){
            D[i][j]=0;
        }
    }

//---------------------------------------------------------------------------------------------------------------------------------------------------------------------

    // Import of Data

    // Import of Distance Data
    ifstream ifs_1;
    ifs_1.open("DATA.txt");

    for (int i=0;i<imax;i++){
        for (int j=0;j<jmax;j++){
            ifs_1 >> D[i][j];
        }
    }

    // Subroutine for creation of dynamic submatrix
        vector<int>a;

        cout<<"Insert the number of each customer to be served"<<endl;
            for(i=0;i<n-1;i++){
            a.push_back(30);
            cin>> a[i];

        while (a[i]<0 || a[i]>imax-2){
        cout<<"Wrong number!Out of bounds!Try again"<<endl;
        cin>>a[i];
        cout<<endl;
        }
            for (int z=0;z<i;z++){
            int t=0;
            while (t==0){
            if (a[i]!=a[z]){
            t=1;
            }
            else{
                cout<<"The same number cannot be inserted twice into the problem! Try again!"<<endl;
                cin>>a[i];
                z=0;
            }
        }
    }
}
    cout<<"Type the number of the starting point"<<endl;
    a.push_back(30);
    cin>> a[n-1];
    while (a[n-1]!=99){
        cout<<"Wrong number! Number 99 represents the starting point. Try again!"<<endl;
        cin>> a[n-1];
    }

    l=0;
    m=0;
        for(i=0;i<n;i++){
               p=0;
               k=0;
                   for(j=0;j<n;j++){
                   A[l][k]=D[a[m]][a[p]];
                   p=p+1;
                   k=k+1;
               }
           m=m+1;
           l=l+1;
           }

       //-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

           // Building Model

           IloEnv env;

           try {
               IloModel model (env);

               typedef IloArray<IloNumVarArray> IloNumVarMatrix2x2;
               typedef IloArray<IloNumVarMatrix2x2> IloNumVarMatrix3x3;
               typedef IloArray<IloNumVarMatrix3x3> IloNumVarMatrix4x4;

               typedef IloArray<IloRangeArray> IloRangeMatrix2x2;
               typedef IloArray<IloRangeMatrix2x2> IloRangeMatrix3x3;
               typedef IloArray<IloRangeMatrix3x3> IloRangeMatrix4x4;

               IloCplex cplex(env);
       //-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

           // Decision Variables

           // Xij: Binary variable which indicates if a certain route is made or not
                       IloNumVarMatrix2x2 Xij(env,0);
                       for (i=0;i<n;i++){
                           IloNumVarArray Xj(env,0);
                           for (j=0;j<n;j++){
                               char Path[70];
                               sprintf(Path,"Xij(i%d,j%d)",i,j);
                               IloNumVar X(env,0,1,ILOBOOL,Path);
                               Xj.add(X);
                           }
                           Xij.add(Xj);
                       }

           //ui: Non-negative discrete variable which indicates the turn that every node is visited
                       IloNumVarArray ui(env,0);
                       for (i=0;i<n;i++){
                           char Turn[70];
                           sprintf(Turn,"ui(i%d)",i);
                           IloNumVar u(env,1,n,ILOINT,Turn);
                           ui.add(u);
                       }

       //----------------------------------------------------------------------------------------------------------------------------------------------------------------------

           // Constraints

           // Constraints (1): Each customer is visited exactly once
               IloRangeArray Sum2_Xj(env,0);
               for (j=0;j<n-1;j++){
```

```
            IloExpr expr(env,0);
            for (i=0;i<n;i++){
                  expr+=Xij[i][j];
            }
            int LB=1, UB=1;
            IloRange Sum2_X(env,LB,expr,UB);
            expr.end();
            model.add(Sum2_X);
            Sum2_Xj.add(Sum2_X);
          }


        // Constraints (2): From every node of a customer, the vehicle will depart for another node.
          IloRangeArray Sum5_Xi(env,0);
          for (i=0;i<n-1;i++){
            IloExpr expr(env,0);
            for (j=0;j<n;j++){
                  expr+=Xij[i][j];
                }
            int LB=1, UB=1;
            IloRange Sum5_X(env,LB,expr,UB);
            expr.end();
            model.add(Sum5_X);
            Sum5_Xi.add(Sum5_X);
          }


        //Constraints (3): The vehicle will serve one customer node when departing from the depot.
          IloRangeArray Sum6_Xi(env,0);
          for (i=n;i<n;i++){
            IloExpr expr(env,0);
            for (j=0;j<n-1;j++){
              expr+=Xij[i][j];
            }
            int LB=1, UB=1;
            IloRange Sum6_X(env,LB,expr,UB);
            expr.end();
            model.add(Sum6_X);
            Sum6_Xi.add(Sum6_X);
          }

        //Constraints (4): The vehicle must return to the depot.
          IloRangeArray Sum7_Xj(env,0);
          for (j=n-1;j<n;j++){
            IloExpr expr(env,0);
            for (i=0;i<n-1;i++){
                  expr+=Xij[i][j];
                }
            int LB=1, UB=1;
            IloRange Sum7_X(env,LB,expr,UB);
            expr.end();
            model.add(Sum7_X);
            Sum7_Xj.add(Sum7_X);
          }

        //Constraints (5): Sub-tour elimination
          IloRangeMatrix2x2 Subtourij(env,0);
            for(i=0;i<n-1;i++){
              IloRangeArray Subtourj(env,0);
              for(j=0;j<n-1;j++){
                if (i!=j){
                  IloExpr expr(env,0);
                  expr+=ui[i]-ui[j]+1-(n-1)*(1-Xij[i][j]);
```

```
                               char Subtours[70];
                               sprintf(Subtours,"Subtour(i%d,j%d)",i,j);
                               float LB=-IloInfinity,UB=0;
                               IloRange Subtour(env,LB,expr,UB,Subtours);
                               expr.end();
                               model.add(Subtour);
                               Subtourj.add(Subtour);
                           }
                       }
                       Subtourij.add(Subtourj);
                   }

           // Constraints (5.1)
               IloRangeArray Sub_ui(env,0);
                       for (i=n-1;i<n;i++){
                           IloExpr expr(env,0);
                               expr+=ui[i];
                               int LB=1, UB=1;
                               IloRange Sub_u(env,LB,expr,UB);
                               expr.end();
                               model.add(Sub_u);
                               Sub_ui.add(Sub_u);
                           }
           // Constraints (5.2)
               IloRangeArray Sub1_ui(env,0);
                       for (i=0;i<n-1;i++){
                           IloExpr expr(env,0);
                               expr+=ui[i];
                               int LB=2, UB=n;
                               IloRange Sub1_u(env,LB,expr,UB);
                               expr.end();
                               model.add(Sub1_u);
                               Sub1_ui.add(Sub1_u);
                           }


           // Constraints (6): The vehicle cannot leave a node in order to visit the same node.
               IloRangeArray Zero1_Xi(env,0);
                       for (i=0;i<n;i++){
                           IloExpr expr(env,0);
                               expr+=Xij[i][i];
                               int LB=0, UB=0;
                               IloRange Zero1_X(env,LB,expr,UB);
                               expr.end();
                               model.add(Zero1_X);
                               Zero1_Xi.add(Zero1_X);
                           }

       //-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

           // Objective Function: Minimization of the total travelling distance.
               IloExpr expr_obj(env);
                       for (i=0;i<n;i++){
                           for (j=0;j<n;j++){
                               expr_obj+=A[i][j]*Xij[i][j];
                           }
                       }

               model.add(IloMinimize(env,expr_obj));
               expr_obj.end();
       //-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

           // Solve
               cplex.extract(model);
               cplex.exportModel("onoma.lp");


           // Print Results
               if (!cplex.solve ()){
                   env.error()<<"Failed to optimize LP."<<endl;
                   throw(-1);
               }

               env.out()<<"Solution status = " <<cplex.getStatus()<<endl;
               env.out()<<"Solution value = " <<cplex.getObjValue()<<endl;

               cplex.solve();

           // Print Xij
               cout<<"The optimal routes are:"<<endl;
               for (i=0;i<n;i++){
                   for (j=0;j<n;j++){
                               int g = cplex.getValue(Xij[i][j]);
                                   if(g!=0)cout<<"Xij"<<"("<<a[i]<<","<<a[j]<<")"<<"="<<g<<endl;
                   }
               }
               cout<<endl;

           // Print ui
               cout<<"The turn of each service is:"<<endl;
               for (i=0;i<n;i++){
                       int g = cplex.getValue(ui[i]);
                       if(g!=0) cout<<"ui"<<"("<<a[i]<<")"<<"="<<g<<endl;
               }
                   cout<<endl;
       }
               catch ( IloException& e){
                   cerr <<"concert exception caught:"<<e<<endl;
               }
               catch (...){
                   cerr <<"Unknown exception caught"<<endl;
               }
       //-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

           // End of env
               env.end();

           // Thank you Message
               cout<<"<------> Thank you for using TSP Solver! <------>"<<endl;

               system("pause");
               return 0;

