

# SMART and FLEXible mobile DATA COLLECTOR for GIS

(Acronym, MOBILO)

ENTERPRISES 0916/0055

## [D12] Report on Network Covering Algorithm

<b>Deliverable n.</b>	D12	<b>Deliverable title</b>	Report on Network Covering Algorithm
<b>Workpackage</b>	WP6	<b>WP title</b>	Route Planning – data gathering GUI
<b>Editors</b>	Elias Frentzos (GEO), Dimitrios Skarlatos (CUT)		
<b>Contributors</b>	Elias Frentzos (GEO), Dimitrios Skarlatos (CUT), Petros Katsikadacos (GEO),		
<b>Status</b>	Final		
<b>Distribution</b>	Public		
<b>Issue date</b>	2020-07-31	<b>Creation date</b>	2020-07-01

## Contents

LIST OF FIGURES .....	3
LIST OF TABLES .....	4
LIST OF ABBREVIATIONS.....	5
REVISION CHART AND HISTORY LOG.....	6
1 Introduction.....	8
2 Chinese Postman Algorithm .....	8
3 Variations.....	11
3.1 Different starting and ending vertices.....	11
3.2 Chinese Postman on directed graphs.....	11
3.3 Rural postman problem.....	13
4 Implementation.....	13
5 Conclusion .....	15
6 References .....	17

## LIST OF FIGURES

Figure 1: An example road network graph.....	8
Figure 2: Vertex and graph Order.....	9
Figure 3: Artificial graph edge addition to make the graph Eulerian .....	10
Figure 4: Vertex Degree in a directed graph .....	12
Figure 5: Adding artificial edges in in a directed graph.....	12
Figure 6: UML diagram for CPP Wrapper .....	14

## LIST OF TABLES

Table 1: Properties and Methods of CPPInput Class.....	15
Table 2: Properties and Methods of Vertex Class that are not part of the GeoAPI.Coordinate class .....	15
Table 3: Properties and Methods of Edge Class.....	15

## LIST OF ABBREVIATIONS

<b>API</b>	Application Programming Interface
<b>DLL</b>	Dynamic Link Library
<b>CPP</b>	Chinese Postman Problem
<b>RPP</b>	Rural Postman Problem
<b>TSP</b>	Traveling Salesman Problem

## REVISION CHART AND HISTORY LOG

REV	DATE	REASON
0.1	01/07/2020	Initial
0.2	15/07/2020	Draft
0.3	31/07/2020	Final

## Executive Summary

This deliverable describes part of the results of MOBIL0's project WP6. In this deliverable the algorithm that fully covers a road network graph with a single continuous traversal that has the minimum cost is presented. This problem can be formulated by the "Chinese Postman Problem" (CPP), that is to find a shortest closed path or circuit that visits every edge of a (connected) undirected graph. In our case, the graph is rather directed, since in real – world road networks there are several one-way streets that should be modeled as that. Moreover, we invest on a variation of the CPP, i.e., the Rural Postman Problem which implies that several graph's edges need not to be obligatory visited, which is often the case in real world where several network edges have been already visited (or need not to be). We employ existing implementations that solve this kind of problems and we extensively refactor them in order to be used in our infrastructure. We finally provide a .NET wrapper that consumes the C++ dynamic link library and exposes its results in a NET framework interface, allowing thus to be used by modern programming languages as C# and VB.

## 1 Introduction

The current deliverable aims to provide an algorithm that fully covers a road network graph (Figure 1) with a single continuous traversal that has the minimum (or near minimum in the case of a heuristic) cost. Our problem can be formulated by the “Chinese Postman Problem” (CPP), that is to find a shortest closed path or circuit that visits every edge of an (connected) undirected graph. In our case, the graph is rather directed, since in real – world road networks there are several one-way streets that should be modeled as that. Moreover, often there are cases where several graph’s edges need not to be mandatory visited, which is often the case in real world where several network edges have been already visited (or need not to be). As such we are interested in the Rural variation of CPP that also treats such directed graphs.



