4th International Conference on Industrial Engineering and Industrial Management
XIV Congreso de Ingeniería de Organización
Donostia- San Sebastián , September 8th -10th 2010

Using real world distances in logistics management

Kostanca Katragjini[1], Rubén Ruiz[1], Alejandro Rodríguez[2]

[1] Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Edificio 7A, Camino de Vera S/N, 46021 Valencia, España.kostanca@iti.es, rruiz@eio.upv.es
2 Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Pza. Ferrándiz Carbonell, 2 03801 Alcoy, España. arodriguez@doe.upv.es

Abstract

Logistics deals with the planning and control of material flows and related information in organizations, both in the public and private sectors. Modeling and solving logistic problems often requires a considerable amount of data and the quality of the gathered data may influence algorithms' results. In this study we address the issue of gathering automatically, from geographic information systems (GIS), real world distances between nodes in a logistic network. We will show how the efficiency is of paramount importance when retrieving large amounts of logistic data and how our proposed architecture makes possible real data retrieving in reasonable times

Keywords: logistics, VRP, GIS, real travel distances

1. Logistics Management

A logistics system is made up of a set of nodes (facilities) connected by transportation services and by a transportation network. Nodes are locations where materials are processed, i.e., manufactured, stored, sold or consumed. Transportation services move materials between facilities using vehicles and equipment such as trucks, tractors, trailers, crews, pallets, containers, cars, trains, etc. The design of a logistic system typically deals with the best configuration, size and location of facilities. It also deals with transportation system design and optimization such as fixing the mode of transportation to use the best fleet size, schedule shipments, optimize vehicle routes, etc. (Ghiani, Laporte, Musmanno 2004). Modeling and solving logistic problems often requires a considerable amount of data and the quality of the gathered data may influence the quality of the results given by algorithms. Finding, verifying and tabulating logistic information and data is a difficult and time consuming task and requires a big effort and accuracy in order to guarantee correct optimization results. In this study we address the issue of gathering automatically, from geographic information systems (GIS), real world distances between nodes in a logistic network. We will show how the efficiency is of paramount importance when retrieving large amounts of logistic data and how our proposed architecture makes possible real data retrieving in reasonable times. Our final intention with this research is to stir the research filed in logistic optimization and to move it away from the well known and common assumption that distances between nodes are assumed to be Euclidean.

2. Travel distances in common logistic problems

We first briefly describe some of the most well known and studied logistic problems in order to highlight the fact that transportation distances between nodes need to be known before solving their mathematical formulation or before applying any other optimization method.

New facility location problems deal with the determination of the optimal new facility location in a network configuration, so that the total transportation cost is kept to minimum. In *p-centre models* the aim is to locate *p* facilities on a transportation network in such a way that the maximum travel time from a user to the closest facility is minimized. In *location-covering models* the aim is to determine the least-cost set of facilities such that each user can be reached within a given maximum travel time from the closest facility. *Freight traffic assignment problems (TAPs)* consist of determining a least-cost routing of goods over a network of transportation services from their origins (e.g. manufacturing plants) to their destinations (e.g. retail outlets) and are modeled as a minimum cost flow problem. (Complete descriptions in Ghiani, Laporte and Musmanno, 2004) *Vehicle routing problems* are central to distribution management and logistics and typically must be solved routinely by carriers. They involve finding efficient routes for vehicles along transportation networks, in order to minimize route length, service cost, travel time, number of vehicles, etc. In practice, several variants of the problem exist because of the diversity of operating rules and constraints encountered in real-life applications. (Laporte, 2007)

All these problems are usually modeled on a directed, undirected or mixed graph $G(V, A, E)$ with V being the set of vertices, representing terminals, plants, warehouses, demand points, road intersections etc. A and E denote the sets of arcs and edges, respectively and model transportation links, road connections, material flows, etc. Arc and edges are associated with transportation costs that typically depend on travel distances between vertices, travel times, quantity of moved material or any other measure or performance indicator.