         }
        // End main
```

Chapter 4: Visualization Process

4.1 The visualization approach

As mentioned before, the aim of this project is to approach a real-life instance with the TSP formulation. In order for the user to obtain the big picture of the solution produced, its visualization should be kept in as real standards as possible. OpenStreetMap along with QGIS play a significant role to the above due to the services they provide.

4.1.1 OpenStreetMap

OSM is a project that was built in order to create an editable world map, free for all users. The restrictions on use as well as availability of map information across the world have contributed in the continuous growth of the project. It was created by Steve Coast in the UK in 2004 and has reached over 2 million registered users. These users can use GPS devices and whichever free source they prefer for updating the OSM database. The map as displayed in the official website of OSM is presented in Fig. 1:

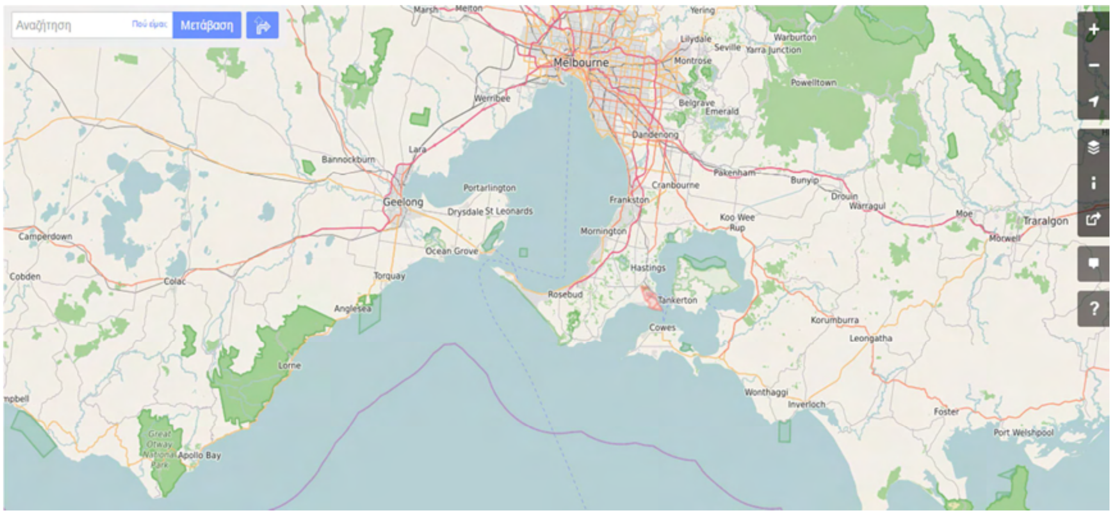
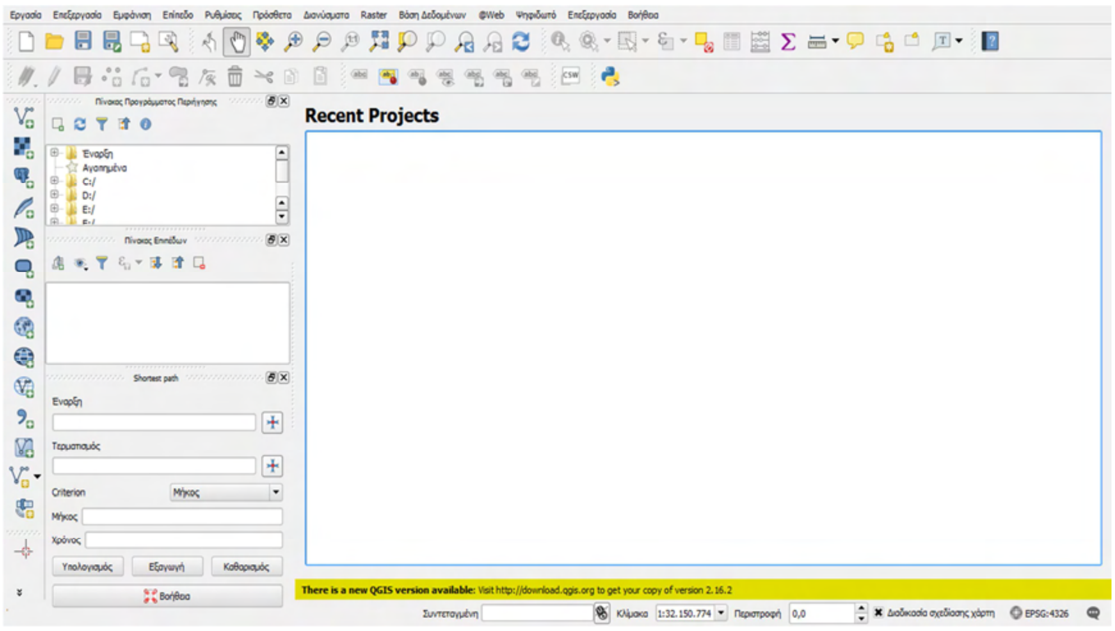


Fig. 1: OpenStreetMap platform[1]

[1] https://www.openstreetmap.org/

4.1.2 QGIS

QGIS or else known as Quantum GIS, is a geographic information system application which provides the services of not only viewing but also editing data. It allows users to edit maps as well as maps to be composed of raster or vector layers. As someone can easily realize, this software has a variety of tools which can contribute to displaying the optimal tour of the TSP. One can take a glance at the interface of QGIS 2.14 in Fig. 2.



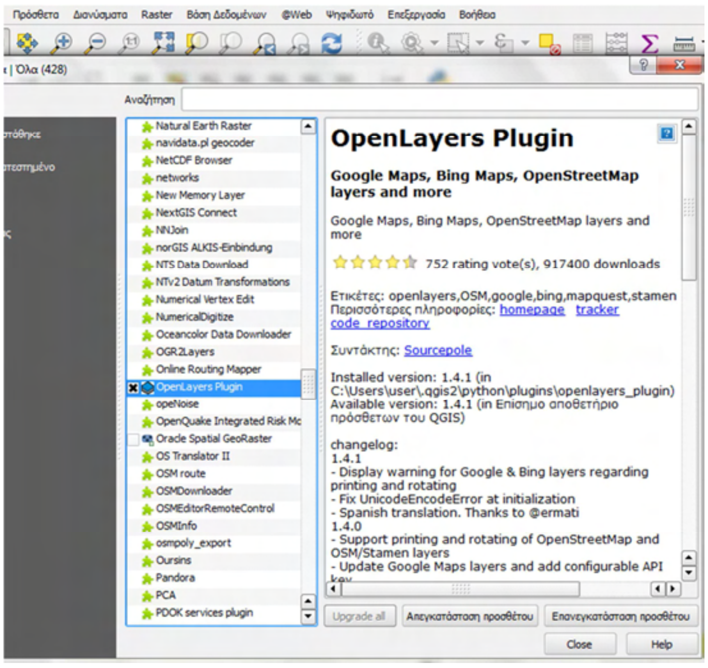
There is a new QGIS version available: Visit http://download.qgis.org to get your copy of version 2.16.2

Fig. 2: Quantum QGIS 2.14 interface

In order to display the map of OSM through QGIS, the OpenLayersPlugin from the Plugins menu must first be installed. To achieve the above, the user should follow the path presented below:

Plugins Menu → Manage Plugins → OpenLayersPlugin

For this procedure to be even clearer, the path is shown in Fig. 3.



OpenLayers Plugin

Google Maps, Bing Maps, OpenStreetMap layers and more

Google Maps, Bing Maps, OpenStreetMap layers and more

☆☆☆☆☆ 752 rating vote(s), 917400 downloads

Ετικέτες: openlayers,OSM,google,bing,mapquest,stamen
Περισσότερες πληροφορίες: homepage tracker code_repository

Συντάκτης: Sourcepole

Installed version: 1.4.1 (in C:\Users\user\.qgis2\python\plugins\openlayers_plugin)
Available version: 1.4.1 (in Επίσημο αποθετήριο πρόσθετων του QGIS)

changelog:
1.4.1
- Display warning for Google & Bing layers regarding printing and rotating
- Fix UnicodeEncodeError at initialization
- Spanish translation. Thanks to @ermati
1.4.0
- Support printing and rotating of OpenStreetMap and OSM/Stamen layers
- Update Google Maps layers and add configurable API key

Fig. 3: OpenLayersPlugin installation

Once the above procedure is completed everything is set for the map to be displayed at the interface of QGIS. The following path ensures that the map is loaded and ready to be edited:

@Web → OpenLayers plugin → OpenStreetMap → OpenStreetMap

The path along with its results is displayed in Fig. 4:

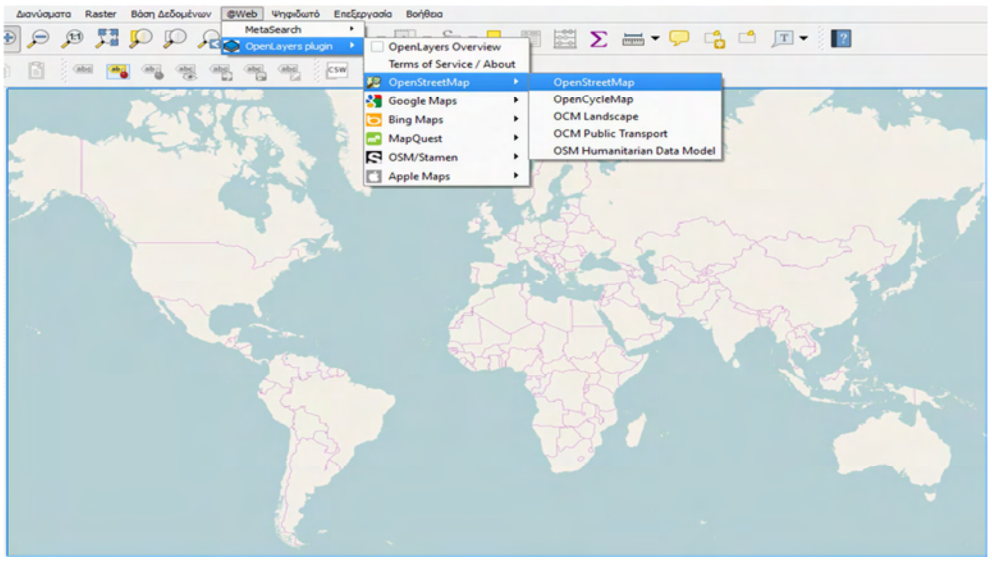


Fig. 4: OSM through QGIS

4.2 OSM data

For the map to be editable via QGIS, the appropriate OSM data must be inserted to it. The real-life problem which is examined in this project refers to the city of Volos. Therefore, a way must be found in order to collect the road network data of the city and then import it to the map. This will enable the display of the shortest path for each route that exists in the TSP.

4.2.1 Overpass turbo

Overpass turbo is a web based tool that runs specific queries in order to enable the user to mine data for OpenStreetMap. The results of each query are shown on an interactive map and there is also the feature of extracting the data in a variety of types. The structure of this online platform is presented in Fig. 5.

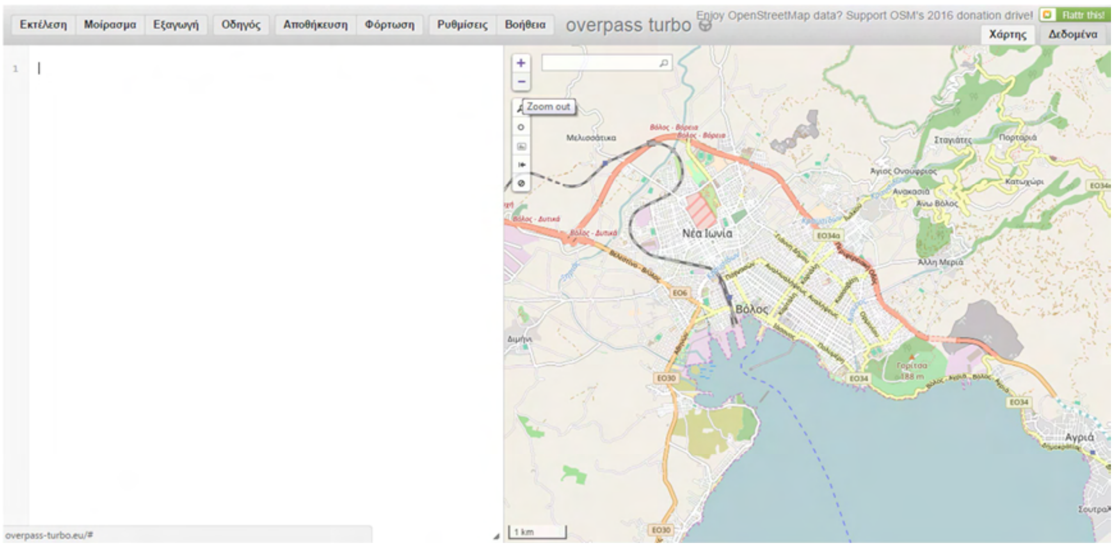


Fig. 5: City of Volos in the interface of Overpass turbo[2]

The blank space on the left side of the above figure is for the queries to be written and then sent to the Overpass API, which is an online database for OSM. Thus, the next step would be to form a code for a query, which would demand the road network data of Volos.

[2] http://overpass-turbo.eu/

4.2.2 Code for road network

The purpose of this section is to present a code that was written in XML in order to form a query that would ask for the road network data of Volos from the Overpass API. This code is based on three main axes of information:

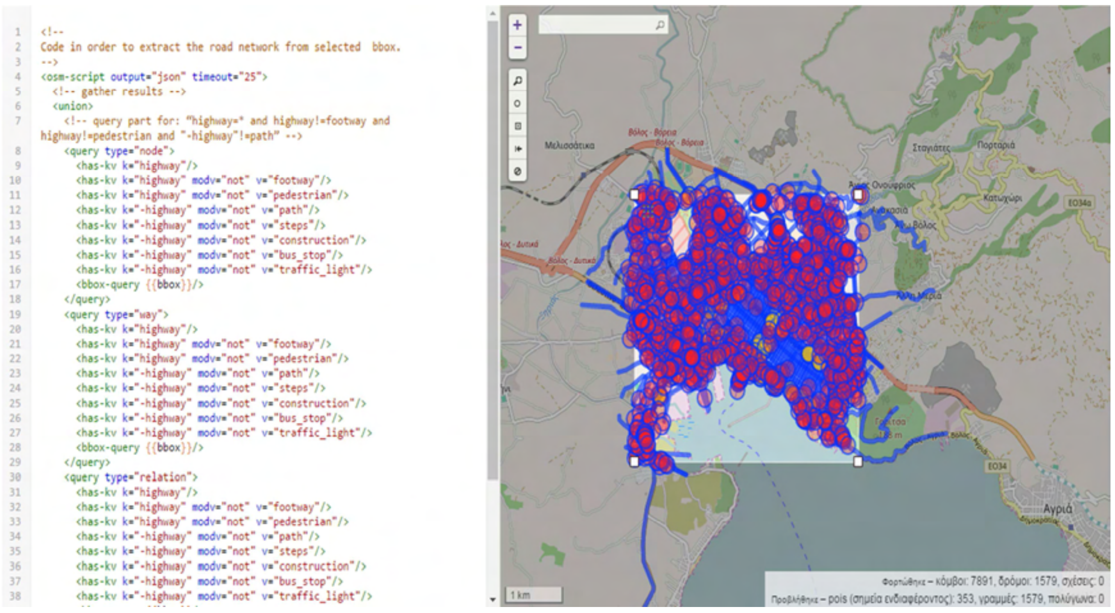
- Node: refers to the nodes that are required in order to determine every road in the network,
- Way: for the lines that each road forms,
- Relation: concerning the name as well as the direction of each road.

The complete road-network code is provided below:

