TerraNetwork: a Urban Street Network Analysis System

Geraldo Ribeiro Filho, Reinaldo Arakaki, Marcio P. de Aquino, Luiz A. N. Lorena

Instituto Nacional de Pesquisas Espaciais – INPE

LAC - Cx. Postal 515 – 12227-010 – São José dos Campos – SP – Brazil

{geraldo, reinaldo.arakaki, marcio.paim, lorena}@lac.inpe.br

Abstract. *This paper presents the current state of the TerraNetwork project, its structure, characteristics, functionalities and the implemented combinatorial optimization algorithms, used to solve urban street networks related to location and transportation problems. The street clusters functionality and the Chinese Postman are particularly detailed. Ongoing development and some future work possibilities and objectives are also described.*

1. Introduction

There are many practical decision making problems involving service facilities installation points such as hospitals, factories, product distribution centers, warehouses, mobile telephony antennas etc. These location problems are linked to transportation problems with the objective of optimize the use of such facilities. This kind of problems can be very difficult to solve, and therefore attract special interest when occur in cities with large street networks, demanding solution techniques and algorithms found in the combinatorial optimization field.

These problems have some common characteristics: typically occur within a context with a finite number of possible locations and routes, but the number of options can be very large, making the problems hard to solve; the data are spatially distributed and fit software systems aimed to process spatial entities; the spatial data can be modeled as networks, structures formed by nodes and arcs that naturally describe streets, avenues, intersections and addresses.

Geographic Information Systems (GIS) are modern software tools used to process spatially distributed data. GIS integrate graphical user interfaces for data visualization with spatial databases. These systems allow the association of network elements to attributes like street names, lengths and traffic directions, among others, commonly used as input for location and transportation problems.

TerraNetwork is an ongoing two year project which objective is to create software to apply developed optimization techniques and algorithms to location and transportation problems in a GIS environment. A C++ class library has been created based on the TerraLib, a free spatial database model and access classes developed by the National Institute of Space Research (INPE). TerraNetwork classes also use the Boost Graph Library (BGL), a free library that provides fast implementations of useful graph processing algorithms [http://www.lac.inpe.br/~terranetwork].

The following sections present support software libraries, the TerraNetwok classes' structure and the implemented functionalities, and particularly, the street clusters algorithm based on seed points and the use of the Chinese Postman algorithm to find paths that optimally cover an entire street cluster. Finally, current development and perspective for future work is presented.

2. Support Software

The TerraNetwork classes use two support software libraries: the TerraLib and the Boost Graph Library, briefly described bellow.

2.1 TerraLib

Currently there are two possibilities for GIS store spatial data: the use of one or more isolated files with some system specific format, or the use of databases with some spatial data appropriated schema. The market domain of some GIS can turn its isolated files format into "de facto" standards for spatial data storage. This is the case of the "shape files" format developed by a company named ESRI (http://www.esri.com) for its GIS products. Although, research efforts have being made to propose database schemata specifically designed to store spatial data, taking advantage of the facilities and quality and security guaranties offered by many Database Management Systems (DBMS).

TerraLib (http://www.terralib.org) is a project developed within the National Institute of Space Research (INPE) and proposes a relational database schema to store spatial data and its associated attributes. The project also makes available a free C++ class library to access and maintain stored data. Beside general use tables, the database schema has two kinds of tables to store spatial data: the geometric data tables used to store geometric objects like points, polygons and lines that represent spatial entities, and the attribute tables used to store information associated to those geometric objects.

As in many GIS, the spatial data is organized in layers, and in the TerraLib model there are views and themes associated to layers, used to display information in user interfaces. There is also available a free graphic user interface called TerraView that can be used to data visualization. TerraLib routines can connect to several DBMS to create and manage the database, and there are also routines to import ESRI shape files.

The TerraNetwork classes provide functionalities to connect to TerraLib databases where there must be "a priori" at least one street network layer and eventually one or more layers with interest points. All data is geographically positioned according to some coordinate system, normally latitude and longitude, and the interest points lies over streets. Algorithms solutions are inserted into the database in new added layers.

2.2 Street Graphs and the Boost Graph Library

The combinatorial optimization algorithms used to solve network problems act over graph data structures. TerraNetwork classes read the street layer from the TerraLib database and build a directed graph $G=\{V, E\}$, where each vertex $v_i \in V$ corresponds to a street intersection, and each edge $e_i \in E$ corresponds to a street segment between two intersections. Edge directions are taken from a special field in the street attribute table called "oneway" that holds a code indicating traffic direction for the street. The classes have the ability to merge interest points into the graph as new vertices, proportionally splitting edges and numeric attributes. The graph structure also can hold vertex and edge attributes which names are given in a string array parameter to the graph creation method. The network graph can be saved into the TerraLib database general use tables.