*Figure 1: An example road network graph*

In the following we present the basic notions for examining the CPP algorithm that is employed in our context. In Section 2 we provide the basic theoretic background for understanding CPP algorithm. Section 3 presents the variations of the algorithm regarding real-world needs. Section 4 provides details about our implementation, while Section 5 concludes the deliverable summarizing its results.

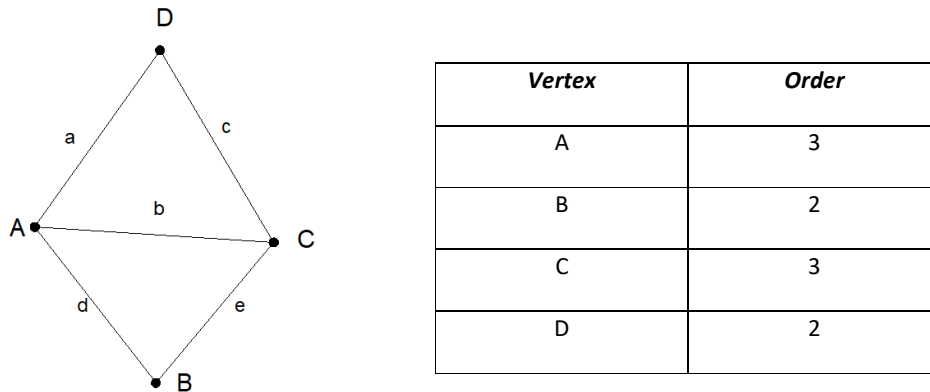
## 2 Chinese Postman Algorithm

In graph theory [1], the *Chinese postman*, or *route inspection*, problem is to *find a shortest closed path that visits every edge of a (connected) undirected graph*. When the graph has a Eulerian trail (a closed walk that covers every edge once), that path is an optimal solution. Otherwise, the optimization problem is to find the smallest number of graph edges to duplicate (or the subset of edges with the minimum possible total weight) so that the resulting multigraph does have a Eulerian circuit. The problem can be solved in polynomial time and was originally addressed in [2]; due to the country of origin of the initial paper the



problem named "Chinese postman problem". The problem has a solution if and only if the graph is strongly connected (each node can be reached from every other node) and contains no negative cycles (closed paths with negative total edge weight).

In order to address the algorithmic solution of the Chinese Postman Problem we have to examine some features of graph theory. To start with, we first define the *order* of a vertex in the graph, to be the number of edges that pass through this vertex. For example, in Figure 2, vertices A and C have order of 3 and vertices B and D have order of 2.

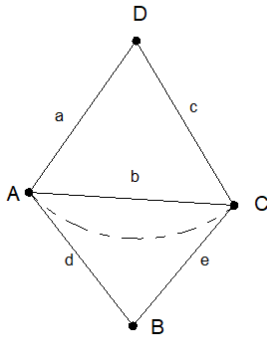


*Figure 2: Vertex and graph Order*

Then, the concept of *traversable graph* is introduced, which is any graph that can be drawn without taking a pen from the paper and without retracing the same edge; then the graph it is said to have a *Eulerian trail* or *Eulerian path*. When this path starts and ends at the same vertex it is said to form a *Eulerian Circuit* or *Eulerian Circle*. It can be proven that for a graph to be traversable, all vertices must be of even order. Moreover, if the graph has two odd vertices, it is said to be semi-Eulerian: the graph can be drawn without taking the pen from the paper and without retracing the same edge, by starting at the one odd vertex and end at the other.

Now, the problem of finding the optimal Chinese Postman Route can be reduced to finding an Euler Tour in the graph. This is due to the fact that *if an Euler Tour exists, it must be the optimal tour, since each edge is traversed exactly once*. Otherwise, one or more edges will have to be visited multiple times.