3. Using Geographic Information Systems (GIS) for real travel distances data retrieval

The number of nodes (facilities) in a transportation network can exceed several thousands depending on company dimension and transportation sector. Urban waste collection, food distribution, parcel shipment and delivery, etc. are examples of applications where the transportation network can reach tens of thousands of nodes. In order to keep algorithms computation times and hardware requirements acceptable, several techniques of data gathering and clustering can be found in literature. With regard to travel distances between nodes in transportation networks the typical simplification that can be found in literature is the use of the Euclidian distance metric or the spherical geometry. The Euclidean metric is the distance between two points in the plane that one would measure with a ruler, and is given by the Pythagorean formula. The spherical geometry deals with great-circle distances or orthodromic distances that are shortest distances between any two points on the surface of a sphere (great circle distance). In general, Euclidean and orthodromic distances are a lower bound estimation of the real shortest path distances between two locations on Earth. Figure 1 shows the orthodomic distance and a shortest path distance provided by a public GIS, calculated between two major Spanish cities: Valencia and Barcelona.

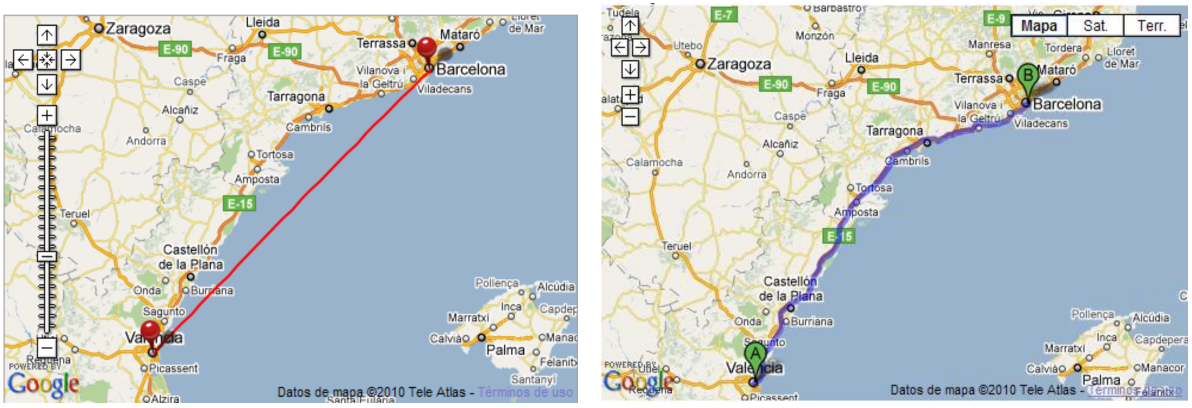


Figure 1: Route between Valencia and Barcelona: Left orthodomic distance of 303.365 km., Right shortest path distance of 349 km.

One could think that orthodromic distances are acceptable. However, we have previously demonstrated (Rodríguez and Ruiz 2009) that there is a tremendous effect in solution quality between using real distances and orthodromic approximations.

Real world distances and travel times actually depend on the physical transportation infrastructure that connects each couple of nodes in a logistic network. Geographic Information Systems (GIS) can provide real distance information derived from digital road networks in order to accurately estimate distances between each couple of nodes in a logistic network. Originally, when GIS data was mostly based on large computers and stored in internal records, GIS software was basically a stand-alone product. With the increased access to the Internet, demand for distributed geographic data grew and GIS software gradually changed its entire outlook to the delivery of data over a network. Public access to geographic information is nowadays provided by online resources such as Google Earth and Microsoft Bing, among others. Users can easily include their APIs in custom applications using web service requests or http requests and the Internet.

3.1. Obtaining large real-world distance matrices with threaded algorithms

With the arrival of web-based GIS it is possible to include in user applications the logic for calculating real distances between locations (latitude, longitude) and to automatically obtain large real distance matrixes for all types of logistics optimizations. This logic basically consists of executing minimum path requests to a web-based GIS. A path request typically contains the geographical coordinates of *from-to* locations, the desired type of minimum path (fastest or shortest) and an indication on the travel method (walking, by car, etc.). Minimum path calculations are executed in GIS servers and minimum path algorithms, such as Dijkstra's method are used. In this study, first we address the problem of optimizing the process of the automatic collection of shortest paths from a web-based GIS, minimizing GIS response waiting times. For a transportation network with 1000 nodes, 999,000 paths should be calculated, one for each couple of nodes (1000×999). Several days could be necessary to complete the process if the 999,000 path requests are run sequentially. A very effective way to accelerate the process of obtaining big distance matrixes from GIS remote servers is the utilization of multithreading techniques in user application logics. With multithreading, the total GIS response waiting times are minimized because new requests can be started during the waiting times of already started requests. The idea is to start as many new requests as possible, each one in a separate thread, during GIS response waiting times, in order to have a pipeline-like execution process. Let us consider the situation of a pool of similar requests to run, each one having the same $T_{\text{Waiting GIS Response}}$ times. If n is the number of requests that can be started in this time window, total waiting time of the *n* requests is $2 \cdot T_{\text{Waiting GIS Response}}$, as shown in Figure 2:

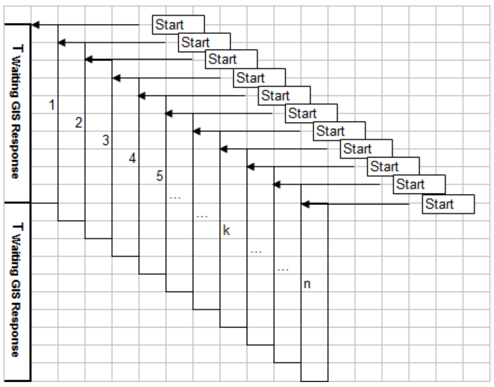


Figure 2: Example of a total response waiting time for *n* GIS requests started in interval $T_{\text{Waiting GIS Response}}$.

If requests are executed sequentially, total waiting time should be $n \cdot T_{\text{Waiting GIS Response}}$. The acceleration in the total waiting time is:

$$\text{Acceleration} = n \cdot T_{\text{Waiting GIS Response}} - 2 \cdot T_{\text{Waiting GIS Response}} = (n-2) \cdot T_{\text{Waiting GIS Response}}$$

The above theoretical result shows that the maximum acceleration that we can give to the total GIS response waiting times depends only on $T_{\text{Waiting GIS Response}}$ and *n*, number of requests that can be started in this time window. Note also that in modern multi-core computers, nothing precludes us from launching more than one initial request to the web-based GIS and therefore the schema shown in Figure 2 can be replicated a number of times and the acceleration factor can be increased accordingly.

For big instances with thousands of requests to calculate, the differences in response waiting times between different web-based GIS that depend on the efficiency of shortest path algorithms and GIS server optimizations can be dampened by the above mechanism of accelerations. Our experiments confirmed the existence of an upper bound limit of the possible acceleration that can be given to the calculation process. In real life distance calculations, GIS waiting times are variable and depend on request complexity given from the distances between the two points, origin and destination of the path. The number of requests that can be started during GIS responses waiting times is variable and the calculation of the best number of threads to run simultaneously can only be done on a statistical basis.

In order to fix the maximum number of threads to use we executed full distance matrix calculations for different sizes *n* of transportation networks where *n* is the number of nodes and different number of threads as shown in Figure 3.

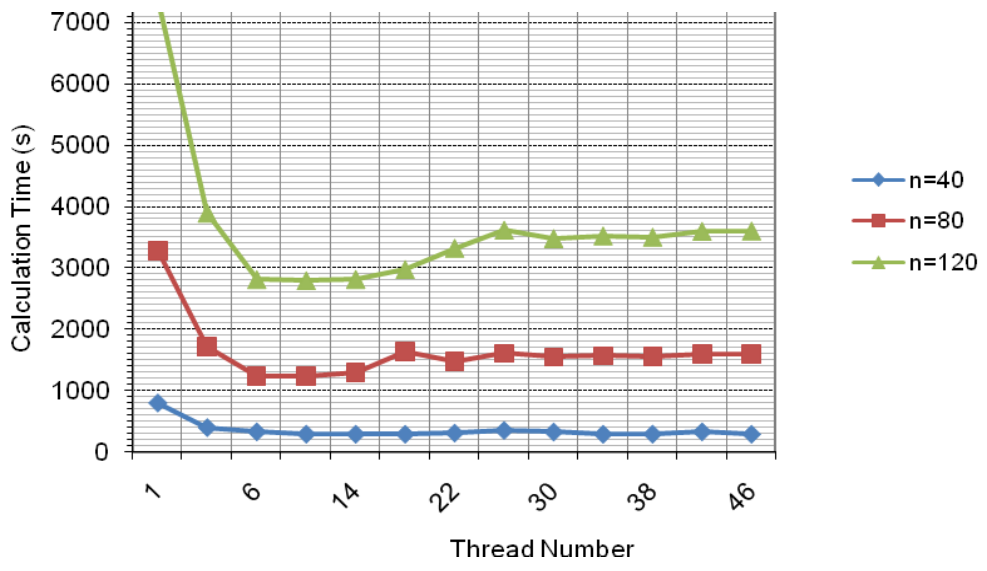


Figure 3: Calculation time vs. number of threads when calculating distance matrixes

The experiments were run in a single-core CPU 2.5 GHz virtual machine with 2 GB of RAM and the code was written in C# 3.0 under the .NET platform 3.5.