There is some useful well known graph processing algorithms with good performance implemented in a free software library called Boost Graph Library (BGL) (http://www.boost.org). The TerraNetwork classes call BGL routines to implement some of its methods. Currently, the Dijkstra and the Floyd_Warshall (Cormem et al. 2001) algorithms are used to find the shortest path between two vertices, and a third algorithm is used to find the number of connected components of the graph.

3. TerraNetwork Classes and Current Functionalities

The TerraNetwork simple class model has eight related to each as shown by Figure 1. Model main class is called *Net*, and has several methods to implement the algorithms used to solve network problems, as described in the next subsections.

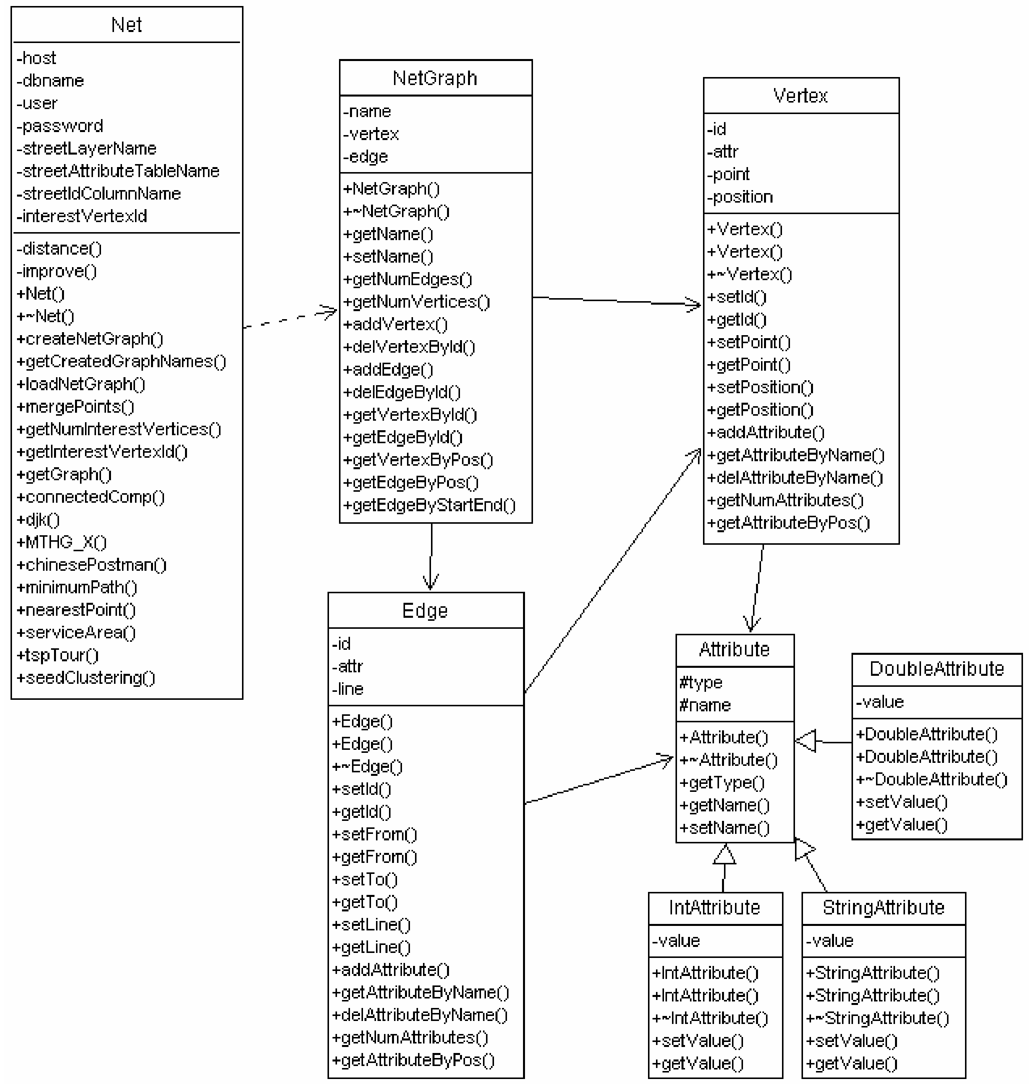

Figure 1. TerraNetwork Simple Class Model

The class *NetGraph* models a street graph identified by a name and having collections of *Edge* class and *Vertex* class objects as attributes. Both *Edge* and *Vertex* objects have collections of *Attribute* class objects. The *Attribute* class has three specialized subclasses: *IntAttribute*, *DoubleAttribute* and *StringAttribute* to store integer, double precision and string data, respectively. While the model classes have private attributes, the constructor, destructor, attribute access, and most of the service methods are public.