In order to make a non-traversable graph, traversable, we have to add one or more artificial edges to the graph, connecting vertices with odd order between them. For example, vertices A and C in Figure 2, can be connected by an artificial edge as illustrated in Figure 3, thus making the order of all vertices even.



*Figure 3: Artificial graph edge addition to make the graph Eulerian*

This situation can be modeled by adding additional Edges to the graph that represent edges used multiple times (and have the same weight as the original edge). Then the round trip using the edge multiple times will have the same cost as the corresponding Euler Tour in the new graph.

Therefore, the evolving problem now is to provide an efficient way of pairing the odd vertices between them. It can be proved that the number  $m$  of possible pairings of  $n$  odd vertices is given by the following equation:

$$m = (n - 1) \times (n - 3) \times (n - 5) \times \dots \times 1$$

Having these facts in mind, to find a minimum Chinese postman route, we must traverse each edge at least once and in addition we must walk along the least pairings of odd vertices on one extra occasion. To sum-up, an algorithm for finding an optimal Chinese postman route length (sum of edge weights) is:

- **Step 1:** Determine all odd vertices in graph.
- **Step 2:** Resolve all possible pairings of odd vertices between them.
- **Step 3:** For each possible vertex pairing calculate the route with the minimum weight between vertices.
- **Step 4:** Find the pairings such that the sum of the weights is minimized.
- **Step 5:** Add the artificial edges that have been found in Step 4 on the original graph.
- **Step 6:** The length of an optimal Chinese postman route is the sum of all the edges added to the total found in Step 4.

While the algorithm for determining the length of the optimal Chinese postman route is rather simple, the determination of the actual order of edge traversal corresponding to this route can be quite complicated, especially in the case of complex graphs. For this reason, one should previously determine the number of times each vertex will appear in a Chinese postman route. The respective algorithm is as follows:

- **Step 1:** Make the graph Eulerian by adding the extra edges to the initial graph.

- **Step 2:** Determine each vertex's order. Given step 1, all vertices should have even order.
- **Step 3:** The number of times each edge will appear in a Chinese postman route will be half the order of its vertex, with the exception being vertex A (the start/finish vertex), as this will be found in the solution one extra time.

Now a route corresponding to the minimum weight, can be easily found by simply traversing the graph choosing at every next step a vertex that has been visited less than the times calculated in step 3 above.

### 3 Variations

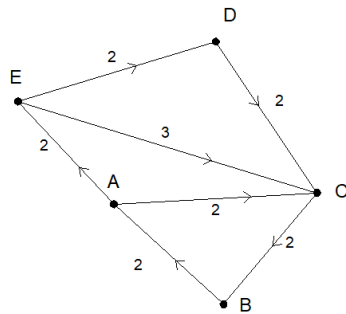
There are many variations of the original Chinese Postman Problem regarding the context on which the problem and its algorithms apply. For example, the starting and ending vertices do not always have to be identical since the vehicle may start and end its route on different points. However, the most practical variations of the problem and the respective algorithm are the ones that can be applied on a directed graph, and the one where some of the graph edges are not compulsory. We will deal with these problems in the next sections.

#### 3.1 Different starting and ending vertices

Sporadically, problems may be set where the start and finish vertices do not have to be the same. As stated earlier, any graph with only two odd vertices is semi-Eulerian. In this type of graph, the length of the Chinese postman route is the sum of all the edges of the network. In a network with four vertices, the graph is semi-Eulerian plus two odd edges. In addition to the start and finish vertices there are two other odd vertices. The shortest Chinese postman route is the sum of all the edges plus the shortest distance connecting the two remaining odd vertices.