The best results correspond to a number of threads equal to 10. For greater number of threads, calculations times can be worse because of inefficiencies caused by having many threads running in the same single-core CPU. Recall that all threads compete for the same processor and context-change and other cache-related inefficiencies quickly degrade computing resources.

3.2. An effective algorithm for calculating real distances from a web-based GIS

The acceleration of real distance calculation process can be greatly improved if the web-based GIS minimum path requests are built in an intelligent way. Public GIS like Google Maps and Microsoft Bing allow for multi-point minimum path requests to be executed. It is possible to request for a path calculation that passes through a certain number of fixed locations i.e.. going from location A to location B passing by locations C, D, E, etc. GIS return minimum distances and travel times for each of the ways that compose the whole path, AC, CD, DE,…,EB. Given a node network of n nodes, the shortest path matrix, with all paths between each couple of nodes, has $n\cdot(n-1)$ elements. Theoretically, $n\cdot(n-1)$ minimum path requests should be launched to a remote web-based GIS. We have created an algorithm that constructs a unique multipoint sequence that contains all the $n\cdot$ $(n-1)$ paths as intermediate paths and does not repeat any path.

All the n nodes are numbered from 1 to n and are increasingly ordered 1, 2, 3,….,n-1, n. The objective is to establish a criterion for visiting all the n nodes. We first iteratively build a starting sequence of numbers called $S1_n$ in the following manner.

Step 1: visited node = 1; $S1_n = \{1\}$

Step 2: Next visited node = 2; $S1_n = \{1, 2\}$

At each step, we try to include in the sequence already visited nodes in order to obtain the return ways. The candidate at this step for the next node to visit is node 1. The algorithm consists in accepting only nodes whose distance from the last visited node is ≥ 2. This distance is calculated as the difference between node numbers. If a backward node is accepted, its number and the number of the last visited node are added to the sequence.

Following the example, at this step, $2-1 < 2$, so node 1 is not accepted in the sequence and Step 2 finishes.

Step 3: Next visited node = 3; $S1_n = \{1, 2, 3\}$

Look for backward elements:

$3-1 \geq 2$ so add to sequence 1 and 3, therefore $S1_n = \{1, 2, 3, 1, 3\}$

$3-2 < 2$, step 3 finishes.

Step 4: next visited node = 4; $S1_n = \{1, 2, 3, 1, 3, 4\}$

Look for backward elements:

$4-1 \geq 2$ so add to sequence node 1 and 4, therefore $S1_n = \{1, 2, 3, 1, 3, 4, 1, 4\}$

$4-2 \geq 2$ so add to sequence node 2 and 4, therefore $S1_n = \{1, 2, 3, 1, 3, 4, 1, 4, 2, 4\}$, step 4 finishes.

…

Step n, Next visited node = n; $S1_n = \{1, 2, 3, 1, 3, 4, 1, 4, 2, 4, 5, 1, 5, 2, 5, 3, 5,\ldots, n\}$

Look for backward elements:

$n-1 \geq 2$ so add to sequence node n and 1, therefore $S1_n = \{ 1, 2, 3, 1, 3, 4, 1, 4, 2, 4, 5, 1, 5, 2, 5, 3, 5,\ldots, n, 1, n\}$

$n-2 \geq 2$ so add to sequence node n and 2, therefore $S1_n =\{ 1, 2, 3, 1, 3, 4, 1, 4, 2, 4, 5, 1, 5, 2, 5, 3, 5,\ldots,n, 1, n, 2, n\}$

$n-(n-2) \geq 2$ so add to sequence node n and node $(n-2)$ therefore final $S1_n$ is

$\{1, 2, 3, 1, 3, 4, 1, 4, 2, 4, 5, 1, 5, 2, 5, 3, 5,\ldots,n, 1, n, 2, n,\ldots, n-2, n\}$ and step n finishes.