A typical application initially creates a *Net* class object, passing the necessary information to connect to a DBMS as constructor parameters. Next, the application chooses one of the already created street graphs and loads it using the *loadNetGraph* method, or creates a new street graph using the *createNetGraph* method. Graph creation demands several information regarding spatial data layers and attributes, passed as parameters to the creation method. Optionally, the application can then insert interest points as new vertices merged to the loaded or created graph. Also, spatial data and attribute information are required. Finally, the application can invoke some *Net* class service method to solve a problem regarding the interest points and streets modeled in the graph structure. Solutions spatial data and attributes are inserted as new information layers in the database.

3.1 Shortest Paths

One of the most common tasks performed with a GIS is the determination of the shortest path between two vertices of a network. The TerraNetwork *Net* class has a method called *minimumPath* that performs this task. This method requires several parameters: the path start vertex identification, the path end vertex identification, the name of the edges cost attribute, and information used to store the solution data into the database, the path layer, view and theme names, a string array with the path attribute names, and the table name in which these attributes will be stored. Attributes must be selected from the vertices and edges attributes.

The minimumPath method calls a BGL routine that implements the Dijkstra shortest path algorithm [Dijkstra 1959]. Before use any BGL graph routines, the TerraNetwork class methods must build a BGL specific graph representation based on adjacency lists. The BGL routine output is then used to create a new information layer in the database with the street segments that compose the path. Therefore, the solution can be shown in any TerraLib viewer. The TerraView (http://www.dpi.inpe.br/terraview) partial screen shot in Figure 2 shows the shortest path between two extreme points of the São José dos Campos city.

Another GIS application may require the determination of the shortest path among several interest points in the network. This problem is equivalent to the classic Travelling Salesman Problem (TSP) [Lawler at al. 1986] which objective is to find the minimum cost path taken by a salesman to, starting from an origin point, visit exactly one time every other interest point, and return to the origin. The TSP naturally arises as an abstraction of several transportation and logistics applications, for example the problem of arranging school bus routes to pick up the students in a school district, the scheduling of service calls at cable firms, the delivery of meals to homebound persons, the routing of trucks for parcel post pickup, and many others.

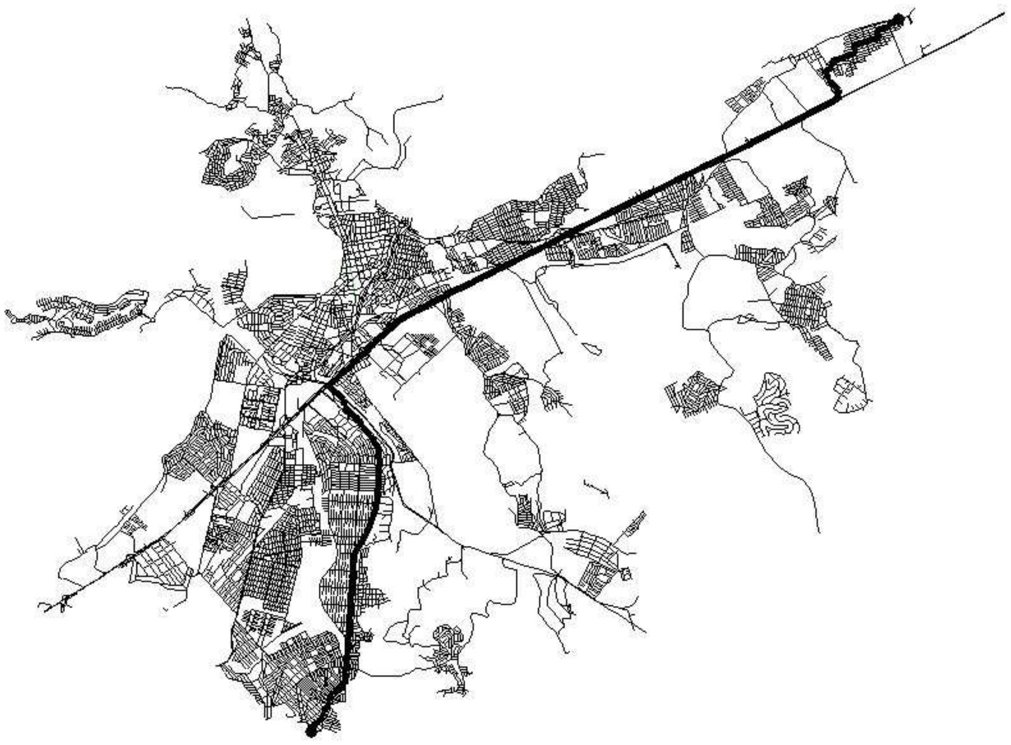

Figure 2. Shortest Path Between two Interest Points

The *Net* class has a method called *tspTour* that uses a heuristic to compute a tour that starts and finishes at an interest point, which identification is passed as a parameter to the routine, and visit exactly once each other point from a given layer. These points are previously merged into a street graph. Solution tour is saved as a new layer in the database and the routine also receives as parameters the name of the edges cost attribute, the tour layer, view and theme names, a string array with the tour attribute names, and the table name to store these attributes.