```
<!--
Code in order to extract the road network from selected bbox.
-->
<osm-script output="json" timeout="25">
  <!-- gather results -->
  <union>
    <!--
 query part for: â€œhighway=* and highway!=footway and highway!=pedestrian a
nd "-highway"!=pathâ€  -->
    <query type="node">
      <has-kv k="highway"/>
      <has-kv k="highway" modv="not" v="footway"/>
      <has-kv k="highway" modv="not" v="pedestrian"/>
      <has-kv k="-highway" modv="not" v="path"/>
      <has-kv k="-highway" modv="not" v="steps"/>
      <has-kv k="-highway" modv="not" v="construction"/>
      <has-kv k="-highway" modv="not" v="bus_stop"/>
      <has-kv k="-highway" modv="not" v="traffic_light"/>
      <bbox-query {{bbox}}/>
    </query>
    <query type="way">
      <has-kv k="highway"/>
      <has-kv k="highway" modv="not" v="footway"/>
      <has-kv k="highway" modv="not" v="pedestrian"/>
      <has-kv k="-highway" modv="not" v="path"/>
      <has-kv k="-highway" modv="not" v="steps"/>
      <has-kv k="-highway" modv="not" v="construction"/>
      <has-kv k="-highway" modv="not" v="bus_stop"/>
      <has-kv k="-highway" modv="not" v="traffic_light"/>
      <bbox-query {{bbox}}/>
    </query>
    <query type="relation">
      <has-kv k="highway"/>
      <has-kv k="highway" modv="not" v="footway"/>
      <has-kv k="highway" modv="not" v="pedestrian"/>
      <has-kv k="-highway" modv="not" v="path"/>
      <has-kv k="-highway" modv="not" v="steps"/>
      <has-kv k="-highway" modv="not" v="construction"/>
      <has-kv k="-highway" modv="not" v="bus_stop"/>
      <has-kv k="-highway" modv="not" v="traffic_light"/>
      <bbox-query {{bbox}}/>
    </query>
  </union>
  <!-- print results -->
  <print mode="body"/>
  <recurse type="down"/>
  <print mode="skeleton" order="quadtile"/>
</osm-script>
```

4.2.3 Extraction of data

The feature of bbox allows the user to run the code for a specific region. In this specific case, the city of Volos is selected and once the code is executed, the results are displayed on the map. Fig. 6 shows the bbox selected for the city of Volos as well as the results that occurred from the query that was formed.



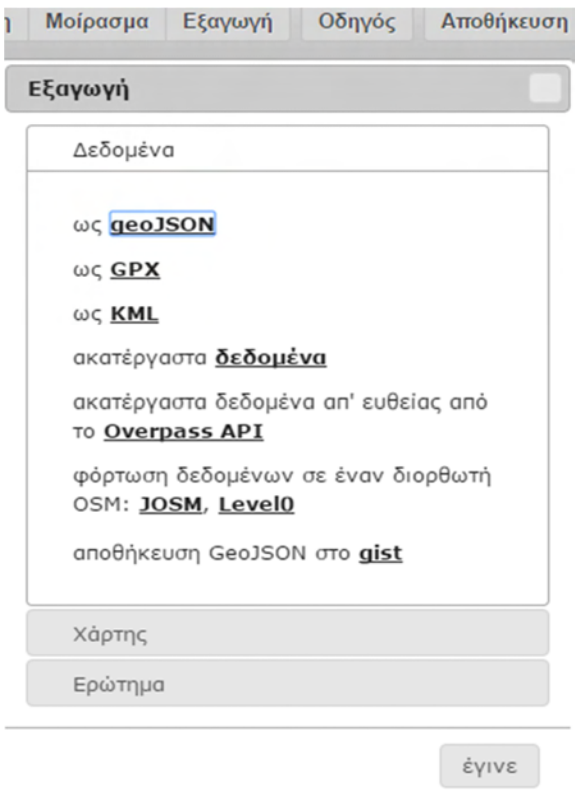
```
<!--
Code in order to extract the road network from selected  bbox.
-->
<osm-script output="json" timeout="25">
  <!-- gather results -->
  <union>
    <!-- query part for: "highway=" and highway!=footway and
highway!=pedestrian and "-highway"!=path" -->
    <query type="node">
      <has-kv k="highway"/>
      <has-kv k="highway" modv="not" v="footway"/>
      <has-kv k="highway" modv="not" v="pedestrian"/>
      <has-kv k="-highway" modv="not" v="path"/>
      <has-kv k="-highway" modv="not" v="steps"/>
      <has-kv k="-highway" modv="not" v="construction"/>
      <has-kv k="-highway" modv="not" v="bus_stop"/>
      <has-kv k="-highway" modv="not" v="traffic_light"/>
      <bbox-query {{bbox}}/>
    </query>
    <query type="way">
      <has-kv k="highway"/>
      <has-kv k="highway" modv="not" v="footway"/>
      <has-kv k="highway" modv="not" v="pedestrian"/>
      <has-kv k="-highway" modv="not" v="path"/>
      <has-kv k="-highway" modv="not" v="steps"/>
      <has-kv k="-highway" modv="not" v="construction"/>
      <has-kv k="-highway" modv="not" v="bus_stop"/>
      <has-kv k="-highway" modv="not" v="traffic_light"/>
      <bbox-query {{bbox}}/>
    </query>
    <query type="relation">
      <has-kv k="highway"/>
      <has-kv k="highway" modv="not" v="footway"/>
      <has-kv k="highway" modv="not" v="pedestrian"/>
      <has-kv k="-highway" modv="not" v="path"/>
      <has-kv k="-highway" modv="not" v="steps"/>
      <has-kv k="-highway" modv="not" v="construction"/>
      <has-kv k="-highway" modv="not" v="bus_stop"/>
      <has-kv k="-highway" modv="not" v="traffic_light"/>
```

Φορτώθηκε – κόμβοι: 7891, δρόμοι: 1579, σχέσεις: 0
Προβλήθηκε – pois (σημεία ενδιαφέροντος): 353, γραμμές: 1579, πολύγωνα: 0

Fig. 6: Volos' bbox and query results[3]

The final step is to extract the data from the Overpass turbo platform. This is possible by selecting the "Extract" command and by choosing the preferred type of the produced data. At this point, it should be stated that the type "geoJSON" was selected, as it is compatible with QGIS. Fig. 7 clarifies the above mentioned procedure.

[3] http://overpass-turbo.eu/



ως geoJSON

ως GPX

ως KML

ακατέργαστα δεδομένα

ακατέργαστα δεδομένα απ' ευθείας από το Overpass API

φόρτωση δεδομένων σε έναν διορθωτή OSM: JOSM, Level0

αποθήκευση GeoJSON στο gist

Fig. 7: Extraction of road-network data as "geoJSON"[4]

4.3 Editing with QGIS

Having extracted all the necessary data for the map, the user can now utilize the tools of QGIS for editing and creating shortest path routes, which are part of the solution of any real-life TSP. In this section, the main methodology of displaying shortest path routes between nodes will be presented. For this purpose, all the essential features of QGIS will be put into practice for the simplest example of two nodes.

4.3.1 A two-node example

Two random nodes in the city of Volos have been selected for the purpose of demonstrating the procedure in order to display the shortest path distance between them on the OSM. Before the steps are analyzed, it should be highlighted that the order with which a layer is opened through QGIS, plays a significant role to how it will be displayed on the map. Therefore, in this example, the OpenStreetMap layer should be opened first, followed by that of the road-network. The final layer must be the one with the two selected nodes.

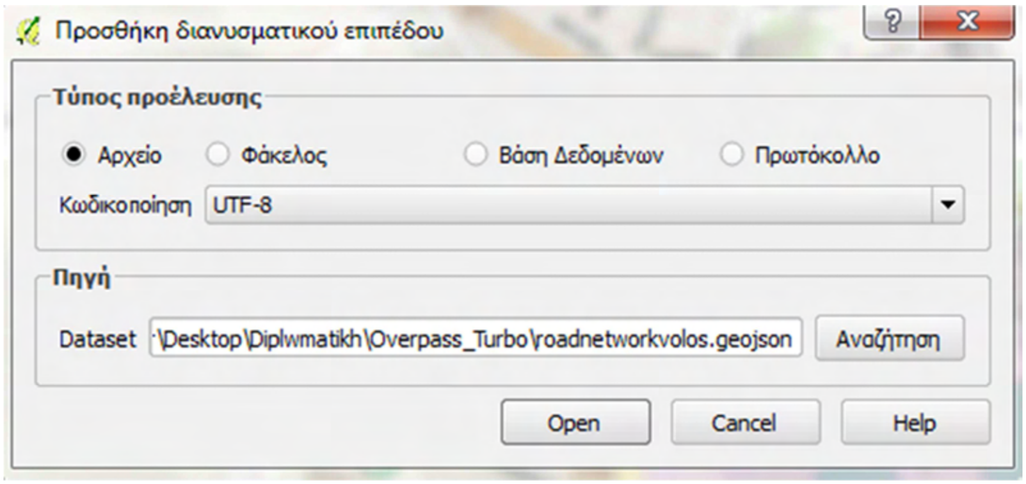
[4] http://overpass-turbo.eu/

4.3.2 Road-network data through QGIS

Having opened the OpenStreetMap layer, the road-network data of Volos is next to being imported to QGIS. The above is possible by following the path:

Layer → Add layer → Add vector layer

The file including the road-network data should be selected and then opened with the appropriate encoding type, which is UTF-8. The procedure as well as its outcome is shown in Fig. 8 and in Fig. 9:


Προσθήκη διανυσματικού επιπέδου

Τύπος προέλευσης

[✓ Αρχείο] Φάκελος Βάση Δεδομένων Πρωτόκολλο

Κωδικοποίηση UTF-8

Πηγή

Dataset \Desktop\Diplwmatikh\Overpass_Turbo\roadnetworkvolos.geojson

Αναζήτηση

Open Cancel Help

Fig. 8: Road-network data selection

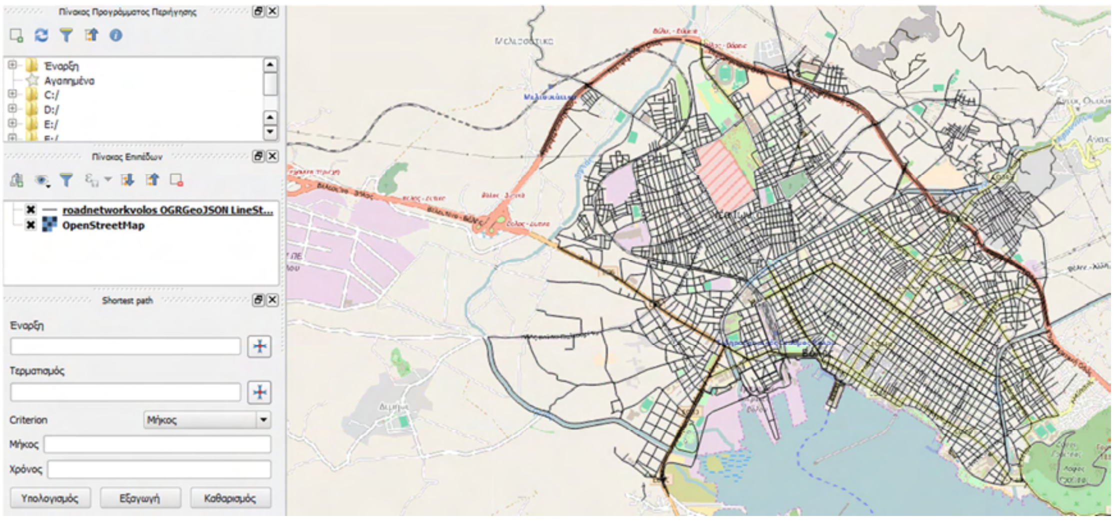
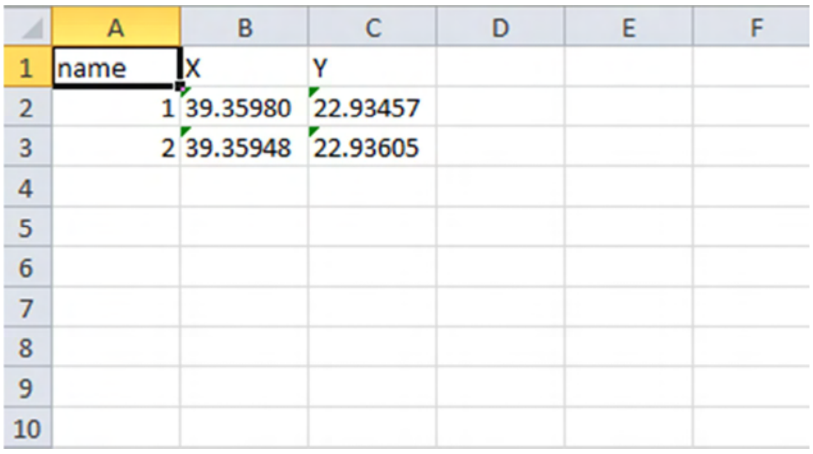


Fig. 9: Depiction of the road-network