At this point we build a sequence $S2_n$ that contains all the return ways among nodes that have distance 1 and were not accepted in sequence 1.

$S2_n = n, n-1, n-2,\ldots, 1$

The final sequence S_n is given by

$$S_n = S1n \setminus \{n\} \cup S2_n \tag{1}$$

$S_n = \{1, 2, 3, 1, 3, 4, 1, 4, 2, 4, 5, 1, 5, 2, 5, 3, 5,\ldots, n, 1, n, 2, n,\ldots, n-2, n, n-1, n-2,\ldots,1\}$

Let us add a new node to the problem, node $n+1$:

$S1_{n+1}$ is then $\{1, 2, 3, 1,3, 4, 1, 4, 2, 4, 5, 1, 5, 2, 5, 3, 5\ldots n, 1, n, 2, n,\ldots, n-2,n, n+1, 1, n+1, 2, n+1, 3, n+1,\ldots, n-1 ,n+1 \} = S1_n \cup \{ n+1, 1, n+1, 2, n+1, 3, n+1,\ldots,n-1,n+1\}$

$S2_{n+1} = n+1, n, n-1, n-2,\ldots\ldots,1 = \{n+1\} \cup S2_n$

Generalizing:

$$S1_{k+1} = S1_k \cup \{k+1, 1, k+1, 2, k+1, 3,\ldots, k+1, k-1, k+1\} \quad (2)$$

Observe that the last element is always n+1 and the distance between each node k+1 and the next node in sequence is 2.(Example: $S1_5 = S1_4 \cup \{5, 1, 5, 2, 5, 3, 5\} = \{1, 2, 3, 1, 3, 4, 1, 4, 2, 4\} \cup \{5, 1, 5, 2, 5, 3, 5\}$).

$$S2_{k+1} = \{k+1\} \cup S2_k \quad (3)$$

$S_{k+1} = S1_{k+1} \setminus \{k+1\} \cup S2_{k+1}$ as stated in (1), therefore replacing (2) and (3) we have that:

$$S_{k+1} = S1_k \cup \{k+1, 1, k+1, 2, k+1, 3, k+1, \ldots, k-1, k+1\}_1 \cup S2_k \quad (4)$$

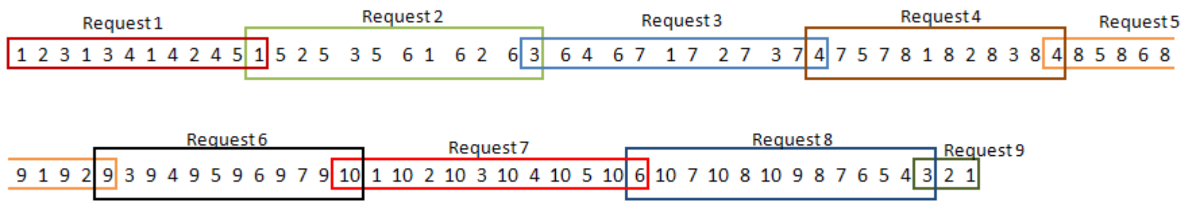
Given n nodes, the sequence can be created in n iterations, iterating k from 1 to n with initial condition $S1_1 = 1$ and $S2_1=1$.

An example with 10 nodes numbered from 1 to 10:

1,2,3,1,3,4,1,4,2,4,5,1,5,2,5,3,5,6,1,6,2,6,3,6,4,6,7,1,7,2,7,3,7,4,7,5,7,8,1,8,2,8,3,8,4,8,5,8,6,8,9,1,9,2,9,3,9,4,9,5,9,6,9,7,9,10,1,10,2,10,3,10,4,10,5,10,6,10,7,10,8,10,9,8,7,6,5,4,3,2,1.