The TSP is NP-Complete (Fortnow and Homer 2002) and instances with many nodes requires heuristic approaches. There are basically two algorithm classes to solve the TSP: constructive algorithms that build a solution, and improvement algorithms that try to make better solutions from a previous one.

The *tspTour* method uses a greedy heuristic to compute an initial tour. It starts from the origin vertex and repeatedly inserts the nearest vertex to the last one inserted. Distances were previously computed using the Dijkstra algorithm from BGL.

After the initial tour has been computed, *tspTour* starts an improvement phase based on a tour transformation known as 2-optimal (Croes, 1958). It consists in remove two chosen edges from the tour and reconnects the opposite vertices. The *stpTour* tests all edge pairs and remove the pair that will mostly reduce the tour total cost. This process is repeated until no improvement is possible.

3.2 Closest Facility and Service Area

Two immediate applications of shortest paths are also implemented as TerraNetwork *Net* class methods: find the closest facility to a location on a network, and determine a service area around a location.

The term "closest facility" refers to any provider of a certain kind of service that is closest to a given location. For example, it could be the closest tow truck to a stranded car, the closest fire hydrant to a fire, or the closest drug store to a house.

TerraNetwork *Net* class provides a method called *nearestPoint* that receives input parameters to identify the destination vertex, the edges cost attribute name, and information used to store the solution data into the database, the path layer, view and theme names, and a string array with the path attribute names that will be stored.

The *nearestPoiny* routine simply runs the BGL Dijkstra implementation, select the nearest facility to the destination point, and store the solution data into a new layer. The facilities and the destination points are represented in an interest point layer and must be previously merged into a created or loaded street graph.

The word "accessibility" refers to how easy it is to reach a location and can be measured in travel time or distance. Examining accessibility can help, for example, a decision maker find how suitable a site is for a new business.

The TerraNetwork *Net* class has a method called *serviceArea* that can be used to find which interest points from a set are within a given travel distance or time from a given origin point. The origin point and the other interest points must be on a database layer that is merged into a street graph as vertices, and the method receives as parameters the identification of the origin vertex, the edges cost attribute name and the distance or time limit (or radius) to define the service area. Actually, the area can be defined by a limit value in the same measurement unit of the edges cost attribute, not necessarily it must be distance or time units.

The method also uses the BGL Dijkstra implementation to determine minimum network distances from the origin to every other interest point, select the points that are within the given limit and builds a geometric representation.

As the *serviceArea* method represents its solution as a convex hull polygon in the database, the method also must receive as parameters the layer, view and theme names for the polygon, a list of attribute names and table name for the interest points eventually positioned within the service area.

The polygon is built to include all graph vertices which distance from the origin is less or equal to the given limit. Therefore it includes both interest points and street intersections. As the routine uses the TerraLib facilities to create a polygon filled with a solid transparent color, visually, the polygon may be shown over the street network representation and the all street segments with both extremes reachable within limit may be seen inside the polygon.

Figure 3 shows a TerraView partial screen shot with a test instance solution with two interest points within the service area. The most central is the origin and the other is the only remained point lying inside the area.