Now everything is set for the layer including the two nodes of the example to be opened.

4.3.3 Depiction of the two nodes

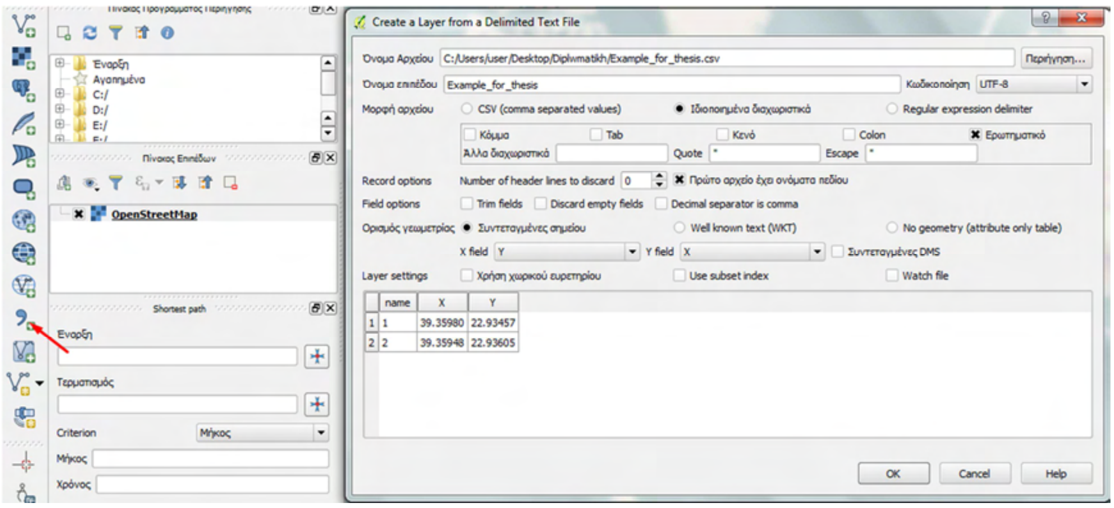
The random nodes that have been selected must be first inserted into a Microsoft Excel file, having the form that is demonstrated in Fig. 10.



	A	B	C	D	E	F
1	name	X	Y			
2	1	39.35980	22.93457			
3	2	39.35948	22.93605			
4						
5						
6						
7						
8						
9						
10						

Fig. 10: Microsoft Excel file with the two nodes

When the Microsoft Excel file is completed, it should be saved as a specific type of file that is compatible with QGIS. Therefore, a CSV comma delimited type of file must be created. This is crucial in order to proceed with the display of the two nodes on the map. The next step would be to utilize an extension of QGIS that allows the user to create a layer from a delimited text file as presented in Fig. 11:



Create a Layer from a Delimited Text File

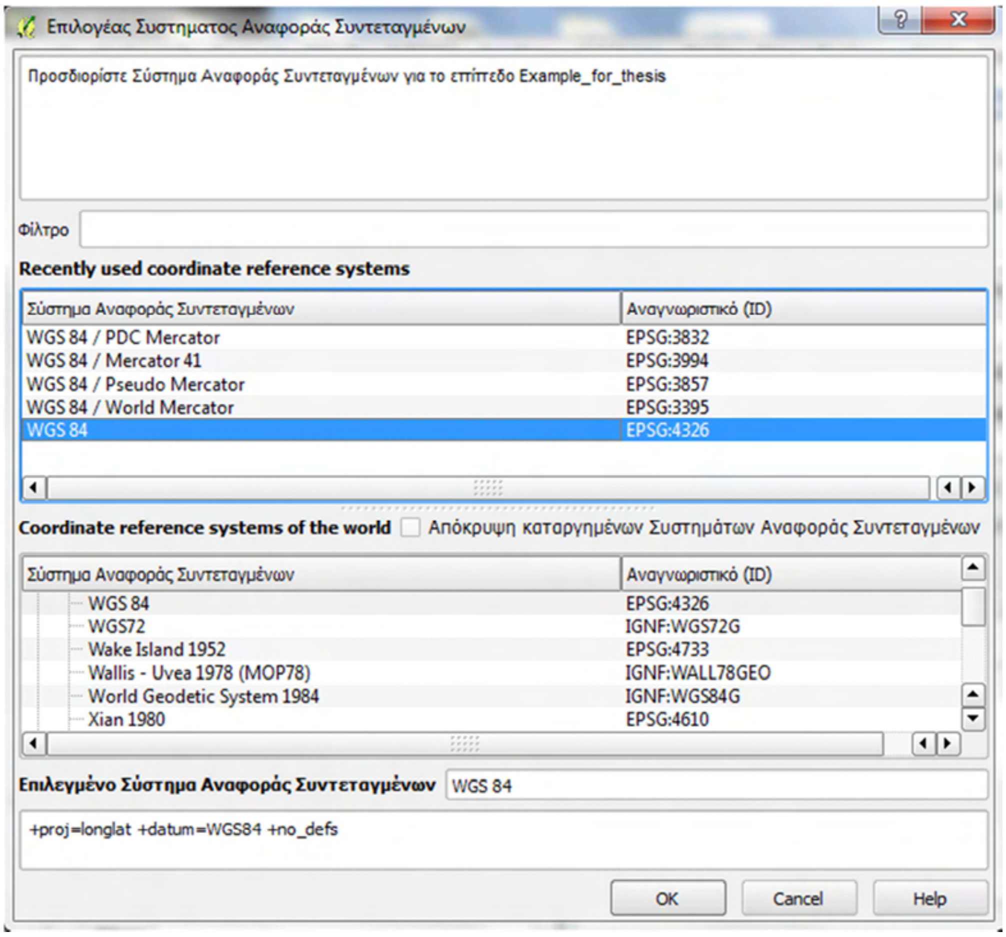
Όνομα Αρχείου: C:/Users/user/Desktop/Diplwmatikh/Example_for_thesis.csv
Όνομα επιπέδου: Example_for_thesis — Κωδικοποίηση: UTF-8
Μορφή αρχείου: CSV (comma separated values) [✓ Ιδιοποιημένα διαχωριστικά] Regular expression delimiter
Κόμμα, Tab, Κενό, Colon, [✓ Ερωτηματικό]
Άλλα διαχωριστικά: — Quote: " — Escape: "
Record options: Number of header lines to discard: 0 — [✓ Πρώτο αρχείο έχει ονόματα πεδίου]
Field options: Trim fields, Discard empty fields, Decimal separator is comma
Ορισμός γεωμετρίας: [✓ Συντεταγμένες σημείου] Well known text (WKT) No geometry (attribute only table)
X field: Y — Y field: X — Συντεταγμένες DMS
Layer settings: Χρήση χωρικού ευρετηρίου, Use subset index, Watch file

	name	X	Y
1	1	39.35980	22.93457
2	2	39.35948	22.93605

Shortest path: Έναρξη; Τερματισμός; Criterion: Μήκος; Μήκος; Χρόνος

Fig. 11: Selection of CSV file

For this step, a very important note should be made. QGIS reads the coordinates of the notes, while transposing them. Therefore, in the field of the *X* axis the value *Y* must be inserted and the opposite stands for the field of the *Y* axis. By selecting "OK", a new window comes up, which requires the selection of the coordinates' system for the depiction of the nodes. In order to clarify the above, Fig. 12 is displayed.



Επιλογέας Συστηματος Αναφοράς Συντεταγμένων

Προσδιορίστε Σύστημα Αναφοράς Συντεταγμένων για το επίπεδο Example_for_thesis

Φίλτρο

Recently used coordinate reference systems

Σύστημα Αναφοράς Συντεταγμένων	Αναγνωριστικό (ID)
WGS 84 / PDC Mercator	EPSG:3832
WGS 84 / Mercator 41	EPSG:3994
WGS 84 / Pseudo Mercator	EPSG:3857
WGS 84 / World Mercator	EPSG:3395
WGS 84	EPSG:4326

Coordinate reference systems of the world ☐ Απόκρυψη καταργημένων Συστημάτων Αναφοράς Συντεταγμένων

Σύστημα Αναφοράς Συντεταγμένων	Αναγνωριστικό (ID)
WGS 84	EPSG:4326
WGS72	IGNF:WGS72G
Wake Island 1952	EPSG:4733
Wallis - Uvea 1978 (MOP78)	IGNF:WALL78GEO
World Geodetic System 1984	IGNF:WGS84G
Xian 1980	EPSG:4610

Επιλεγμένο Σύστημα Αναφοράς Συντεταγμένων WGS 84

+proj=longlat +datum=WGS84 +no_defs

Fig. 12: Selection of coordinates' reference system

As shown above, the latest revision of the world geodetic system is selected, which is no other than the WGS 84. It is the most popular system amongst companies and therefore is the ideal fit for the approach of real-life travelling salesman problems. Everything is set, in order to display the two nodes along with the road-network of Volos' city (Fig. 13).

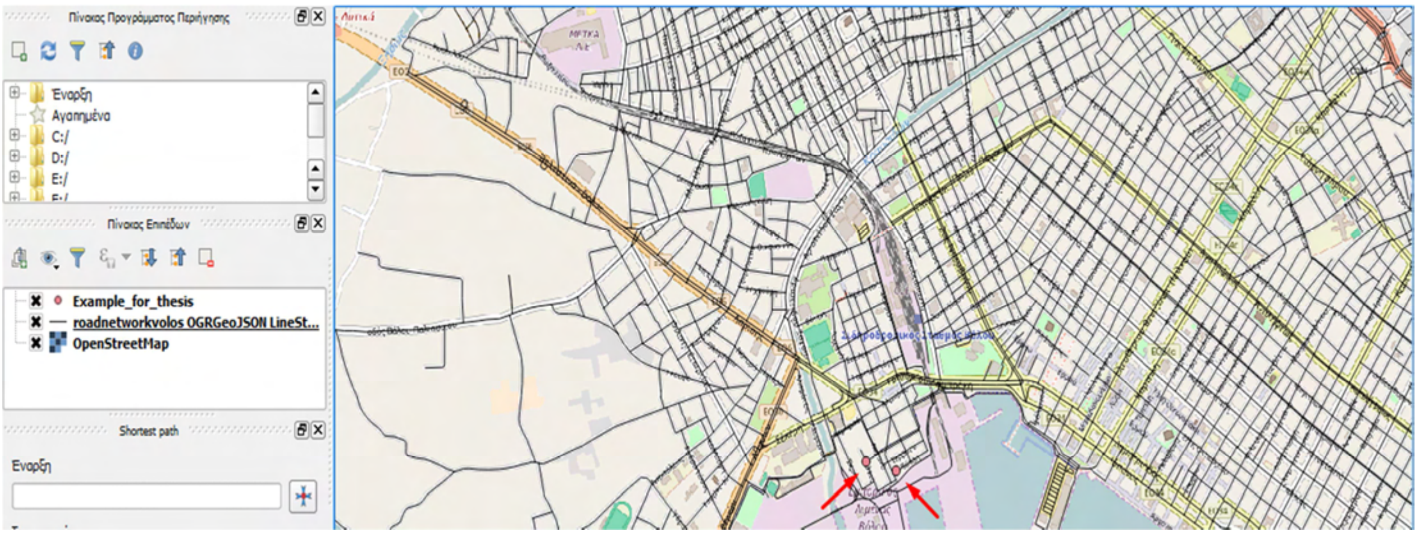


Fig. 13: Nodes in the road-network of Volos

4.3.4 Shortest path via QGIS

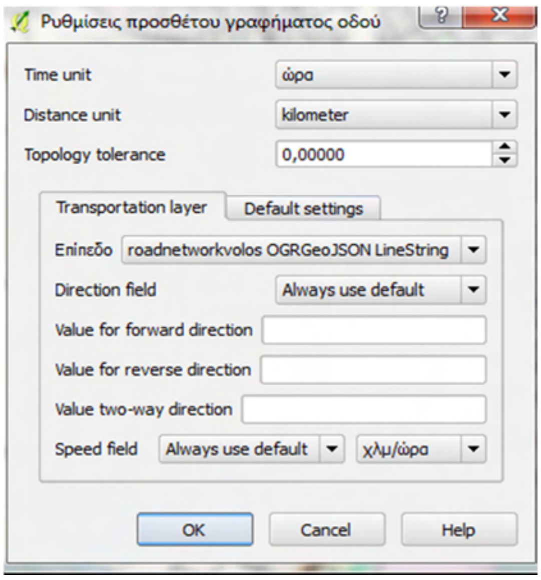
The calculation of the shortest path between the two nodes is the final step in order to complete the visualization procedure. QGIS provides a tool named “shortest path”, which can calculate the shortest path from one node to another, just by selecting the two points from the OSM. This plugin will appear in the interface of QGIS by following the path stated below:

View ⟶ Panels ⟶ Shortest path

However, in order to be able to utilize this tool, the extension named “road graph” must be configured. This is possible through the following path:

Vectors ⟶ Road graph ⟶ Settings

In the settings’ panel, one can select the distance units of the shortest path as well as adjusting a variety of parameters, which affect the operation of the “shortest path” tool. Fig. 14 summarizes the above mentioned.



Ρυθμίσεις προσθέτου γραφήματος οδού

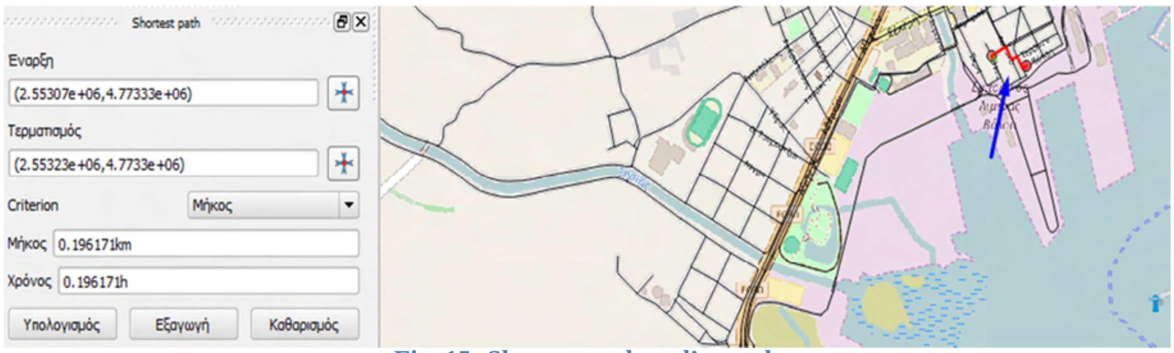
Time unit	ώρα
Distance unit	kilometer
Topology tolerance	0,00000

Transportation layer | Default settings

Επίπεδο	roadnetworkvolos OGRGeoJSON LineString
Direction field	Always use default
Value for forward direction	
Value for reverse direction	
Value two-way direction	
Speed field	Always use default / χλμ/ώρα

Fig. 14: Road graph configuration

Having the “road graph” extension configured, one can now calculate the shortest path between two points though the “shortest path” tool. An example that demonstrates the services of this tool is provided by Fig. 15.



Shortest path

Εναρξη	(2.55307e+06,4.77333e+06)
Τερματισμός	(2.55323e+06,4.7733e+06)
Criterion	Μήκος
Μήκος	0.196171km
Χρόνος	0.196171h

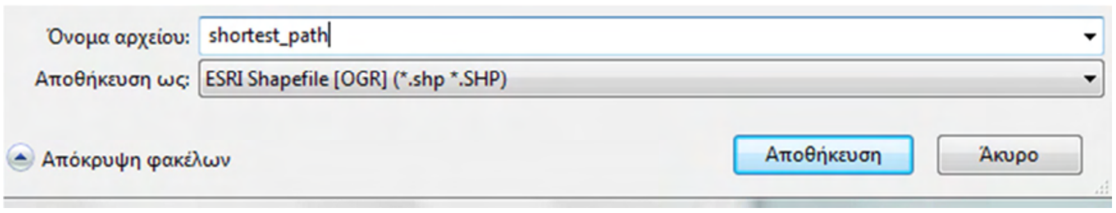
Fig. 15: Shortest path tool’s result

Finally, the shortest path can be extracted and then saved. This is very convenient as it allows the user to import the shortest path and display it on the map without the road-network. For this purpose, the term of “shapefile” will be introduced.

Shapefiles

The term shapefile refers to a digital vector storage format. This format enables the storage of geometric location as well as associated attribute information. It is very useful due to the fact that primitive geometric data types of points, lines and polygons can be stored. This combination of shape data along with their attributes provides the ability for accurate computations.

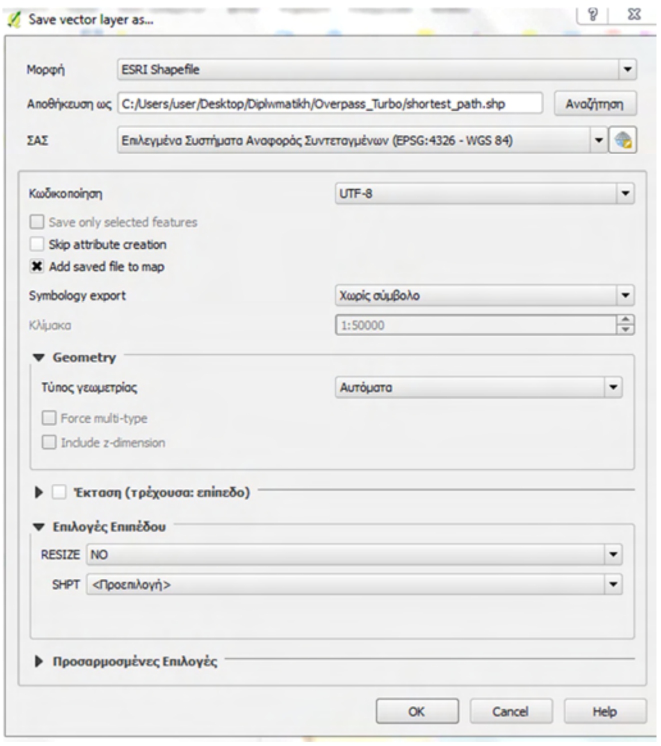
This chapter concludes with the presentation of Fig. 16 and Fig. 17, which sum up the procedure of saving a shortest path as a shapefile.



Όνομα αρχείου: shortest_path

Αποθήκευση ως: ESRI Shapefile [OGR] (*.shp *.SHP)

Fig. 16: Save as shapefile



Save vector layer as...

Μορφή: ESRI Shapefile

Αποθήκευση ως: C:/Users/user/Desktop/Diplwmatikh/Overpass_Turbo/shortest_path.shp

ΣΑΣ: Επιλεγμένα Συστήματα Αναφοράς Συντεταγμένων (EPSG:4326 - WGS 84)