#### 3.2 Chinese Postman on directed graphs

On a directed graph, the problem can be also reduced to finding an Euler Tour in the graph. In this case, it is employed the notion of the number of incoming edges at a vertex (the *In-Degree* of that vertex), and the number of outgoing ones (the *Out-Degree*). Then, the *Degree*  $\delta_v$  of vertex  $v$ , is given as the difference between In and Out degrees (Figure 4). It can be proven [1] that a graph has an Euler Tour if and only if the In- and Out-degrees are the same for all vertices in the graph, i.e., the Degree of each vertex is Zero,  $\delta_v = 0 \forall v$ . Therefore, it is sufficient to add additional paths to the graph, such that In- and Out-Degrees match for all nodes after inserting the new paths. In this case, the sum of the edge weights of these paths should be minimal.



Vertex	Degree
A	1
B	0
C	-2
D	0
E	1

Figure 4: Vertex Degree in a directed graph

Specifically, we need to insert paths from those nodes with too few outgoing edges to those nodes with too few incoming edges. Since our goal is to minimize the costs, all these artificial edges should be shortest paths. One efficient way to find those shortest paths is to employ the Floyd-Warshall algorithm [3], instead of Dijkstra [4], which is especially useful here, as it simultaneously finds the shortest paths between all pairs of vertices. The costs of the shortest paths will be used in the Matching Phase so as to determine the optimal set of shortest paths to use.

By inserting additional paths, we aim to balance In- and Out-degrees for all vertices of the graph, as only then a Euler Tour will exist. In order to determine which paths to add, we need a mapping from nodes with negative to those with positive degree. As such,  $\delta_v$  also specifies the number of new paths that need to start (if positive) or end (if negative) in  $v$ . In our example in Figure 4 Vertex C has degree of -2 while vertices A and E have degree 1, therefore two artificial edges from C to A and from C to E need to be calculated as shortest paths: in Figure 5 we demonstrate that this edges from C to A will be C-B-A (cost 4) and from C to E will be C-B-A-E (cost 6).

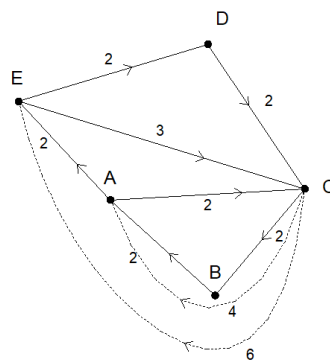


Figure 5: Adding artificial edges in in a directed graph

The total weight of this additional – artificial - paths to balance in and out degrees must be minimized. To do so, we create a new bipartite graph of nodes with degree  $\neq 0$  adding on it as many copies of each imbalanced vertex as its (absolute value of) degree. The weight of an

edge in the matching graph represents the length of the shortest path from its origin node (negative degree) to its destination node (positive degree). Now, an algorithm for finding optimal weighted matchings can be used (such as the Hungarian Method [5]) yielding the optimal matching so we can proceed to add the additional paths. *The edges of these paths then represent the edges which will appear multiple times in the optimal Round Trip.*

After making sure that the graph contains an Euler Tour, we still have to find it. To do so, we may use any appropriate algorithm, such as the Hierholzer method [1] for computing Eulerian Circuits. The cost of the optimal round trip then is exactly the sum of all edge weights in the Euler Tour.

### 3.3 Rural postman problem

Rural Postman Problem (RPP) is a practical extension of the well-known Chinese postman problem (CPP), where a subset of the underlying graph's edges is not required to be traversed at all. The Rural Postman Problem is NP-complete if this subset does not form a weakly connected graph. This is actually a very practical extension of the problem given real-world applications where some road network links have already been visited, while they can be used to access other parts of the non-visited network road segments.