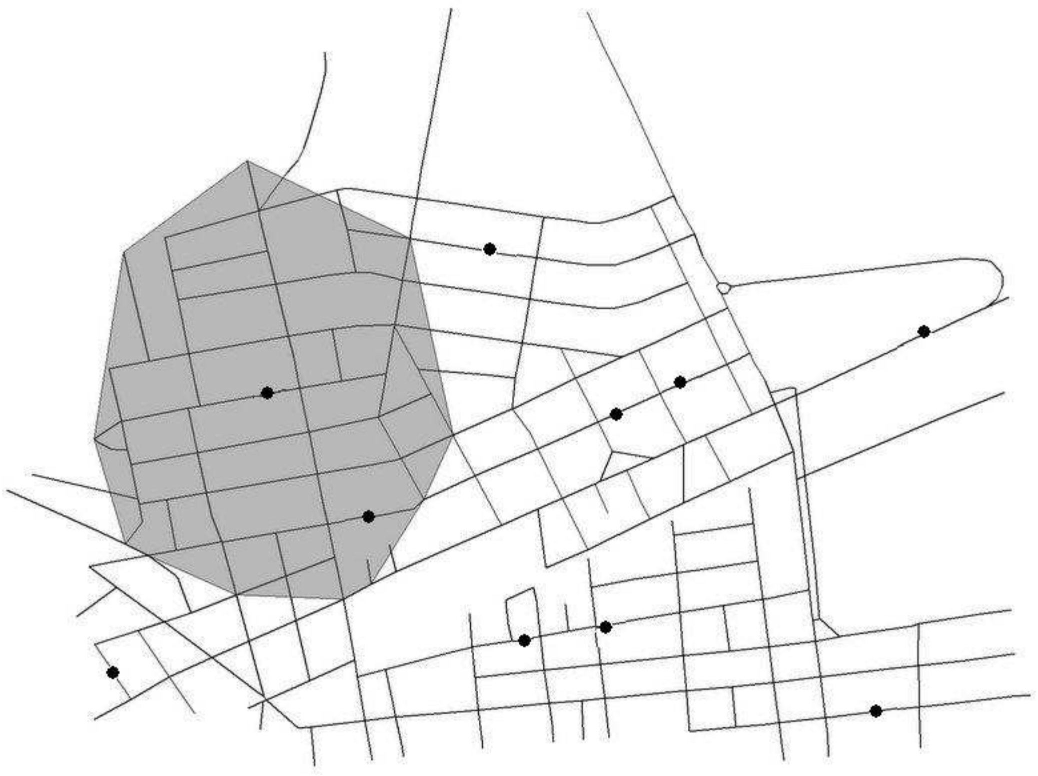

Figure 3. Service Area Polygon

3.3 Street Clustering

Another functionally provided by TerraNetwork is the ability to divide the network in regions according to capacity constraints. This procedure may be useful in practical applications like items collection by limited capacity vehicles or data acquisition by public service employees with restrictions regarding total distance, time or amount of data.

Osman and Christofides (1994) describe the Capacitated Clustering Problem (CCP) as the problem in which a given set of elements is to be partitioned into a set of clusters. Each element has an associated weight (or demand) and must be associated to exactly one cluster. Each cluster has a given capacity which must not be exceeded by the total weight of elements in the cluster. A dissimilarity measure is a cost (or distance) between any two elements. For a given cluster, a *centre* is an element of the cluster from which the sum of the dissimilarities to all other elements in the cluster is minimized. The objective is to find the *centres*, and therefore the clusters, which minimize the sum of dissimilarities (or distances) for all clusters. As TSP, CCP is NP-Complete and heuristic methods are used to solve large instances.

The TerraNetwork *Net* class has a method called *seedClustering* that uses a local search heuristic to find near optimal location for *seed* points acting as street clusters *centres*.

The number of desired street clusters must be known *a priori* and the initial seed locations are given by interest points on a layer that must be merged as vertices into a created or loaded street graph. There must be one seed for each cluster to be formed.

Assignment of a street segment to a seed is based on the distance from the seed to the segment. The *seedClustering* consider this distance as the network distance from the seed to its nearest segment extremity plus the segment length. Street demand is given by one of its attributes and each seed has a capacity also given by one of its attributes.

The *seedClustering* local search heuristic tries to move each seed to it neighbor vertex that gives the greatest total cost reduction. The neighborhood is delimited by a network distance radius passed as a parameter to the routine. This process is repeated until no improvement is obtained and the seeds remain stopped. Figure 4 shows a pseudo code representation of this algorithm. Similar techniques have been used in other works related to location problems (Arakaki 2002, Lorena and Pereira 2002).

```
While (improvement)
  For each moving seed
    Calculate distance from seeds to the streets
    Assign streets to seeds
    For each neighbor vertex
      Move seed to vertex
      Calculate distances from moved seed
      Assign streets to seeds
      If best move so far then save vertex
    End for
    Move seed to best neighbor vertex
  End For
End While
```

Figure 4. Street Clustering Algorithm

Network distances are calculated by Dijkstra BGL implementation, and the street to seed assignment is made by a heuristic that maintain capacity constraint. The heuristic was developed by Martello and Toth to solve the NP-Complete Generalized Assignment Problem (GAP). The GAP consists in assigning a set of tasks to a set of agents with minimum cost. Each agent has a limited amount of a single resource and each task must be assigned to one and only one agent, requiring a certain amount of the resource of the agent (Martello and Toth, 1990).

Several parameters must be passed to the *seedClustering* routine: the street length attribute name, the street demand attribute name, the seed point capacity attribute name, the neighborhood radius for local search. The routine creates a layer for each cluster with its streets and seed point. Required parameters for solution output are the clusters layers, views and themes name prefix, clusters attributes table name prefix and a list with desired attributes for clusters elements.

Figure 5 shows a TerraView partial screen shot with three street clusters and its respective centres final location for a test instance.