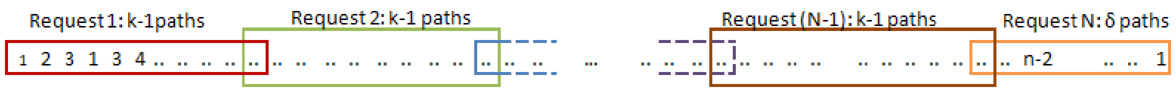
Moreover, if we have calculated previously all the distances for n nodes and at a certain point we add to the transportation matrix some α new nodes, (1), (2), (3) and (4) can be used to calculate the optimized unique sequence that contains only the new ways added to the problem. For doing this we only need to numerate all nodes whose full distance matrix was already calculated from 1 to n and the new ones form n+1 to n+α and eliminate from the sequence $S_{(n+\alpha)}$ the sequence S_n.

From a web-based GIS point of view, the sequence is interpreted like a multi stop path request that starts in the first node of the sequence, finishes in the last node of the sequence, and passes by intermediate nodes in the same order as in the request. Web-based GIS have practical limits in the maximum number of intermediate stops so it could be necessary to cut the unique request into a set of smaller requests of maximum k nodes in each request, if k is the practical limit. In the following example it is shown how to cut the sequence if the practical limit is of 12 nodes.



The whole sequence it is decomposed in 9 multi path requests.

All requests contain 12 nodes, which is equivalent to 11 intermediate paths, except for the last one that contains the remaining 3 nodes (2 intermediate paths). Request 1 is a multi path request that starts in node 1, finishes in node 1 and passes by the intermediate nodes delimited by the red rectangle. Request 2 starts in node 1, ends in node 3 and passes by the intermediate nodes delimited by the green rectangle. Observe that it is necessary to start in node 1, the last of the previous request, and not in node 5, in order to not loose the intermediate path 1-5 of the original sequence. The same can be said for the other requests. Given a node network of *n* nodes, and a practical limit of maximum *k* nodes in each multi path request, let us calculate the number *N* of total requests that will be created by splitting the first unique multipoint sequence, that contains all the *n*·(*n*-1) minimum cost paths. The first *N*-1 requests contain each one of the *k* nodes and (*k*-1) intermediate minimum cost paths like it is shown in the following example:



Observe that the last request *N* contains the remaining δ number of paths where $0 \le \delta < k$-1.

For δ=0:

$$N(k-1) = n(n-1) \Rightarrow N = \frac{n(n-1)}{k-1}$$

For 0< δ < *k*-1:

$$(N-1)(k-1) + \delta = n(n-1) \Rightarrow N = \frac{n(n-1)}{k-1} + 1 - \frac{\delta}{k-1}; 0 < \delta < k-1, therefore: 0 < \frac{\delta}{k-1} < 1 \Rightarrow$$

$$0 < 1 - \frac{\delta}{k-1} = \varepsilon < 1$$

$$N = \frac{n(n-1)}{k-1} + \varepsilon \Rightarrow \frac{n(n-1)}{k-1} < \frac{n(n-1)}{k-1} + \varepsilon < \frac{n(n-1)}{k-1} + 1 \Rightarrow \frac{n(n-1)}{k-1} < N < \frac{n(n-1)}{k-1} + 1$$

N is an integer number thus: $N = \left\lfloor \frac{n(n-1)}{k-1} + 1 \right\rfloor$

In such a way we pass from the initial $n(n-1)$ simple path GIS requests, to $N = \left\lfloor \frac{n(n-1)}{k-1} + 1 \right\rfloor$ multi-point minimum path requests. Each of these requests can then be executed in a separate thread in order to benefit from the multithreading accelerations as described in the previous sections. The result is a drastic reduction of the duration of the process of real distance calculations. In our experiments we calculated real distances for different problem sizes *n* where *n* ∊ {40, 80,120,160,..., 1040} as it is shown in Table 1. Microsoft Bing Route Web Service was used for obtaining real shortest path distances.

For each problem size we measured the calculation times running the $n(n-1)$ requests sequentially and in multithreading (10 threads), and the optimized $N = \left\lfloor \frac{n(n-1)}{k-1} + 1 \right\rfloor$ requests, for *k* = 24, run in multithreading (10 threads). Microsoft Bing Route Web Service exposes an API for developers that permit to easily create and run shortest path request to Microsoft servers calling a remote web service. In Table 1 are reported the execution times in hours and in minutes for each problem size and method of distance matrix calculation.