## 4 Implementation

Towards the employment of the already existing algorithms in our context we used existing work by [6]. Specifically, [6] provides a solver that can handle all the possible combinations for a graph type (directed – undirected), problem's variants and output path requirements. As such, graph's edges can be either all directed, all undirected or a mixture of the two. While the standard CPP problem require all the edges to be visited, the rural variation it is possible to specify a subset of edges which is not compulsory to be visited. The output can be either a circuit, i.e., a closed path where the start and end nodes are coincident, or a generic open path. Regarding different types of solutions, there are different papers that support the implementation [7], [8]

The original repository provides C++ code for addressing the CPP problem. The code has been developed and tested on Ubuntu 16.04 having the following requirements:

- Eigen [9]
- OpenCV (only for visualization) [10]
- ROS kinetic (only for the ROS plugin)
- Nav2d (only for the ROS plugin)

The original objective of CPP implementation in [6] was to use it in a robotic environment. The library is used to create a ROS (Robot Operating System) exploration plugin for the Nav2d ROS package. ROS provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source,

BSD license. The plugin of [6] allows the user to draw a graph and autonomously explore it while building a map of the environment using the Nav2d SLAM algorithms.

On the other hand, our focus is on employing the algorithms on real-world road network data, using the software on windows machines. Following the rest of our infrastructure all software components should provide functionality on Windows Machines and be written preferably in Visual C++ and C# programming languages. As such we proceeded to a complete refactor of the code provided in [6] excluding parts that are not useful in our context. Specifically, from the above requirements, only Eigen was kept and all other requirements and supporting code were excluded from our implementation. The final output was of the project was provided in the form of a dynamic link library (dll). Moreover, a C# wrapper that handles communication between .NET managed objects and the C++ dynamic link library was developed.

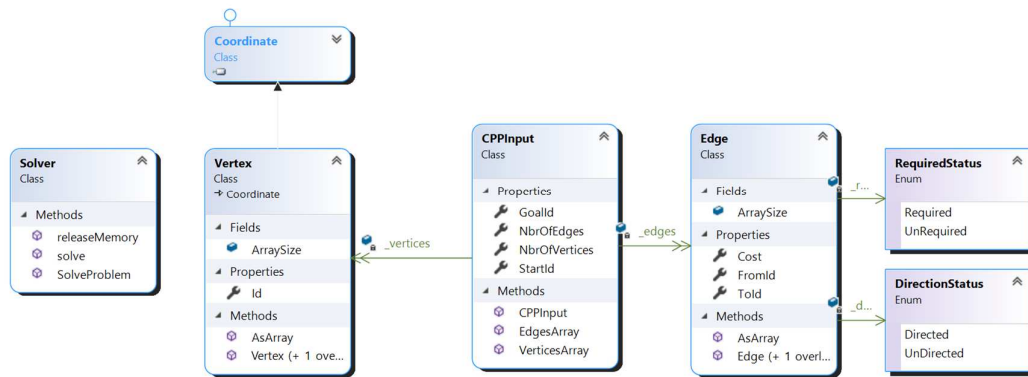



Figure 6: UML diagram for CPP Wrapper

The wrapper provides a static Class *Solver* with a single public method *SolveProblem*, given a *CPPInput* object and returns the order of vertices to be returned by the algorithm. Several other classes are implemented in the wrapper:




- **Class CPPInput** : Helper class that provides to the solver all necessary information to be passed to the C++ dll (graph and other parameters)

CppInput	⊕	Constructor method that creates a new object by a list of Edges, a list of Vertices and a starting and an ending vertex
GoalId	🔑	The Id of the goal (last vertex) of the algorithm
StartId	🔑	The Id of the start (first vertex) of the algorithm
NbrOfEdges	🔑	Number of edges in the graph
NbrOfVertices	🔑	Number of vertices in the graph
EdgesArray	⊕	Gets the graph Edges in the form of a 2-d double array to be passed in C++ dll

VerticesArray		Gets the graph Vertices in the form of a 2-d double array to be passed in C++ dll
---------------	---	---







**Table 1:** Properties and Methods of CPPInput Class

- **Class Vertex:** Class that represents a vertex in the graph by its Id and coordinates. Class inherits from GeoAPI.Coordinate object to provide simple functionality for regarding a point in the space (e.g., distance between points). Properties and methods other than the inherited ones are provided in the following table.