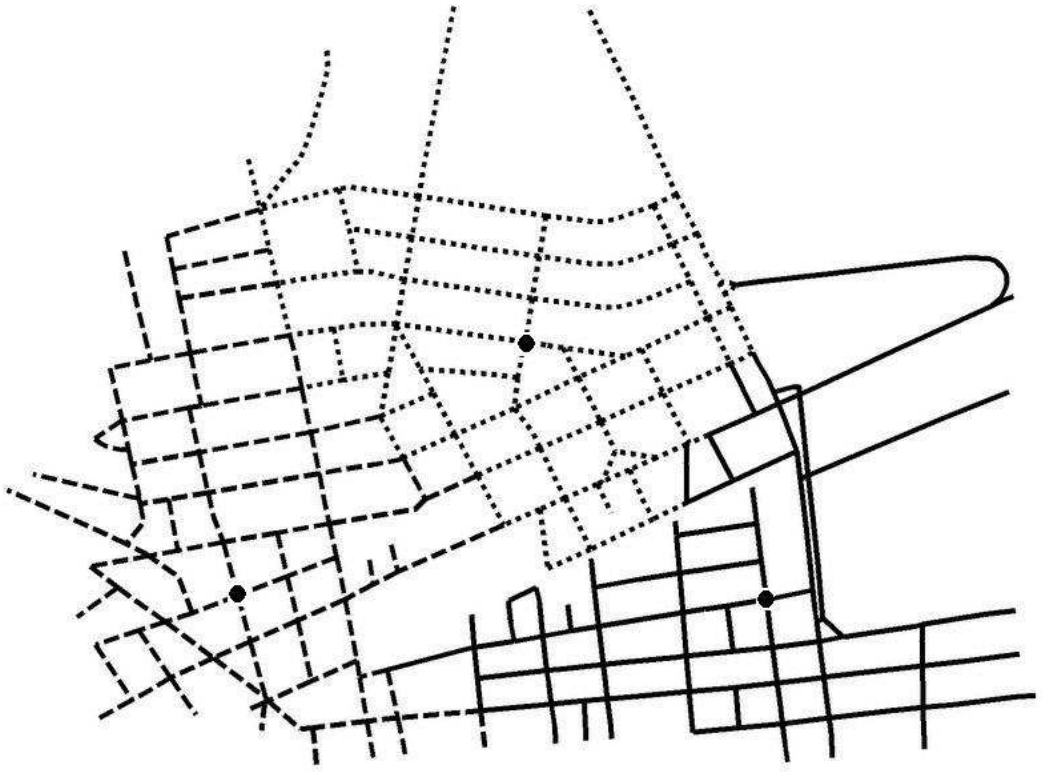

Figure 5. Three Street Clusters

Most street networks have street segments allowing traffic on both directions, and in these cases the direct street graph has two edges between vertices corresponding to segment extremities. At the end of the clustering process, there may be two directions streets at the cluster border apparently belonging to an adjacent cluster as well. This fact results from graph edges corresponding to a single street assigned to different seeds.

4. Chinese Postman Algorithm

Consider the case of a mailman who is responsible for the delivery of mail in a city area. The mailman must always begin his delivery route at the location where the post office is located; must traverse every single street in his area; and, eventually, must return to the origin. The natural problem that arises is to find the mailman's route that minimizes the total distance he walks, while at the same time traversing every street segment at least once. This type of edge-covering problem is known as the *Chinese postman's problem* (CPP), it can be formulated for undirected and directed graphs, and is NP-Complete for mixed graphs (Larson and Odoni, 1981).

The CPP was examined in detail by the great mathematician and physicist Leonhard Euler in the eighteenth century. Euler tried to find a way in which a parade procession could cross all seven bridges at the Russian city of Konigsberg (now Kaliningrad). Euler derived general results that provide motivation for some of the solution approaches to the CPP. The problem name derives from the fact that an early paper discussing this problem appeared in the journal Chinese Mathematics (Mei 1952).

TerraNetwork *Net* class has a method called *chinesePostman* that implements an algorithm for undirected graphs. This algorithm uses Graph Theory concepts related to Euler´s work.

An Euler tour is a circuit which traverses every edge on a graph exactly once, beginning and terminating at the same vertex. A connected graph G has an Euler tour if, and only if, G contains exactly zero vertices of odd degree. This fundamental result comes from the Euler´s Theorem.

Clearly, if a graph possesses an Euler tour, then this tour is the solution to the CPP. Euler´s theorem provides the guidelines for solving the CPP, even for graphs on which an Euler tour does not exist. The solution approach basically consists of augmenting these graphs in such a way that all odd-degree vertices are transformed to even-degree vertices, which in turn means that an Euler tour can be drawn on the augmented graph.

Let G = (V, E) be a connected graph with no vertices of odd degree. An Euler tour can be obtained beginning at some initial vertex $v_0 \in V$ and drawing a circuit C_0 through G, returning to v_0. If C_0 happens to be an Euler tour the process stop, otherwise we remove from G all edges used by circuit C_0. There must be some edges not traversed by C_0 and at least two of these edges must be incident on some vertex v_1 through which the circuit C_0 has passed. Thus, it is possible to draw another circuit C_1, originating and terminating at v_1, which uses only the edges left after we have removed the edges of C_0. If all left edges are used by C_1, then we are finished: the Euler tour is the circuit that results from using the initial portion of C_0 to go from v_0 to v_1, then C_1 to return back to v_1, and finally the second portion of C_0 to go from v_1 to v_0. If, however, after eliminating the edges on C_0 and C_1 from G, there are still some edges left uncovered, then at least two of these edges must be incident on a vertex v_2 which is either on C_0 or C_1. Then, it is possible to find a tour C_2, originating and terminating at v_2 and using only the edges left on G after removing the edges covered by C_0 and C_1. This procedure may now be continued until the *k*-th step, when there will be no edges left uncovered. At that time, an Euler tour is obtained as a combination of circuits C_0, C_1, C_2, . . .C_k, in the same manner explained for C_0 and C_1.