Table 1: Execution times in hours and in minutes for each problem size *n* and method for distance matrix calculation.

n	paths	Calculation Time (h)			Calculation Time (min)		
		1 CPU Sequential	1 CPU 10 threads	1 CPU 10 threads requests of 24 nodes	1 CPU Sequential	1 CPU 10 threads	1 CPU 10 threads requests of 24 nodes
40	1560	0.22	0.09	0.04	13	5	3
80	6320	0.95	0.35	0.18	53	21	11
120	14280	2.14	0.79	0.40	119	48	24
160	25440	3.82	1.41	0.71	212	85	42
200	39800	5.97	2.21	1.11	332	133	66
240	57360	8.60	3.19	1.59	478	191	96
280	78120	11.72	4.34	2.17	651	260	130
320	102080	15.31	5.67	2.84	851	340	170
360	129240	19.39	7.18	3.59	1077	431	215
400	159600	23.94	8.87	4.43	1330	532	266
440	193160	28.97	10.73	5.37	1610	644	322
480	229920	34.49	12.77	6.39	1916	766	383
520	269880	40.48	14.99	7.50	2249	900	450
560	313040	46.96	17.39	8.70	2609	1043	522
600	359400	53.91	19.97	9.98	2995	1198	599
640	408960	61.34	22.72	11.36	3408	1363	682
680	461720	69.26	25.65	12.83	3848	1539	770
720	517680	77.65	28.76	14.38	4314	1726	863
760	576840	86.53	32.05	16.02	4807	1923	961
800	639200	95.88	35.51	17.76	5327	2131	1065
840	704760	105.71	39.15	19.58	5873	2349	1175
880	773520	116.03	42.97	21.49	6446	2578	1289
920	845480	126.82	46.97	23.49	7046	2818	1409
960	920640	138.10	51.15	25.57	7672	3069	1534
1000	999000	149.85	55.50	27.75	8325	3330	1665
1040	1080560	162.08	60.03	30.02	9005	3602	1801

The following figure shows the results depicted in a chart.

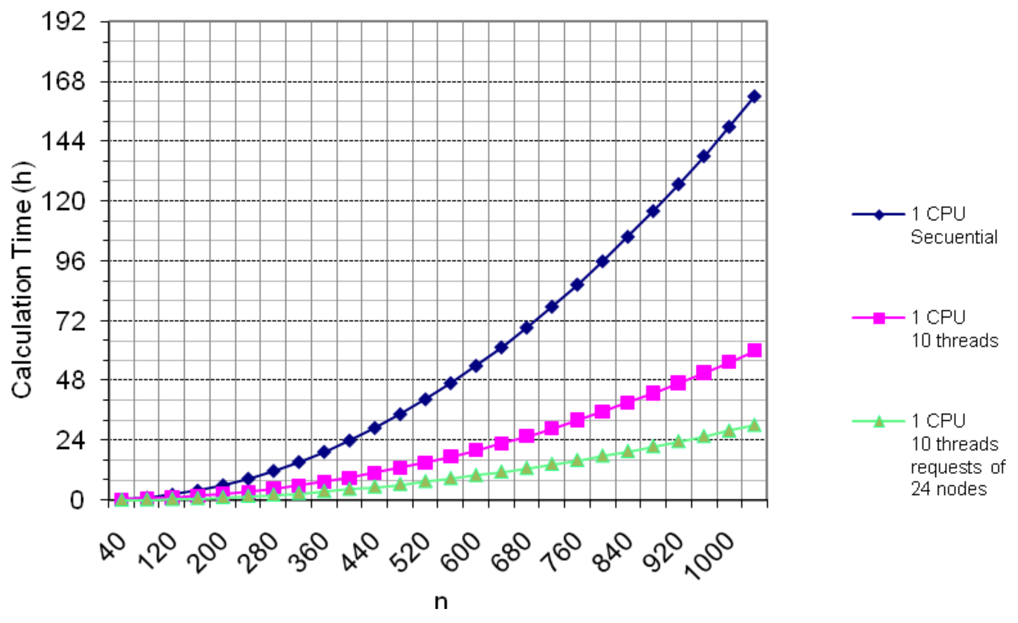


Figure 4: Calculation time vs. size of the transportation matrix *n* and method for calculating the distance matrixes.

For large problem instances like *n*=1000 nodes execution time passes from 6 days and 6 hours approximately to 1 day and 4 hours.

In the following figure, time savings in hours obtained with multithreading and multithreading plus multipoint requests are depicted in a chart.