Κωδικοποίηση: UTF-8

☐ Save only selected features

☐ Skip attribute creation

☒ Add saved file to map

Symbology export: Χωρίς σύμβολο

Κλίμακα: 1:50000

Geometry

Τύπος γεωμετρίας: Αυτόματα

☐ Force multi-type

☐ Include z-dimension

☐ Έκταση (τρέχουσα: επίπεδο)

Επιλογές Επιπέδου

RESIZE: NO

SHPT: <Προεπιλογή>

Προσαρμοσμένες Επιλογές

Fig. 17: Save vector layer shapefile

Chapter 5: Application of the TSP Codes for a Real-Life Instance

5.1 The real-life problem for the city of Volos

The mathematical model of the TSP was applied in a real-life instance for the city of Volos. The goal was to obtain an optimal solution for the daily delivery strategy of "Geniki Taxidromiki", which is a courier company that is situated at 56 Lambraki str. The company's clientele includes the fixed number of a hundred customers. The "Fixed-Matrix Code" was executed for the number of 30 nodes. On the other hand, the "Sub-Matrix Code" was applied in the case of 20 nodes. In Fig. 18, the road-network data of Volos along with the 100 nodes, 99 of which constitute the company's clientele are displayed on the OpenStreetMap.

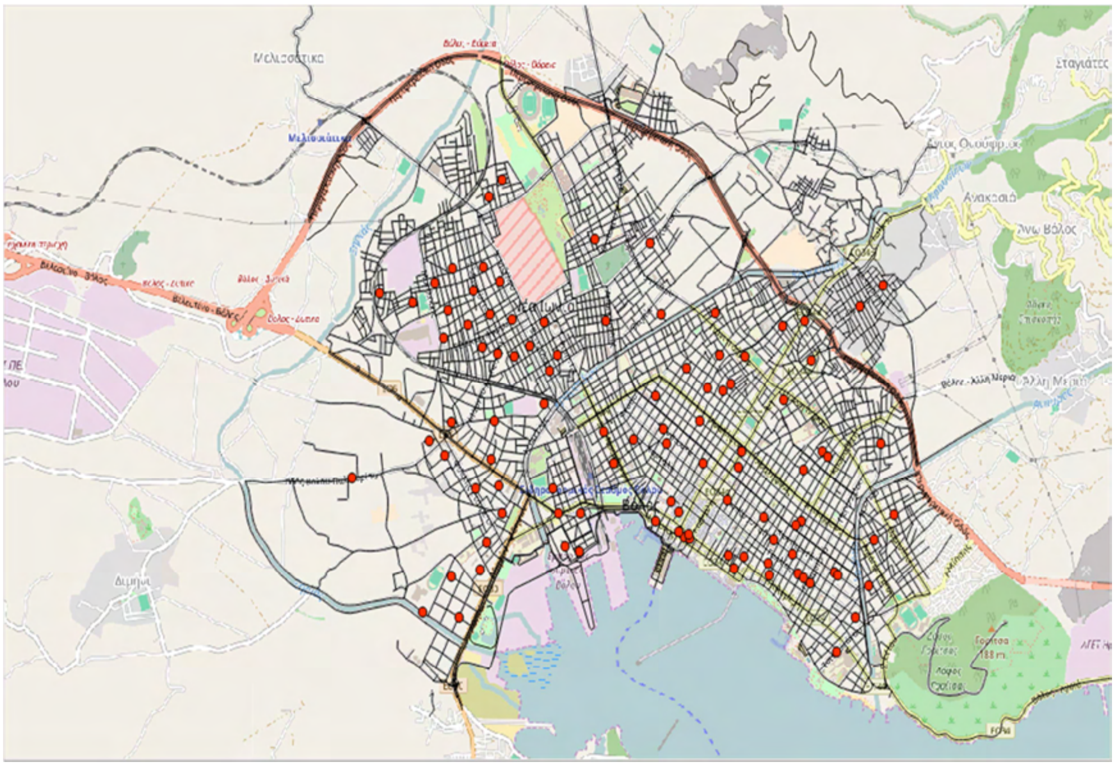


Fig. 18: Clientele of "Geniki Taxidromiki"

5.2 Simulation and results

5.2.1 Simulation

The mathematical model of the TSP was simulated in the programming language of *C*++ by using the CPLEX Optimization Studio. The necessary data for the "Fixed-Matrix Code" that refer to the shortest path distances between all nodes are included in the Appendix at the end of the thesis.

The characteristics of the program as well as the software and hardware of the computer which were used for the solution's procedure are stated below:

Program Edition:

IBM CPLEX Optimization Studio 12.6

Windows Edition:

Windows 7 Home Premium

Operating System:

Processor: AMD E-450 APU with Radeon™ HD Graphics 1.65 *GHz*

Installed memory (RAM): 4.00 *GB*

System type: 64 – bit Operating System

5.2.2 Results

The examples of 30 and 20 nodes respectively were both tested for shortest path distances between the nodes. The results of the two simulations will be examined in the next chapter that will refer to the conclusions that were drawn from their comparison.

At this point, it should be mentioned that in *C*++ the counting of all indexes begins from 0. Therefore, a shift back to the enumeration of nodes is done. For example, the first customer is denoted with number 0 and the starting point is denoted with number 29 and 99 respectively for each problem. The value of 99 occurs due to the creation of a sub-matrix from the complete matrix of nodes, which has a range of 100x100 elements.

5.2.3 Results of the "Fixed-Matrix Code" for 30 nodes

As mentioned above, the "Fixed-Matrix Code" application consists of 29 customers and the starting point. The simulation for this example returns an optimal solution with the value of 22.5 which refers to the travelling distance of the courier van in kilometers.

Fig. 19 displays the nodes of this problem on the OSM.

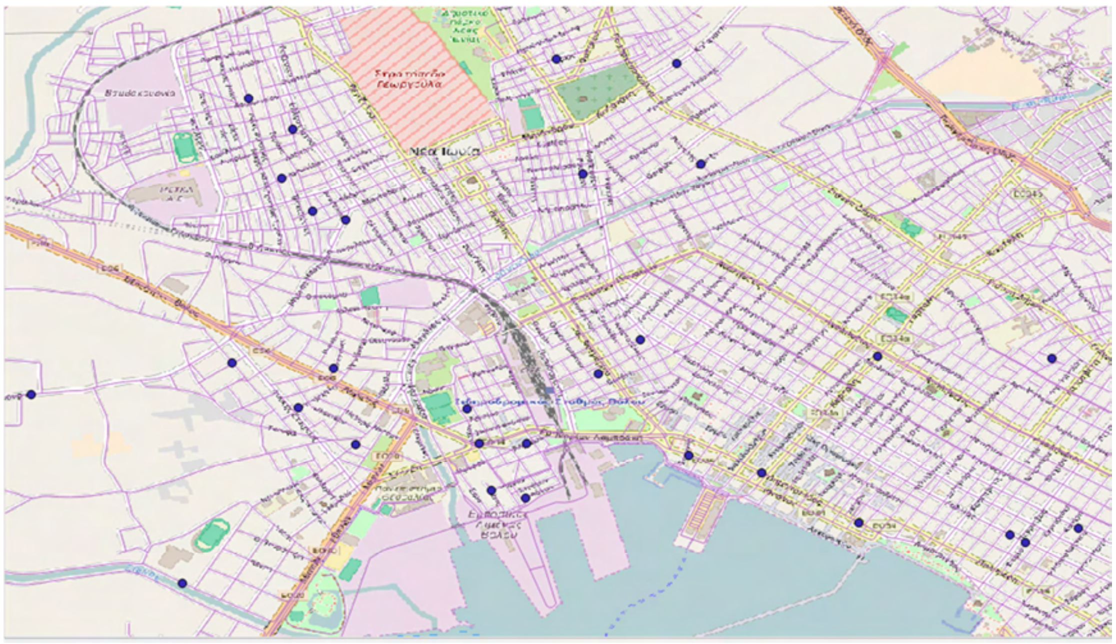


Fig. 19: The 30 nodes of the "Fixed-Matrix Code" example

The optimal routes as well as the optimal order of visit are presented in the following tables.

Order of transition	Transition
1	29 → 3
2	3 → 4
3	4 → 5
4	5 → 6
5	6 → 7
6	7 → 9
7	9 → 8
8	8 → 14
9	14 → 13
10	13 → 12
11	12 → 10
12	10 → 11
13	11 → 17
14	17 → 18
15	18 → 16
16	16 → 15
17	15 → 19
18	19 → 23
19	23 → 25
20	25 → 26
21	26 → 20
22	20 → 21
23	21 → 24
24	24 → 22
25	22 → 27
26	27 → 28
27	28 → 0
28	0 → 1
29	1 → 2
30	2 → 29

Table 1: Optimal routes for “Fixed-Matrix Code” example

Order of visit	Nodes
1	29
2	3
3	4
4	5
5	6
6	7
7	9
8	8
9	14
10	13
11	12
12	10
13	11
14	17
15	18
16	16
17	15
18	19
19	23
20	25
21	26
22	20
23	21
24	24
25	22
26	27
27	28
28	0
29	1
30	2
31	29

Table 2: Order of visit for each node in the “Fixed-Matrix Code” example

This section ends with the demonstration of the optimal solution on the OpenStreetMap, which was based on the visualization procedure analyzed in Chapter 4 (Fig. 20).

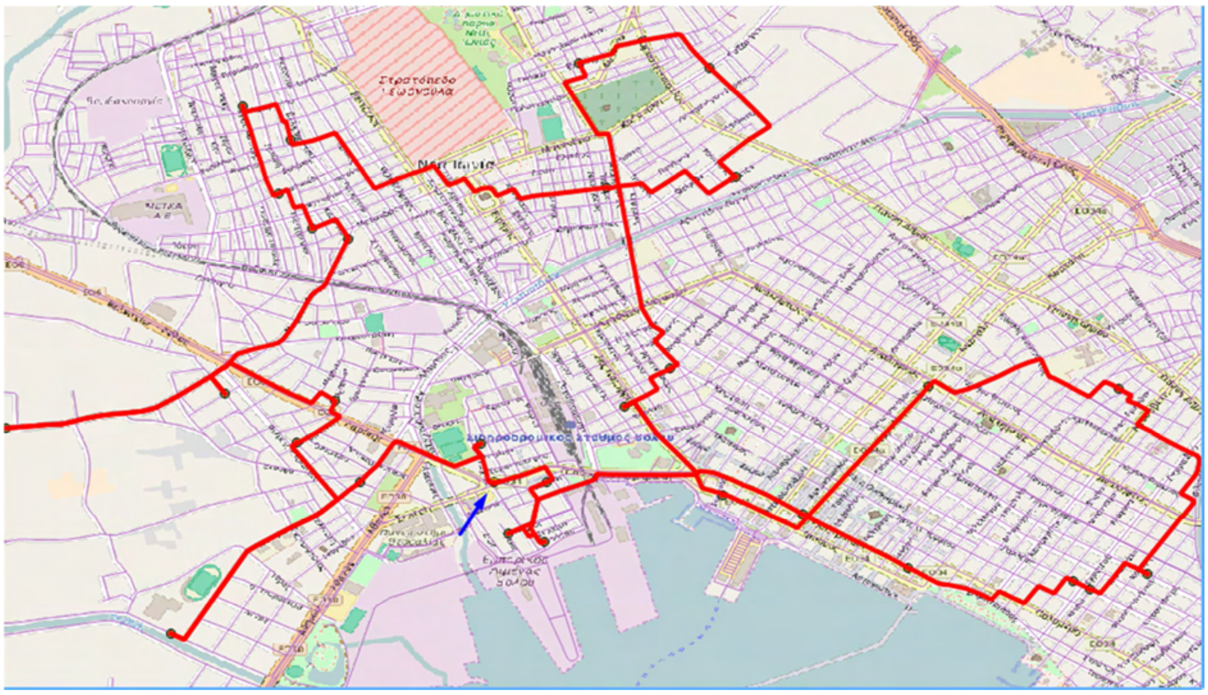


Fig. 20: Visualization of the "Fixed-Matrix Code" example's optimal solution

5.2.4 Results of the "Sub-Matrix Code" for 20 nodes

The "Sub-Matrix Code" application consists of 19 customers and the starting point. The simulation for this example returns an optimal solution with the value of 19.9 which refers to the travelling distance of the courier van in kilometers.

It should be clarified that this code enables the user to select the 20 nodes from a 100x100 matrix, which consists of all the nodes that form the TSP of "Geniki Taxidromiki". Therefore, although this example includes 20 nodes, the values of the nodes have a range from 0 to 99.

Fig. 21 displays the nodes of this problem on the OSM.

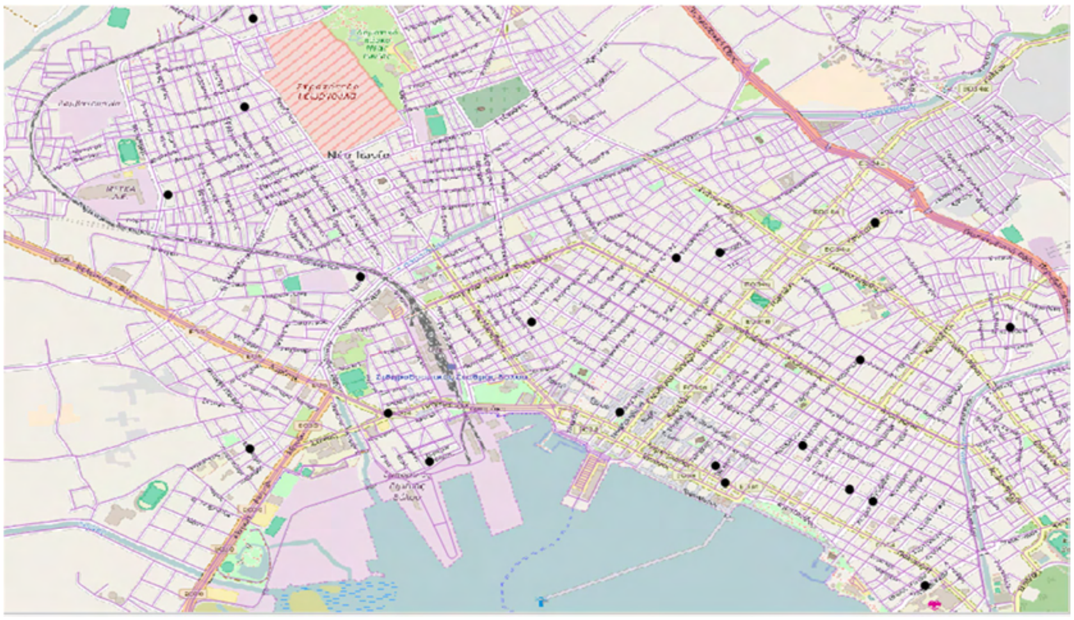


Fig. 21: The 20 nodes of the "Sub-Matrix Code" example

The optimal routes as well as the optimal order of visit are presented in the following tables.

Order of transition	Transition
1	99 → 5
2	5 → 22
3	22 → 38
4	38 → 30
5	30 → 18
6	18 → 44
7	44 → 87
8	87 → 51
9	51 → 81
10	81 → 77
11	77 → 62
12	62 → 89
13	89 → 66
14	66 → 90
15	90 → 73
16	73 → 93

17	93 → 59
18	59 → 96
19	96 → 1
20	1 → 99

Table 3: Optimal routes for "Sub-Matrix Code" example

Order of visit	Nodes
1	99
2	5
3	22
4	38
5	30
6	18
7	44
8	87
9	51
10	81
11	77
12	62
13	89
14	66
15	90
16	73
17	93
18	59
19	96
20	1
21	99

Table 4: Order of visit for each note in the "Sub-Matrix Code" example

This section ends with the demonstration of the optimal solution on the OpenStreetMap, which was based on the visualization procedure analyzed in Chapter 4 (Fig. 22).

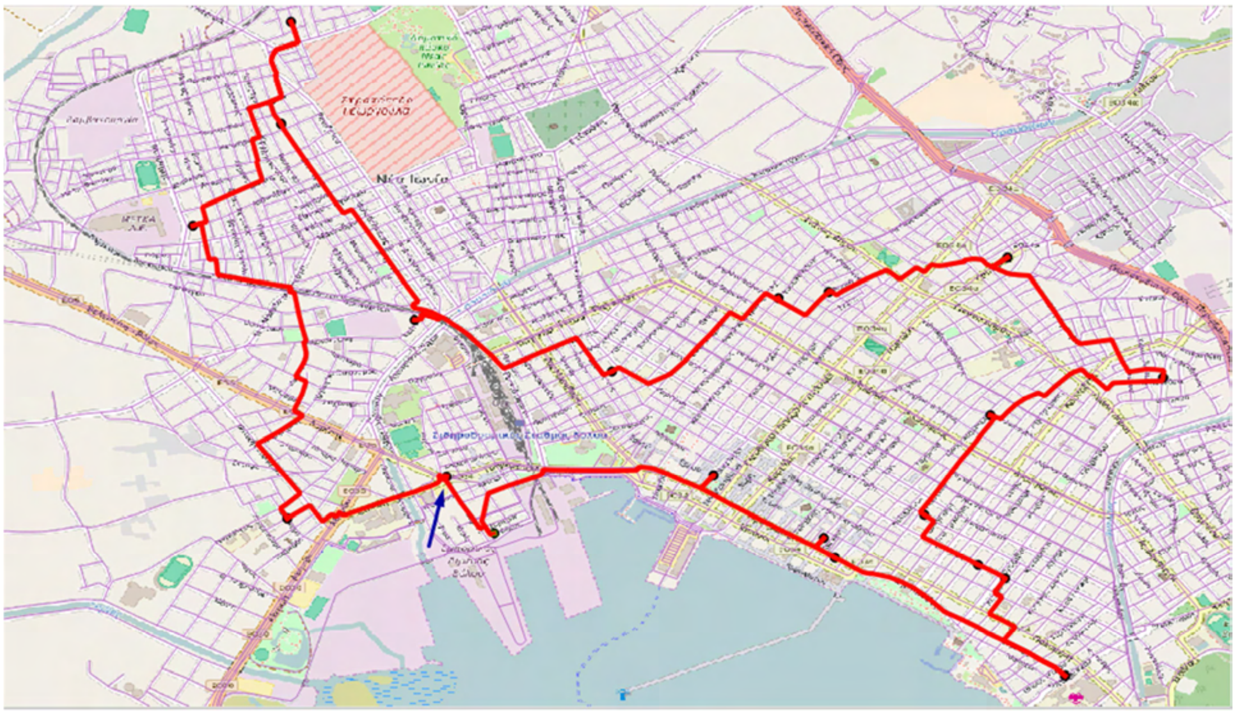


Fig. 22: Visualization of the "Sub-Matrix Code" example's optimal solution

Chapter 6: Conclusions and Future Work Recommendations

6.1 Comparison of the two codes

Having used both source codes for the simulation of two real-life instances of the TSP, it is safe to claim that the “Sub-Matrix Code” is more user friendly than the “Fixed-Matrix Code”. This is due to the fact that, in order for someone to solve many problems including various nodes with the help of the “Fixed-Matrix Code”, the Python code should be executed every time. With the utilization of the “Sub-Matrix Code”, one can overcome this time consuming procedure by selecting a combination of nodes from a large matrix, which includes the whole clientele of the company. The large matrix is created from the Python code that in this case has to be executed only once. As a result, the “Sub-Matrix Code” is the more essential tool that forms a complete application as well as having a better approach to the real-life traveling salesman problem.