The CPP can be solved finding the minimum length, edge-covering tour of a graph G = (V, E) with no restrictions other than that it is connected and undirected.

The solution procedure consists of adding artificial edges, parallel to the existing ones, to the original graph G to obtain a new graph G1 = (V, E1) on which an Euler tour can be drawn. The addition of the artificial edges should turn all odd-degree nodes on G into even-degree nodes on G1, and should be accomplished by selecting the combination of artificial edges which results in the minimum length tour at the end.

The Chinese Postman Algorithm for undirected graphs can be described as follows:

1. Identify the number m (which is always even) of all vertices of odd degree in G = (V, E).
2. Find a minimum-length pairwise matching of the m odd-degree vertices and identify the m/2 shortest paths between the two nodes composing each of the m/2 pairs.
3. For each of the pairs of odd-degree vertices in the minimum-length pairwise matching found in previous step, add to the graph G the edges of the shortest path between the two nodes in the pair. The graph $G^1 = (V, E^1)$ thus obtained contains no vertices of odd degree.
4. Find an Euler tour on G^1. This Euler tour is an optimal solution to the CPP on the original graph G. The length of the optimal tour is equal to the total length of the edges in G, plus the total length of the edges in the minimum-length matching.

A pairwise matching of a subset $V' \subset V$ of vertices of a graph G is a pairing of all the vertices in V', assuming that the number of vertices in V' is even. A minimum-length pairwise matching of the vertices is then a matching such that the total length of the shortest paths between the paired nodes is minimum.

The number of possible pairwise matching combinations increases very quickly with the number of odd-degree nodes. TerraNetwork uses a freely available C code for finding a maximum weight matching in an undirected graph (Gabow, 1973). The code (http://elib.zib.de/pub/Packages/mathprog/matching/text.en.html) primarily finds the maximum matching instead of the minimum required by the CPP algorithm. However, the routine has parameters that can be easily set to make it find the minimum matching.

The Chinese Postman algorithm is applicable to situations that require an economical delivery/pickup route for things such as mail service or any regularly scheduled deliveries. The route can also be helpful to data collection services using handheld devices like water and electricity consume data, data collection for statistical study and information census.

5. Current Development and Future Work

Currently, algorithms developed on previous projects related to GIS are being adapted to work as new TerraNetwork functionalities. The algorithms are designed to work on problems belonging to a class named Location Problems.

Location problems try to find the best place to locate facilities, considering clients that must be attended and some criterion that must be optimized. The generic term “facility” may be understood as plants, schools, warehouses, antennas and others. Commonly the clients are assigned, or allocated, to the facilities, and therefore these problems are also known as location/allocation problems.

Applications can be found in public or private service sectors. Among examples in the public sector is the location of health care centers, schools, fireman stations, ambulances, police vehicles and bust stops. Private applications may involve production plants, shopping stores and radio antennas.

A combinatorial optimization problem that arises from many facility location problems is called Maximum Covering Problem. In this problem, a potential facility site covers a set of demand points. A nonnegative weight is associated to each demand point. The task is to select a subset from the set of potential facility sites, such that the sum of weights of the covered demand points is maximized. Mathematical formulation of some instances can be very hard to solve and algorithms based on lagrangean relaxation have been proposed. (Galvão, Espejo and Boffey 2000).