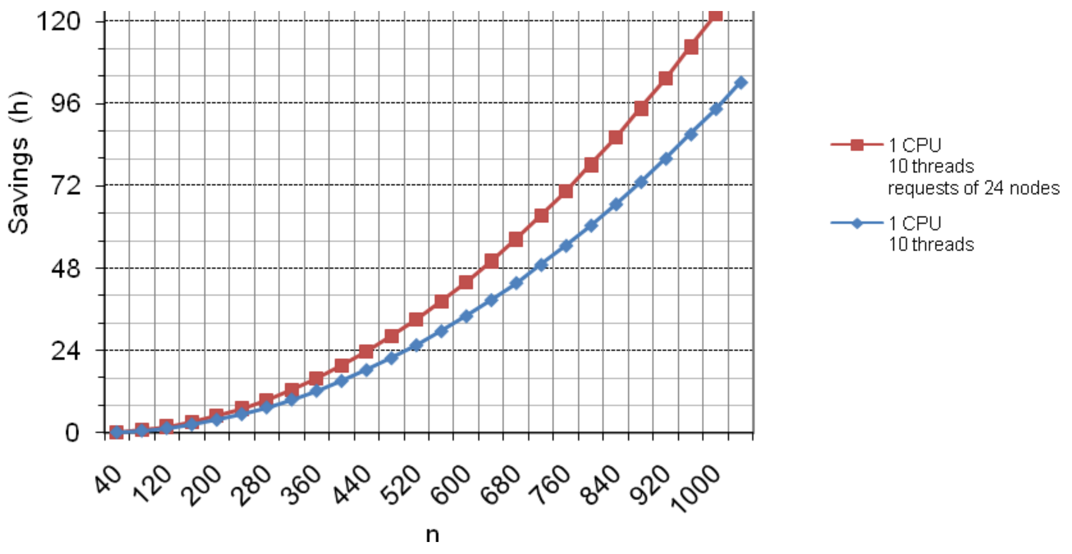


Figure 5: Calculation time vs. size of the transportation matrix *n* and method for calculating the distance matrixes. Multi-threading with and without multipoint requests.

Time savings show to be 63% when multithreading technique is used to calculate full distance matrix and 80% when the multithreading technique is combined with optimized multi point requests.

4. Conclusions

Calculating real distances instead of orthodromic simplifications is a must when dealing with complex logistic management problems. Nowadays, it is perfectly possible to do so by using publicly available web-based GIS systems capable of calculating real distances between nodes in a transportation network efficiently. However, in real logistic problems, with possibly thousands of nodes, the matrixes containing real distances (and/or times) between nodes are challenging to calculate. Actually, the time needed to obtain such matrixes can easily be orders of magnitude bigger than the time needed for solving complex logistic management problems with elaborated metaheuristics. For some reason, the scientific literature has neglected this important fact. In this work we have shown two techniques, namely, threading retrieval and intelligent request construction, that, when combined, allow for drastic improvements in the times needed to calculate such matrixes.

5. Future work

Currently, we are extending our algorithms and implementations to parallel execution in modern CPUs with more than one single core. We started this research work with a single-core CPU in order to create a solid base for further optimization. We expect to further reduce distance matrix execution times, as many times as the number of cores present in the machine.

Acknowledgements

This work is partially funded by the Spanish Ministry of Science and Innovation, under the project “SMPA - Advanced Parallel Multiobjective Sequencing: Practical and Theoretical Advances” with reference DPI2008-03511/DPI. The authors should also thank the IMPIVA - Institute for the Small and Medium Valencian Enterprise, for the project OSC with reference IMIDIC/2009/198 and the Polytechnic University of Valencia, for the project PPAR with reference 3147.

References

Ghiani, G.; Laporte, G.; Musmanno, R. (2004). Introduction to logistic systems planning and control. John Wiley & Sons.

Laporte, G. (2007). What you should know about the vehicle routing problem, Naval Research Logistics, Vol 54, pp. 811-819.

Rodríguez, A.; Ruiz, R. (2009). El impacto de la asimetría en la resolución de problemas de distribución y rutas. In Spanish. 3rd International Conference on Industrial Engineering and Management. XIII Congreso de Ingeniería de Organización. Barcelona-Terrassa, 2009, pp. 1645-1654