6.2 Discussion of the results

The results of the two problems cannot be compared as they refer to 30 and 20 nodes respectively. However, the quality of each solution can be discussed in order for us to be led to some important conclusions. The nature of the TSP is such, that requires an optimal solution the visualization of which, would approach the shape of a circle for the case that the customer-nodes are spread around the starting point. In figure 22, we can observe that the solution of the “Sub-Matrix Code” example with 20 nodes, has a circular shape and as a result it is safe to say that we have obtained a good-quality solution. However, in the case of the “Fixed-Matrix Code” example with 30 nodes, the results arise some questions. This is due to the fact that, as shown in figure 20, two sub-circles are created and lines get crossed. A logical explanation for the results in the above case would be that the combination of the 30 nodes along with the directions of the roads in the network, necessitate the creation of sub-circles. The solution is optimal but it would be quite challenging to persuade a company to follow such a tour.

6.3 Future work recommendations

The procedure followed for the visualization of the results through QGIS can be very time consuming, due to the fact that the shortest path tool can only calculate the shortest path for a couple of nodes. However, QGIS is written in *C*++ and there may be a way to connect this software with the TSP source code in order for the procedure mentioned above to be automated. In addition, the utilization of a different platform could result in a much more user friendly procedure for the visualization of the results.

References

1. **G. Laporte.** The Traveling Salesman Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research.* 1992, 59, pp. 231-247.

2. **D. Davendra.** *Travelling Salesman Problem, Theory and Applications.* s.l. : InTech, 2010.

3. **Hoos H., Stützle T.** *Stochastic Local Search: Foundations and Applications.* s.l. : Elsevier, 2005. pp. 357-416. 8.

4. **G. Pataki.** Teaching Integer Programming Formulations Using the Travelling Salesman Problem. *SLAM Review.* 2003, Vol. 1, 45, pp. 116-123.

Appendix

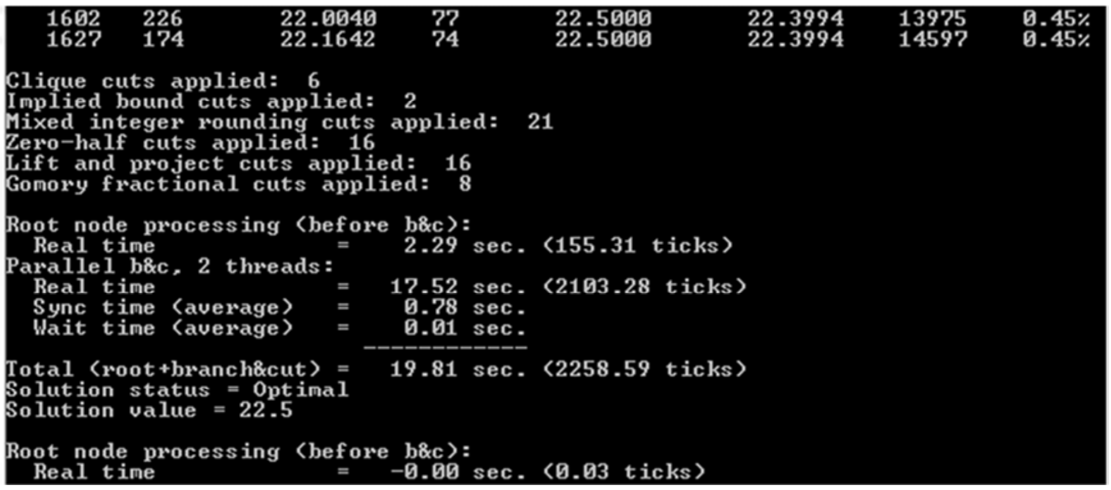
A. Data for “Fixed-Matrix Code” example

0	0.2	0.3	0.9	1.2	2.4	1.3	1.4	1.8	2.5	3.4	3.6	2.6	2.3	2.2	3.1	2.9	2.5	2.6	1.3	3.0	2.9	2.6	1.0	2.6	1.0	2.1	2.0	1.4	0.7
0.2	0	0.4	0.9	1.3	2.5	1.5	1.5	1.9	2.6	3.4	3.6	2.6	2.3	2.2	3.1	2.9	2.4	2.6	1.3	3.0	2.9	2.6	1.0	2.6	1.0	2.1	2.0	1.4	0.6
0.4	0.4	0	0.6	0.9	2.1	1.1	1.1	1.5	2.2	3.0	3.2	2.2	1.9	1.9	2.8	2.5	2.1	2.2	1.0	2.7	2.5	2.3	0.6	2.3	0.6	1.7	1.6	1.0	0.3
0.8	0.8	0.6	0	0.7	1.9	0.9	0.9	1.3	2.0	4.0	3.3	2.3	2.0	1.9	2.7	2.4	2.0	2.1	1.3	3.0	2.9	2.6	1.0	2.6	1.0	2.1	1.9	1.4	0.5
1.3	1.3	0.8	0.9	0	1.0	0.4	0.7	0.8	1.4	2.8	3.0	2.0	1.7	1.6	2.8	2.6	2.1	2.3	1.8	3.5	3.4	3.1	1.5	3.1	1.5	2.6	2.4	1.8	1.4
2.3	2.6	2.2	2.1	1.0	0	1.2	2.2	1.6	2.4	5.2	3.8	2.7	2.4	2.4	4.1	3.9	3.5	3.6	3.1	4.8	4.7	4.4	2.8	4.4	2.8	3.9	3.7	3.2	2.7
1.5	1.5	1.0	1.0	0.4	1.4	0	0.4	0.4	1.3	2.4	2.6	1.6	1.3	1.2	2.7	2.8	2.3	2.5	2.0	3.7	3.6	3.3	1.7	3.3	1.7	2.7	2.6	2.0	1.6
1.7	1.6	1.2	1.4	0.8	1.8	0.5	0	0.7	1.3	2.2	2.4	1.4	1.1	1.1	2.5	2.7	2.2	2.3	2.0	3.9	3.7	3.5	1.8	3.5	1.9	2.9	2.8	2.2	1.6
1.8	1.8	1.4	1.4	0.8	1.8	0.4	0.5	0	0.9	2.1	2.3	1.3	1.0	0.9	2.4	2.4	1.8	2.8	2.3	4.0	3.9	3.6	2.0	3.6	2.0	3.1	2.9	2.4	1.9
2.6	2.6	2.2	2.2	1.4	2.4	1.2	1.3	0.9	0	2.8	3.0	2.0	1.7	1.6	3.1	3.1	2.5	3.6	3.1	4.8	4.7	4.4	2.8	4.4	2.8	3.9	3.7	3.2	2.7
3.1	3.1	2.9	2.6	2.5	3.5	2.5	2.0	2.0	2.7	0	0.7	0.9	1.2	1.5	2.0	3.5	2.0	2.6	2.7	5.7	6.8	4.7	2.7	4.7	3.1	3.8	4.1	3.5	2.9
2.8	2.8	2.6	2.3	2.1	3.1	2.2	1.7	1.7	2.4	0.9	0	0.8	0.6	1.2	2.2	1.9	1.5	2.1	2.2	5.9	4.2	4.3	2.3	4.3	2.7	3.3	3.6	3.0	2.5
3.1	3.1	2.9	2.6	2.4	3.4	2.5	2.0	2.0	2.7	0.5	1.1	0	0.9	1.5	2.5	2.2	1.8	2.5	2.5	6.2	4.5	4.6	2.6	4.6	3.0	3.6	3.9	3.3	2.8
2.2	2.2	2.0	1.7	1.5	2.5	1.6	1.1	1.1	1.8	0.7	1.3	0.4	0	0.6	1.8	1.8	1.4	2.0	2.1	4.0	4.3	4.0	2.1	4.0	2.4	3.2	3.3	2.7	1.9
2.0	2.0	1.8	1.5	1.4	2.4	1.4	0.9	0.9	1.6	1.3	1.4	0.8	0.2	0	1.5	1.5	1.2	1.7	1.9	3.8	4.1	3.8	1.9	3.8	2.2	3.0	3.1	2.6	1.7
3.0	3.1	2.8	2.5	2.6	3.6	2.8	2.5	2.5	3.2	2.1	2.1	1.9	1.7	2.0	0	0.7	1.0	1.1	1.9	4.8	3.7	3.7	2.0	6.3	2.4	2.5	3.3	2.7	2.9
3.0	3.2	2.8	2.6	2.7	3.7	3.0	2.6	8.0	8.6	2.2	2.2	2.0	1.8	2.1	0.7	0	1.0	0.7	2.0	3.5	4.6	4.9	2.0	5.0	2.4	3.7	3.3	2.8	2.9
2.3	2.5	2.1	2.0	2.1	3.1	2.3	2.0	2.0	2.6	2.0	2.0	1.9	1.5	1.4	1.0	0.7	0	0.6	1.3	2.6	3.1	3.1	1.3	3.8	1.7	1.9	2.7	2.1	2.2
2.7	2.8	2.5	2.4	2.5	3.5	2.7	2.4	2.4	3.1	2.5	2.6	2.4	2.0	1.9	1.1	0.7	0.9	0	1.6	1.9	3.3	3.3	1.7	4.0	2.1	2.2	3.0	2.4	2.6
1.7	1.9	1.6	1.8	2.3	3.5	2.5	2.5	2.9	3.6	2.9	2.9	2.2	1.9	1.9	1.9	1.9	1.1	1.3	0	2.1	2.1	2.1	0.7	3.3	1.1	1.2	2.1	1.5	1.7
3.0	3.2	2.9	3.1	3.6	4.8	3.8	3.8	4.2	4.9	6.0	7.8	4.1	3.8	3.7	4.9	3.6	2.9	1.9	2.9	0	1.2	1.1	3.0	1.9	2.4	1.8	1.3	1.8	3.0
3.0	3.2	2.9	3.1	3.6	4.8	3.8	3.8	4.2	4.9	5.4	5.4	4.8	4.5	4.4	4.3	5.0	3.6	3.6	2.9	1.7	0	0.8	2.7	0.5	2.4	1.8	1.4	1.8	3.0
2.4	2.6	2.3	2.5	3.0	4.2	3.2	3.2	3.6	4.3	4.7	4.7	4.0	3.7	3.7	3.6	4.7	2.9	3.4	2.3	1.4	0.4	0	2.1	0.072	1.8	1.4	0.8	1.2	2.4
1.2	1.3	1.0	1.3	1.7	2.9	1.9	1.9	2.4	3.0	2.9	3.1	2.1	1.8	1.7	2.3	2.1	1.5	1.8	1.0	2.6	2.5	2.2	0	2.2	0.6	1.7	1.5	1.0	1.1
2.6	2.8	2.4	2.7	3.2	4.3	3.3	3.4	3.8	4.5	4.6	4.7	4.2	3.9	3.8	3.6	4.7	2.9	3.1	2.4	1.4	0.4	0.3	2.2	0	1.9	1.4	0.9	1.4	2.5
1.1	1.3	0.9	1.2	1.7	2.8	1.8	1.9	2.3	3.0	3.5	3.5	2.7	2.4	2.3	2.8	2.5	1.9	2.2	0.9	2.0	1.9	1.6	0.7	1.6	0	1.1	0.9	0.4	1.0
2.0	2.2	1.9	2.1	2.6	3.8	2.8	2.8	3.2	3.9	3.7	3.7	3.1	2.8	2.7	2.6	3.2	1.9	2.0	2.0	0.9	1.4	1.4	2.0	2.1	1.4	0	1.6	1.1	2.0
1.7	1.8	1.5	1.8	2.2	3.4	2.4	2.4	2.9	3.5	4.1	4.1	3.2	2.9	2.9	3.3	4.4	2.5	2.8	1.5	1.4	1.3	1.3	1.3	1.2	1.0	1.1	0	0.4	1.6
1.2	1.4	1.1	1.3	1.8	3.0	2.0	2.0	2.4	3.1	3.6	3.6	2.8	2.5	2.4	2.9	2.7	2.1	2.4	1.0	1.9	1.8	1.6	0.8	1.6	0.6	1.0	0.9	0	1.1
0.9	0.9	0.7	0.3	0.6	1.8	0.8	0.9	1.3	1.9	4.0	3.2	2.2	1.9	1.8	2.7	2.5	2.0	2.2	1.9	3.1	3.0	2.7	1.1	2.7	1.1	2.2	2.1	1.5	0

Fig. 23: Shortest path distances for “Fixed-Matrix Code” example

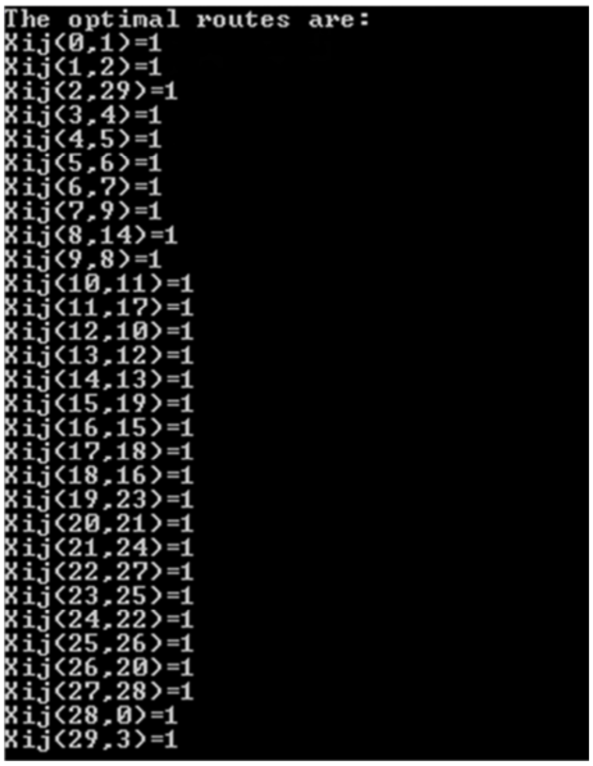
B. Codes' results

"Fixed-Matrix Code" results



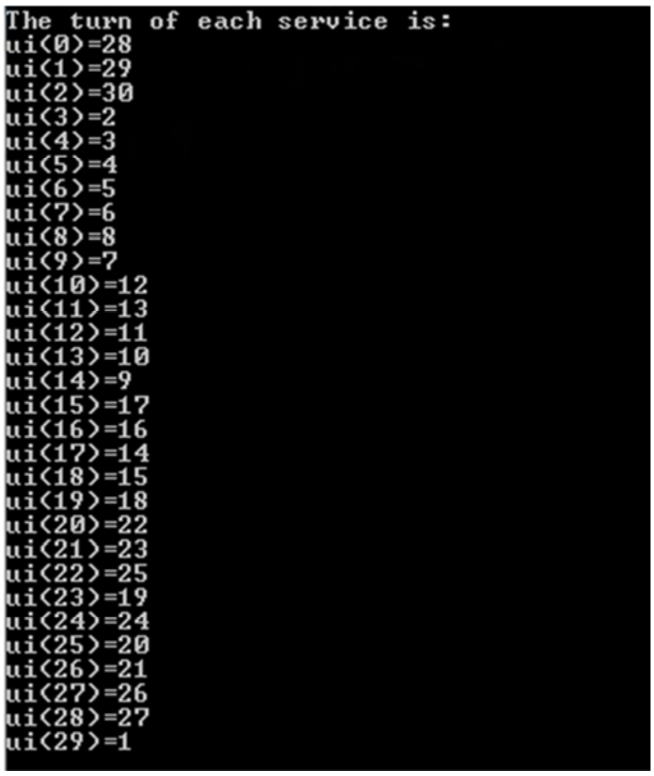
```
   1602   226         22.0040    77         22.5000        22.3994    13975      0.45%
   1627   174         22.1642    74         22.5000        22.3994    14597      0.45%