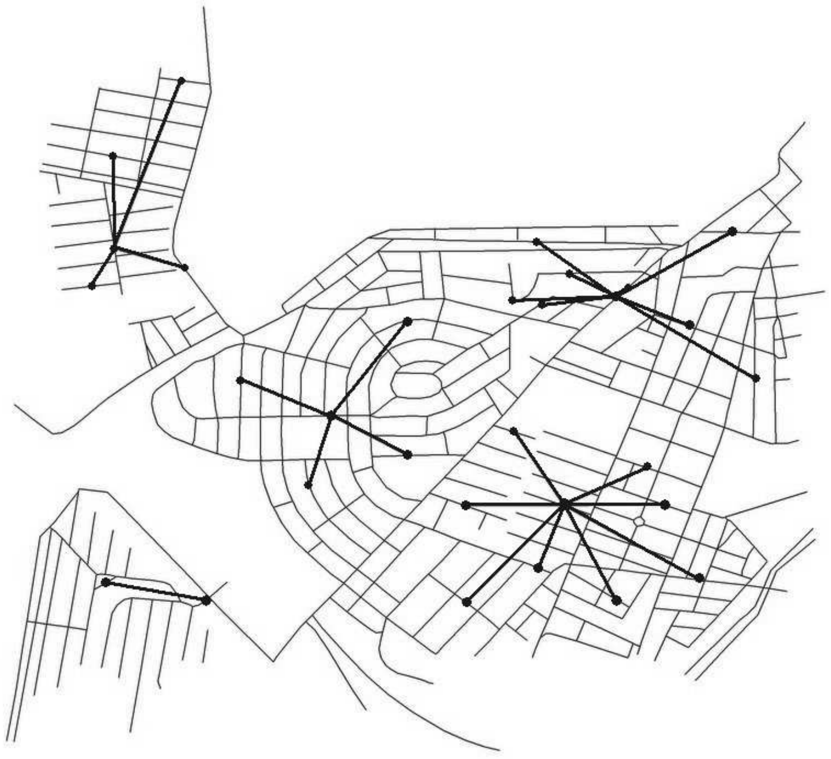

Figure 6. Maximum Covering Problem

Figure 6 shows a TerraView partial screen shot with a solution for a test instance with 31 potential sites for 5 facilities, using Euclidian distances. The heuristic algorithm implemented is based on lagrangean relaxations associated with local search (Lorena and Pereira 2002).

Logistics can be understood as the provision of products and services from supply points to demand points, and represents a fundamental field of application for GIS with functionalities related to urban networks. The main objective of GIS application is to improve transportation management to reduce costs and guarantee customers satisfaction.

One of the most important issues of transport management is effective vehicle routing. The optimization of vehicles routes given various constraints is generally known as the Vehicle Routing Problem (VRP). The variety of involved constraints leads to several forms in which the problem can be formulated. Vehicle capacity and time windows requirements are some of the origin of specific problem forms. Depending on instances the problem can be hard to solve and many approaches have been proposed. Beside mathematical formulations, exact and heuristic algorithms can be found on literature (Toth and Vigo 2001).

The next functionalities to be implemented on TerraNetwork classes will make the system suitable to deal with some forms of the VRP. Implementation of heuristic methods that produce results for real world instances in reasonable time is expected.

6. Conclusion

This paper presented the TerraNetwork project, a GIS prototype applicable to urban network analysis. The actual form of the class model was presented, along with its support software: the TerraLib library and database model, the TerraView user interface, and the Boost Graph Library. Functionalities already implemented and its related algorithms and combinatorial optimization concepts were described. The current effort on development and the perspective for new functionalities to be added to system were also presented.

The relevance of research effort on urban network analysis software tools for real world applications is unquestionable. The main purpose of this paper is to show the association of urban network GIS with classical and important concepts and techniques of combinatorial optimization.

Acknowledgements

The authors acknowledge the Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq (process 506840/2004-2) for partial financial support.

References

Arakaki, R. (2002), Heurística de Localização-Alocação para Problemas de Localizacao de Facilidades. Thesis (Applied Computing Doctor). National Institute of Space Research – INPE. Available from < http://www.lac.inpe.br/~lorena/pos-grad.html >.

Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. (2001), Introduction to Algorithms, MIT Press, second edition.

Croes, G. (1958), A method for solving traveling salesman problems. Operations Research, v. 6, p. 791–812.

Dijkstra, E. W. (1959), A Note on Two Problems with Connection with Graphs. Numerische Mathmatik, 1, 1959.

Fortnow L., Homer S. (2002) “A Short History of Computational Complexity”, In: The History of Mathematical Logic, Edited by D. van Dalen, J. Dawson, and A. Kanamori, North-Holland, Amsterdam.

Gabow H. (1973), Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs, Ph.D. thesis, Stanford University.

Galvão, R. D., Espejo, L. G. A. and Boffey, B. (2000), A Comparison of Lagrangean and Surrogate Relaxations for the Maximal Covering Location Problem. European Journal of Operational Research, 124, p. 377-389.

Larson R. C., Odoni A. R. (1981), Urban Operations Research, Prentice Hall, New Jersey.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. (1986) *The Travelling Salesman Problem*. John Wiley & Sons, England.

Lorena, L. A. N., Pereira M. A.(2002), A Lagrangean/surrogate heuristic for the maximal covering location problem using Hillsman's edition. International Journal of Industrial Engineering, 9(1), p. 57-67.

Martello S. and Toth P. (1990), Knapsack Problems: Algorithms and Computer Implementations, John Wiley and Sons.

Mei Ko, K. (1962), Graphic Programming Using Odd or Even Points, Chinese Mathematics, 1 (3), p. 237-277.

Osman, I. H. and Christofides, N. (1994) Capacitated Clustering Problems by Hybrid Simulated Annealing and Tabu Search, In: Transactions in Operational Research 1, p. 317-336.

Toth, P. and Vigo, D. (2001) The Vehicle Routing Problem, Society for Industrial and Applied Mathematics, Philadelphia, USA.