Vertex		Constructor method that creates a new vertex by its id and X and Y coordinates
Id		The vertex id
AsArray		Gets the vertex information in the form of a double array to be passed in C++ dll

**Table 2:** Properties and Methods of Vertex Class that are not part of the GeoAPI.Coordinate class

- **Class Edge:** Class that represents an edge of the graph containing its start and end vertices, as well as its cost. Properties and methods are provided in the following table.

Edge		Constructor method that creates a new edge by its starting and ending vertex id and the respective cost
FromId		The vertex id of the edge start
ToId		The vertex id of the edge end
Directed		The direction status of the vertex (directed / undirected)
Required		The required status of the vertex (required / unrequired)
AsArray		Gets the edge information in the form of a double array to be passed in C++ dll

**Table 3:** Properties and Methods of Edge Class

Before each call to the solver, a class *CPPInput* object is constructed by a list of Edges, a list of Vertices and a starting and an ending vertex and provides to the solver all necessary information to be passed to the C++ dll. The *Edge class* provides information whether the edge is directed / undirected and Required / Unrequired. In the latter case, the CPP problem transforms to a Rural Postman Problem where some edges are not required to be visited. Results of *Solver* are also transformed to a path in the graph by the wrapper.

## 5 Conclusion

In this deliverable we present the algorithm that fully covers a road network graph with a single continuous traversal that has the minimum cost. Our problem can be formulated by the “Chinese Postman Problem” (CPP), that is to find a shortest closed path or circuit that visits every edge of an (connected) undirected graph. In our case, the graph is rather

directed, since in real – world road networks there are several one-way streets that should be modeled as that; moreover, the Rural Postman Problem implies that several graph's edges need not to be obligatory visited, which is often the case in real world where several network edges have been already visited (or need not to be). As such we are interested in the Rural variation of CPP that treats directed graphs. We utilize existing implementation of this problem extensively refactoring them to be used in our infrastructure that is developed mainly with Visual Studio and .NET managed assemblies. We finally provide a .NET wrapper that consumes the C++ dynamic link library and exposes its results in a NET framework, allowing thus to be used by C# and VB.



## 6 References

- [1] Y. Manolopoulos, Lessons of Graph Theory (in greek), Athens: New technology publications, 1996.
- [2] M.-k. Kwan, "Graphic programming using odd or even points," *Acta Mathematica Sinica (in Chinese)*, vol. 10, pp. 273-277, 1962.
- [3] E. W. Weisstein, "Floyd-Warshall Algorithm," [Online]. Available: <https://mathworld.wolfram.com/Floyd-WarshallAlgorithm.html>.
- [4] E. W. Weisstein, "Dijkstra's Algorithm," [Online]. Available: <https://mathworld.wolfram.com/DijkstrasAlgorithm.html>.
- [5] "Hungarian algorithm," [Online]. Available: [https://en.wikipedia.org/wiki/Hungarian\\_algorithm#cite\\_note-munkres-3](https://en.wikipedia.org/wiki/Hungarian_algorithm#cite_note-munkres-3).
- [6] A. Soragna, "GitHub - alsora /chinese-postman-problem," [Online]. Available: <https://github.com/alsora/chinese-postman-problem>.
- [7] L. Xu, "Graph Planning for Environmental Coverage," Pittsburgh, Pennsylvania, 2011.
- [8] L. Xu and A. Stentz, "A Fast Traversal Heuristic and Optimal Algorithm for Effective Environmental Coverage," in *Proceedings of Robotics: Science and Systems (RSS '10)*, 2010, 2010.
- [9] "Eigen," [Online]. Available: <https://eigen.tuxfamily.org/>.
- [10] "OpenCV software library," [Online]. Available: [1] <https://opencv.org/>. [Accessed 31 3 2019].