Clique cuts applied:  6
Implied bound cuts applied:  2
Mixed integer rounding cuts applied:  21
Zero-half cuts applied:  16
Lift and project cuts applied:  16
Gomory fractional cuts applied:  8

Root node processing (before b&c):
  Real time             =     2.29 sec. (155.31 ticks)
Parallel b&c, 2 threads:
  Real time             =    17.52 sec. (2103.28 ticks)
  Sync time (average)   =     0.78 sec.
  Wait time (average)   =     0.01 sec.
                          ------------
Total (root+branch&cut) =    19.81 sec. (2258.59 ticks)
Solution status = Optimal
Solution value = 22.5

Root node processing (before b&c):
  Real time             =    -0.00 sec. (0.03 ticks)
```

Fig. 24: Optimal solution value for the "Fixed-Matrix Code" example



```
The optimal routes are:
Xij(0,1)=1
Xij(1,2)=1
Xij(2,29)=1
Xij(3,4)=1
Xij(4,5)=1
Xij(5,6)=1
Xij(6,7)=1
Xij(7,9)=1
Xij(8,14)=1
Xij(9,8)=1
Xij(10,11)=1
Xij(11,17)=1
Xij(12,10)=1
Xij(13,12)=1
Xij(14,13)=1
Xij(15,19)=1
Xij(16,15)=1
Xij(17,18)=1
Xij(18,16)=1
Xij(19,23)=1
Xij(20,21)=1
Xij(21,24)=1
Xij(22,27)=1
Xij(23,25)=1
Xij(24,22)=1
Xij(25,26)=1
Xij(26,20)=1
Xij(27,28)=1
Xij(28,0)=1
Xij(29,3)=1
```

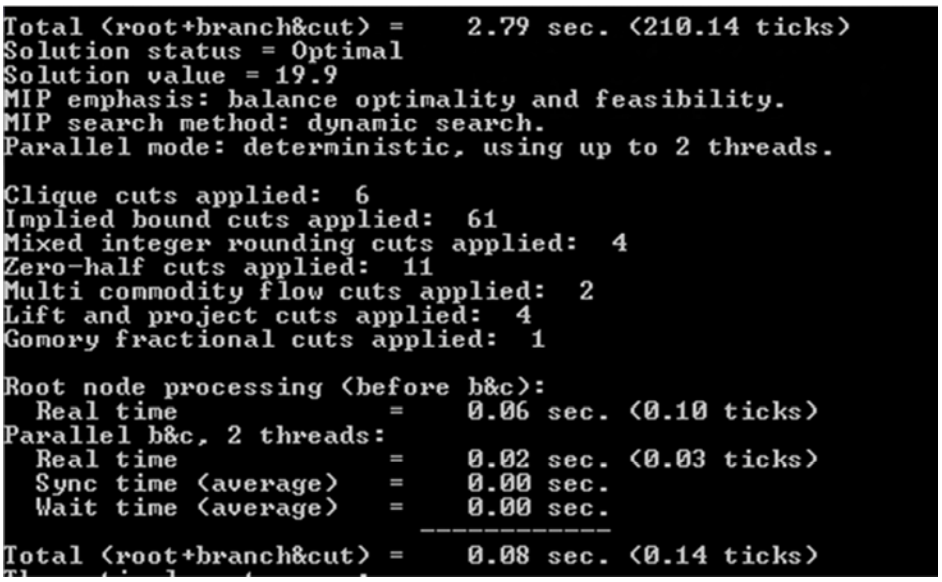
Fig. 25: Optimal routes for the "Fixed-Matrix Code" example



```
The turn of each service is:
ui(0)=28
ui(1)=29
ui(2)=30
ui(3)=2
ui(4)=3
ui(5)=4
ui(6)=5
ui(7)=6
ui(8)=8
ui(9)=7
ui(10)=12
ui(11)=13
ui(12)=11
ui(13)=10
ui(14)=9
ui(15)=17
ui(16)=16
ui(17)=14
ui(18)=15
ui(19)=18
ui(20)=22
ui(21)=23
ui(22)=25
ui(23)=19
ui(24)=24
ui(25)=20
ui(26)=21
ui(27)=26
ui(28)=27
ui(29)=1
```

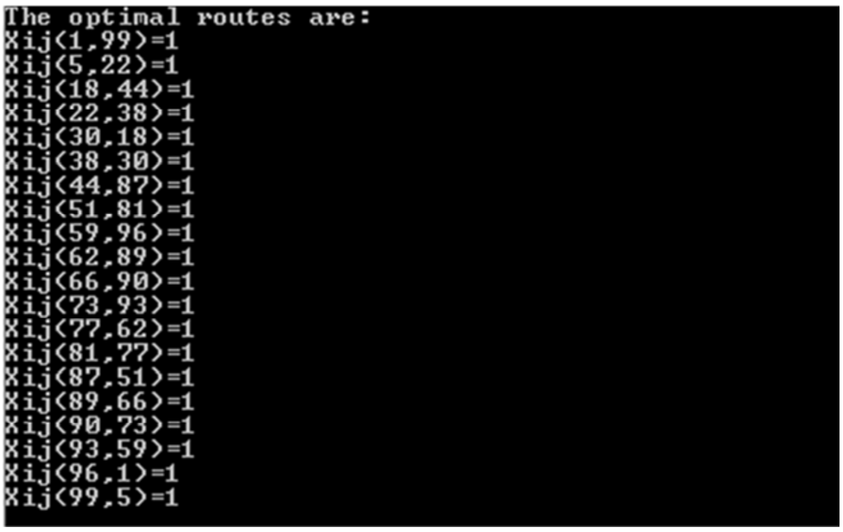
Fig. 26: Optimal order of visit for the “Fixed-Matrix Code” example

“Sub-Matrix Code” results



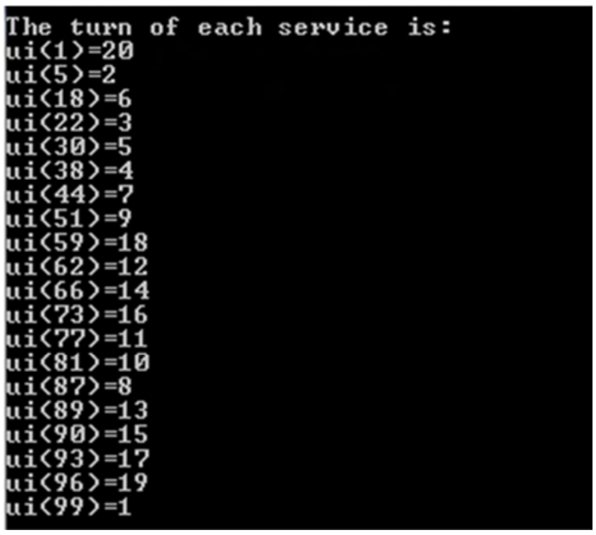
```
Total (root+branch&cut) =    2.79 sec. (210.14 ticks)
Solution status = Optimal
Solution value = 19.9
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 2 threads.

Clique cuts applied:  6
Implied bound cuts applied:  61
Mixed integer rounding cuts applied:  4
Zero-half cuts applied:  11
Multi commodity flow cuts applied:  2
Lift and project cuts applied:  4
Gomory fractional cuts applied:  1

Root node processing (before b&c):
  Real time             =    0.06 sec. (0.10 ticks)
Parallel b&c, 2 threads:
  Real time             =    0.02 sec. (0.03 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.08 sec. (0.14 ticks)
```

Fig. 27: Optimal solution value for the “Sub-Matrix Code” example



```
The optimal routes are:
Xij(1,99)=1
Xij(5,22)=1
Xij(18,44)=1
Xij(22,38)=1
Xij(30,18)=1
Xij(38,30)=1
Xij(44,87)=1
Xij(51,81)=1
Xij(59,96)=1
Xij(62,89)=1
Xij(66,90)=1
Xij(73,93)=1
Xij(77,62)=1
Xij(81,77)=1
Xij(87,51)=1
Xij(89,66)=1
Xij(90,73)=1
Xij(93,59)=1
Xij(96,1)=1
Xij(99,5)=1
```

Fig. 28: Optimal routes for the "Sub-Matrix Code" example



```
The turn of each service is:
ui(1)=20
ui(5)=2
ui(18)=6
ui(22)=3
ui(30)=5
ui(38)=4
ui(44)=7
ui(51)=9
ui(59)=18
ui(62)=12
ui(66)=14
ui(73)=16
ui(77)=11
ui(81)=10
ui(87)=8
ui(89)=13
ui(90)=15
ui(93)=17
ui(96)=19
ui(99)=1
```

Fig. 29: Optimal order of visit for the "Sub-Matrix Code" example

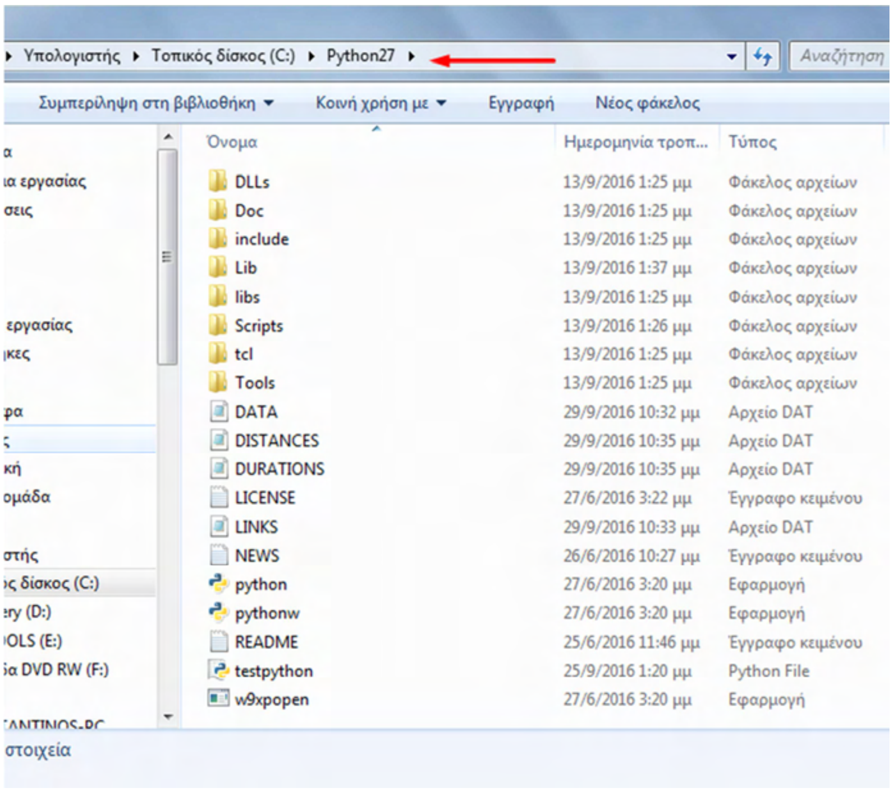
Application's Manual

In this section, all the necessary steps that the user must follow in order to solve a real-life TSP will be analyzed.

Once the coordinates of the nodes that the problem includes are obtained, the user must create the shortest path distance matrix, which is the data in order for the TSP code, written in *C++*, to be executed. The distance matrix is created through the Python code that takes advantage of the services provided by Google's API. As a result, the first step would be to download the version 2.7.0 of Python along with the 2.7.0 setup tools from the link below:

https://www.python.org/download/releases/2.7/

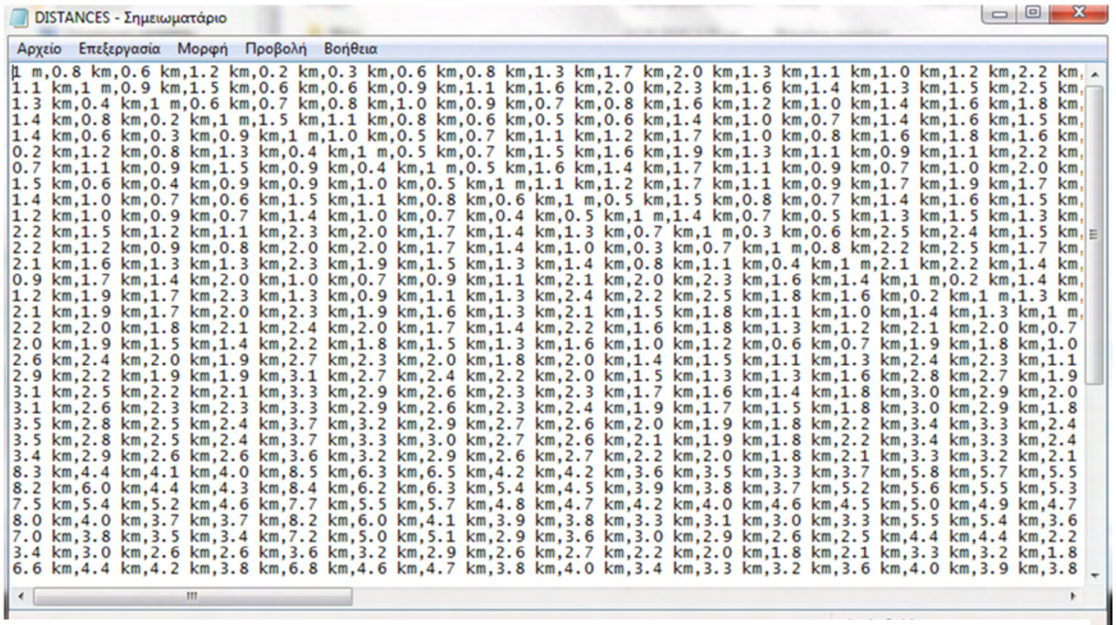
The next step is to properly organize the folder of Python, which is created in the hard disk (C:) of the computer in order for the code to be executed. The needed structure of the folder is depicted in Fig. 30:


Υπολογιστής ▸ Τοπικός δίσκος (C:) ▸ Python27 ▸
Αναζήτηση
Συμπερίληψη στη βιβλιοθήκη
Κοινή χρήση με
Εγγραφή
Νέος φάκελος
στοιχεία

Όνομα	Ημερομηνία τροπ...	Τύπος
DLLs	13/9/2016 1:25 μμ	Φάκελος αρχείων
Doc	13/9/2016 1:25 μμ	Φάκελος αρχείων
include	13/9/2016 1:25 μμ	Φάκελος αρχείων
Lib	13/9/2016 1:37 μμ	Φάκελος αρχείων
libs	13/9/2016 1:25 μμ	Φάκελος αρχείων
Scripts	13/9/2016 1:26 μμ	Φάκελος αρχείων
tcl	13/9/2016 1:25 μμ	Φάκελος αρχείων
Tools	13/9/2016 1:25 μμ	Φάκελος αρχείων
DATA	29/9/2016 10:32 μμ	Αρχείο DAT
DISTANCES	29/9/2016 10:35 μμ	Αρχείο DAT
DURATIONS	29/9/2016 10:35 μμ	Αρχείο DAT
LICENSE	27/6/2016 3:22 μμ	Έγγραφο κειμένου
LINKS	29/9/2016 10:33 μμ	Αρχείο DAT
NEWS	26/6/2016 10:27 μμ	Έγγραφο κειμένου
python	27/6/2016 3:20 μμ	Εφαρμογή
pythonw	27/6/2016 3:20 μμ	Εφαρμογή
README	25/6/2016 11:46 μμ	Έγγραφο κειμένου
testpython	25/9/2016 1:20 μμ	Python File
w9xpopen	27/6/2016 3:20 μμ	Εφαρμογή

Fig. 30: Python folder

The file named "DATA" should include the coordinates of all the nodes with that of the starting point inserted last and the "testpython" file must include the Python code. In order for the user to obtain the distance matrix in the “DISTANCES” file, the command prompt must be opened through this Python file. In the command prompt, one must first type “easy_install simplejson” and press Enter. Finally, “testpython” should be typed and by then pressing Enter the distance matrix will be created (Fig.31).

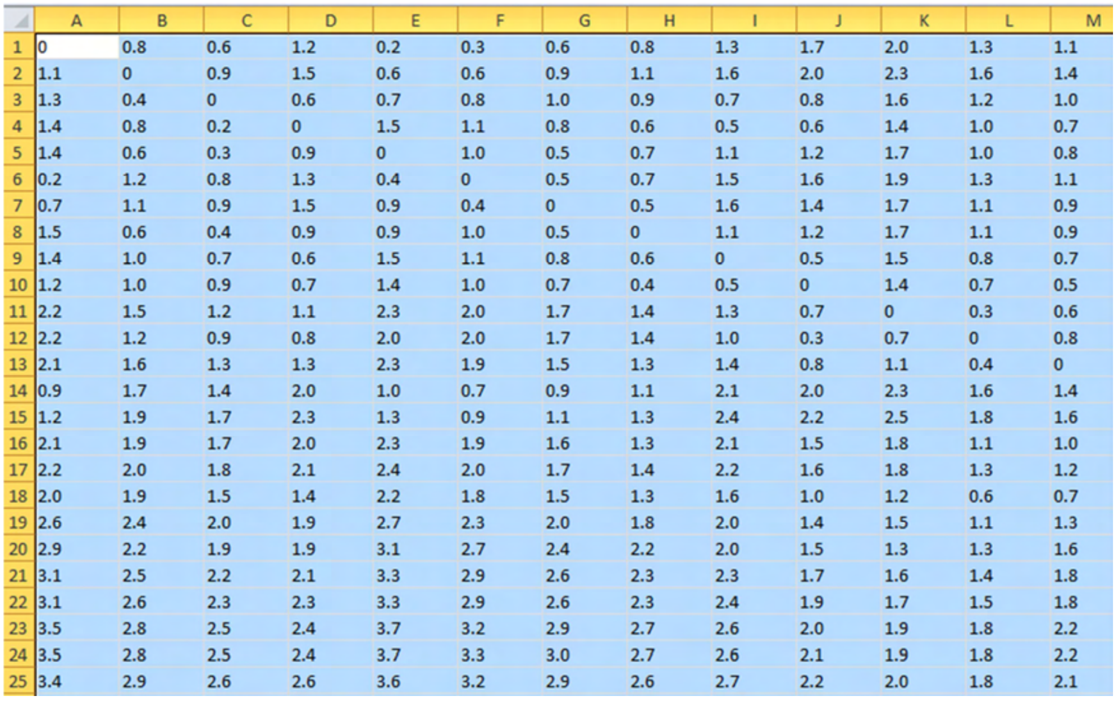


```
1 m,0.8 km,0.6 km,1.2 km,0.2 km,0.3 km,0.6 km,0.8 km,1.3 km,1.7 km,2.0 km,1.3 km,1.1 km,1.0 km,1.2 km,2.2 km,
1.1 km,1 m,0.9 km,1.5 km,0.6 km,0.6 km,0.9 km,1.1 km,1.6 km,2.0 km,2.3 km,1.6 km,1.4 km,1.3 km,1.5 km,2.5 km,
1.3 km,0.4 km,1 m,0.6 km,0.7 km,0.8 km,1.0 km,0.9 km,0.7 km,0.8 km,1.6 km,1.2 km,1.0 km,1.4 km,1.6 km,1.8 km,
1.4 km,0.8 km,0.2 km,1 m,1.5 km,1.1 km,0.8 km,0.6 km,0.5 km,0.6 km,1.4 km,1.0 km,0.7 km,1.4 km,1.6 km,1.5 km,
1.4 km,0.6 km,0.3 km,0.9 km,1 m,1.0 km,0.5 km,0.7 km,1.1 km,1.2 km,1.7 km,1.0 km,0.8 km,1.6 km,1.8 km,1.6 km,
0.2 km,1.2 km,0.8 km,1.3 km,0.4 km,1 m,0.5 km,0.7 km,1.5 km,1.6 km,1.9 km,1.3 km,1.1 km,0.9 km,1.1 km,2.2 km,
0.7 km,1.1 km,0.9 km,1.5 km,0.9 km,0.4 km,1 m,0.5 km,1.6 km,1.4 km,1.7 km,1.1 km,0.9 km,0.7 km,1.0 km,2.0 km,
1.5 km,0.6 km,0.4 km,0.9 km,0.9 km,1.0 km,0.5 km,1 m,1.1 km,1.2 km,1.7 km,1.1 km,0.9 km,1.7 km,1.9 km,1.7 km,
1.4 km,1.0 km,0.7 km,0.6 km,1.5 km,1.1 km,0.8 km,0.6 km,1 m,0.5 km,1.5 km,0.8 km,0.7 km,1.4 km,1.6 km,1.5 km,
1.2 km,1.0 km,0.9 km,0.7 km,1.4 km,1.0 km,0.7 km,0.4 km,0.5 km,1 m,1.4 km,0.7 km,0.5 km,1.3 km,1.5 km,1.3 km,
2.2 km,1.5 km,1.2 km,1.1 km,2.3 km,2.0 km,1.7 km,1.4 km,1.3 km,0.7 km,1 m,0.3 km,0.6 km,2.5 km,2.4 km,1.5 km,
2.2 km,1.2 km,0.9 km,0.8 km,2.0 km,2.0 km,1.7 km,1.4 km,1.0 km,0.3 km,0.7 km,1 m,0.8 km,2.2 km,2.5 km,1.7 km,
2.1 km,1.6 km,1.3 km,1.3 km,2.3 km,1.9 km,1.5 km,1.3 km,1.4 km,0.8 km,1.1 km,0.4 km,1 m,2.1 km,2.2 km,1.4 km,
0.9 km,1.7 km,1.4 km,2.0 km,1.0 km,0.7 km,0.9 km,1.1 km,2.1 km,2.0 km,2.3 km,1.6 km,1.4 km,1 m,0.2 km,1.4 km,
1.2 km,1.9 km,1.7 km,2.3 km,1.3 km,0.9 km,1.1 km,1.3 km,2.4 km,2.2 km,2.5 km,1.8 km,1.6 km,0.2 km,1 m,1.3 km,
2.1 km,1.9 km,1.7 km,2.0 km,2.3 km,1.9 km,1.6 km,1.3 km,2.1 km,1.5 km,1.8 km,1.1 km,1.0 km,1.4 km,1.3 km,1 m,
2.2 km,2.0 km,1.8 km,2.1 km,2.4 km,2.0 km,1.7 km,1.4 km,2.2 km,1.6 km,1.8 km,1.3 km,1.2 km,2.1 km,2.0 km,0.7
2.0 km,1.9 km,1.5 km,1.4 km,2.2 km,1.8 km,1.5 km,1.3 km,1.6 km,1.0 km,1.2 km,0.6 km,0.7 km,1.9 km,1.8 km,1.0
2.6 km,2.4 km,2.0 km,1.9 km,2.7 km,2.3 km,2.0 km,1.8 km,2.0 km,1.4 km,1.5 km,1.1 km,1.3 km,2.4 km,2.3 km,1.1
2.9 km,2.2 km,1.9 km,1.9 km,3.1 km,2.7 km,2.4 km,2.2 km,2.0 km,1.5 km,1.3 km,1.3 km,1.6 km,2.8 km,2.7 km,1.9
3.1 km,2.5 km,2.2 km,2.1 km,3.3 km,2.9 km,2.6 km,2.3 km,2.3 km,1.7 km,1.6 km,1.4 km,1.8 km,3.0 km,2.9 km,2.0
3.1 km,2.6 km,2.3 km,2.3 km,3.3 km,2.9 km,2.6 km,2.3 km,2.4 km,1.9 km,1.7 km,1.5 km,1.8 km,3.0 km,2.9 km,1.8
3.5 km,2.8 km,2.5 km,2.4 km,3.7 km,3.2 km,2.9 km,2.7 km,2.6 km,2.0 km,1.9 km,1.8 km,2.2 km,3.4 km,3.3 km,2.4
3.5 km,2.8 km,2.5 km,2.4 km,3.7 km,3.3 km,3.0 km,2.7 km,2.6 km,2.1 km,1.9 km,1.8 km,2.2 km,3.4 km,3.3 km,2.4
3.4 km,2.9 km,2.6 km,2.6 km,3.6 km,3.2 km,2.9 km,2.6 km,2.7 km,2.2 km,2.0 km,1.8 km,2.1 km,3.3 km,3.2 km,2.1
8.3 km,4.4 km,4.1 km,4.0 km,8.5 km,6.3 km,6.5 km,4.2 km,4.2 km,3.6 km,3.5 km,3.3 km,3.7 km,5.8 km,5.7 km,5.5
8.2 km,6.0 km,4.4 km,4.3 km,8.4 km,6.2 km,6.3 km,5.4 km,4.5 km,3.9 km,3.8 km,3.7 km,5.2 km,5.6 km,5.5 km,5.3
7.5 km,5.4 km,5.2 km,4.6 km,7.7 km,5.5 km,5.7 km,4.8 km,4.7 km,4.2 km,4.0 km,4.6 km,4.5 km,5.0 km,4.9 km,4.7
8.0 km,4.0 km,3.7 km,3.7 km,8.2 km,6.0 km,4.1 km,3.9 km,3.8 km,3.3 km,3.1 km,3.0 km,3.3 km,5.5 km,5.4 km,3.6
7.0 km,3.8 km,3.5 km,3.4 km,7.2 km,5.0 km,5.1 km,2.9 km,3.6 km,3.0 km,2.9 km,2.6 km,2.5 km,4.4 km,4.4 km,2.2
3.4 km,3.0 km,2.6 km,2.6 km,3.6 km,3.2 km,2.9 km,2.6 km,2.7 km,2.2 km,2.0 km,1.8 km,2.1 km,3.3 km,3.2 km,1.8
6.6 km,4.4 km,4.2 km,3.8 km,6.8 km,4.6 km,4.7 km,3.8 km,4.0 km,3.4 km,3.3 km,3.2 km,3.6 km,4.0 km,3.9 km,3.8
```

Fig. 31: Shortest path distance matrix

However, the produced shortest path distances need some modifications in order to be utilized as data for the TSP code. The next figure displays the results of the Python code.

The above file must be transferred into a Microsoft Excel worksheet in order for the distance units to be cleared. In addition, all the diagonal elements of the matrix must be set equal to zero. After the above modifications are completed, the matrix must be saved as a .txt file. The results of the modifications in Microsoft Excel are shown in Fig. 32:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	0	0.8	0.6	1.2	0.2	0.3	0.6	0.8	1.3	1.7	2.0	1.3	1.1
2	1.1	0	0.9	1.5	0.6	0.6	0.9	1.1	1.6	2.0	2.3	1.6	1.4
3	1.3	0.4	0	0.6	0.7	0.8	1.0	0.9	0.7	0.8	1.6	1.2	1.0
4	1.4	0.8	0.2	0	1.5	1.1	0.8	0.6	0.5	0.6	1.4	1.0	0.7
5	1.4	0.6	0.3	0.9	0	1.0	0.5	0.7	1.1	1.2	1.7	1.0	0.8
6	0.2	1.2	0.8	1.3	0.4	0	0.5	0.7	1.5	1.6	1.9	1.3	1.1
7	0.7	1.1	0.9	1.5	0.9	0.4	0	0.5	1.6	1.4	1.7	1.1	0.9
8	1.5	0.6	0.4	0.9	0.9	1.0	0.5	0	1.1	1.2	1.7	1.1	0.9
9	1.4	1.0	0.7	0.6	1.5	1.1	0.8	0.6	0	0.5	1.5	0.8	0.7
10	1.2	1.0	0.9	0.7	1.4	1.0	0.7	0.4	0.5	0	1.4	0.7	0.5
11	2.2	1.5	1.2	1.1	2.3	2.0	1.7	1.4	1.3	0.7	0	0.3	0.6
12	2.2	1.2	0.9	0.8	2.0	2.0	1.7	1.4	1.0	0.3	0.7	0	0.8
13	2.1	1.6	1.3	1.3	2.3	1.9	1.5	1.3	1.4	0.8	1.1	0.4	0
14	0.9	1.7	1.4	2.0	1.0	0.7	0.9	1.1	2.1	2.0	2.3	1.6	1.4
15	1.2	1.9	1.7	2.3	1.3	0.9	1.1	1.3	2.4	2.2	2.5	1.8	1.6
16	2.1	1.9	1.7	2.0	2.3	1.9	1.6	1.3	2.1	1.5	1.8	1.1	1.0
17	2.2	2.0	1.8	2.1	2.4	2.0	1.7	1.4	2.2	1.6	1.8	1.3	1.2
18	2.0	1.9	1.5	1.4	2.2	1.8	1.5	1.3	1.6	1.0	1.2	0.6	0.7
19	2.6	2.4	2.0	1.9	2.7	2.3	2.0	1.8	2.0	1.4	1.5	1.1	1.3
20	2.9	2.2	1.9	1.9	3.1	2.7	2.4	2.2	2.0	1.5	1.3	1.3	1.6
21	3.1	2.5	2.2	2.1	3.3	2.9	2.6	2.3	2.3	1.7	1.6	1.4	1.8
22	3.1	2.6	2.3	2.3	3.3	2.9	2.6	2.3	2.4	1.9	1.7	1.5	1.8
23	3.5	2.8	2.5	2.4	3.7	3.2	2.9	2.7	2.6	2.0	1.9	1.8	2.2
24	3.5	2.8	2.5	2.4	3.7	3.3	3.0	2.7	2.6	2.1	1.9	1.8	2.2
25	3.4	2.9	2.6	2.6	3.6	3.2	2.9	2.6	2.7	2.2	2.0	1.8	2.1

Fig. 32: Modifications in Microsoft Excel

Having the modified distance matrix in a .txt file, one can now proceed with the solution of the real-life TSP. The simulation of the TSP code, written in *C++*, is accomplished with the utilization of IBM CPLEX Optimization Studio 12.6. Both the data file as well as the TSP code in the form of "V*C++* Project" must coexist in the "stat_mda" file that the user can find by following the path presented below:

Hard Disk (C:) → Program Files (x86) → IBM → ILOG → CPLEX_Studio_Preview126 → cplex → examples → x86_windows_vs2010 → stat_mda

Once the above steps are completed, the user will be able to execute the TSP code and obtain the solution that occurs from the simulation.

The visualization process of the optimal routes of the solution is analyzed in Chapter 4 of the thesis. In order to make this procedure even clearer for the user, it is going to be approached in a step-by-step way below:

- Download QGIS 2.14 from http://www.qgis.org/en/site/forusers/download.html,
- Download the "OpenLayersPlugin" extension from the interface of QGIS,
- Display the map of OSM in QGIS,
- Execute the XML code in the platform of Overpass turbo for a specific region,
- Export the road network data as geoJSON,
- Import the road network data as a vector layer on the OpenStreetMap,
- Configure the road graph extension of QGIS,

- Calculate the shortest path distance for every route of the optimal solution by utilizing the shortest path tool of QGIS,
- Save every calculated shortest path as a shapefile,
- Clear the map of OSM from all layers,
- Import all the shapefiles of the solution on the map.