Tile Map Size Optimization for Real World Routing By Using Differential Evolution

Andrés Camero*, Javier Arellano-Verdejo†, Christian Cintrano‡ and Enrique Alba§
Department of Computer Science, University of Málaga, Spain
*andrescamero@uma.es, †javerdejo@lcc.uma.es,
‡cintrano@lcc.uma.es, §eat@lcc.uma.es

***Abstract*—Finding the shortest path between two places is a well known problem in road traveling. While most of the work done up to this moment is focused on algorithmics, efficiently managing the information has received significantly less attention. Nevertheless, real world problems like road map routing present a challenge due to the impact that the immense size of the map has over the temporal complexity of the routing algorithms.**

In this work we propose a strategy for efficiently computing the shortest path in real road maps based on data managing: the *tile map* partitioning. To recreate a real scenario, we implemented a routing system and we tested our strategy using the road map of the Province of Málaga, Spain. Using a Differential Evolution we found the optimal tile size and prove that significant time reductions can be achieved by using the tile map partitioning.

***Index Terms*—Optimization, Heuristic algorithms, Shortest path problem, Evolutionary computation.**

I. INTRODUCTION

What is the shortest way from one place to another? This simple question, yet difficult to answer, has attracted the interest of scientific investigation for decades: the shortest-path (SP) problem. While many algorithms have been proposed [1]–[7], less attention has been paid to efficiently managing the data used to calculate the SP [8], [9]. Moreover, the majority of the algorithms assume that all the information is stored in-memory and many require the computation of additional information [10]. The latter may be explained by the fact that a decade ago there was a little available information regarding road maps, thus the need for sophisticated techniques for managing immense amounts of data was not a priority. However, since the irruption of the smart phone in the market and the explosive growth of the consumption of online services, the generation of data scaled up by thousands, leading not only to the need for new algorithms, but also to the design and implementation of new efficient data management techniques.

For example, if we try to find the SP between two places in the Province of Málaga, Spain, the size of the map may rise up to hundreds of thousands streets, and even worse: streets are constantly changing! According to the statistics of Open Street Map (OSM), more than 41 million intersections and nearly 7 million streets are added each month worldwide[1]. Considering this, we believe that the SP problem has to be faced in a holistic way, considering not only the algorithm, but also the data management as a whole problem.

In this study we propose an approach for dividing a map into optimal logic geographic sectors in order to minimize the routing generation time: the *Tile Map Partitioning*. The *Optimal Tile Size* (i.e. the partitioning size that minimizes the time needed for computing SP) is obtained by using a *Differential Evolution Algorithm*. As shown in the conclusions of this study, this approach offers a balance between time and memory usage.

The remainder of this work is as follows: Section II briefly reviews the state-of-the-art; Section III describes the tiles partitioning and outlines the optimal tiles configuration problem; Section IV presents the results; Section V gives the main conclusions, and Section VI proposes future work.

II. SHORTEST-PATH COMPUTATION

In this section we summarize the basic concepts used in this study (Section II-A), as well as present the state-of-the-art proposals for efficiently managing road maps dedicated to SP computation (Section II-B).

Nowadays many people relies on computer systems (GPS navigation systems[2] or software services[3]) for finding the best route to get to their destinations, not only because of the complexity of road maps, but also because of the *quality* of the route. While the *routing* made by one person depends on his previous knowledge (and experience), computer systems may offer a broader approach, including all the road network, live traffic, incidents information, among others.

Furthermore, the amount of information regarding mobility is rapidly increasing, as well as the amount of people with ubiquitous access to route finding systems. As a consequence, the need for routing systems capable of processing enormous amounts of data in a real time fashion appears.

A. Shortest-Path Problem in Road Maps

A road map may be modeled as a directed weighted graph $G = (V, E, w)$, whose vertices ($v \in V$) correspond to the intersections, the edges ($e \in E$) correspond to the *streets* or links between intersections, and $w(e)$ is a function that assigns a non-negative value to an edge. Then, given a path $P = \{e_0, e_1, ..., e_n\}$, $\forall e_i \in E$ $(0 \leq i \leq n)$, the **length** $W(P)$ of the path P is defined as:

[1] 30-Jan-2017, http://wiki.openstreetmap.org/wiki/Stats

[2] http://www.garmin.com/, https://www.tomtom.com/

[3] https://www.google.com/maps

978-1-5090-4601-0/17/$31.00

$$W(P) = \sum_{i=0}^{n} w(e_i) \quad (1)$$

The **distance** from a vertex v_i to a vertex v_j ($v_i, v_j \in G$), denoted as $\delta(v_i, v_j)$, is defined as the length of the minimum length path (shortest path) from v_i to v_j. In general, the *length* is interpreted as a *cost*, including not just physical distances, but delays in traffic, user preferences, pollution levels, and any other goal for a modern investigation.

SP problem is divided into three main sub-problems: i) finding the SP between two given vertices or *point-to-point shortest path* (P2PSP), ii) finding the distance from one vertex to all other vertices or *single source shortest path* (SSSP), and iii) finding the distance between all pairs of vertices or *all pairs shortest path* (APSP).

The reference algorithm to solve SP problems is the (commonly known as) algorithm of *Dijkstra* [1]. This algorithm is also theoretically the most efficient one for solving SSSP [11]. Usually *A** (or A-Star) [2] is used as a benchmark for P2PSP (if an heuristic distance function is available), and the algorithm proposed by Floyd, *Algorithm 97* [3], is commonly used as a point of reference in APSP.

There are several implementations of the above mentioned algorithms that improve the performance (time) by implementing a specific data structure [11], [12]; by using domain specific information (e.g. road network topology) [13], [14]; or by doing the precalculation of variables (distance, landmarks, among others) [4], [15]. However, most of the efforts are gathered into improving *in memory* computations and barely none of the proposals take into account (real) memory restrictions.

B. Data Management

Although efficiently managing data has gained less attention than algorithmics, there are innovative proposals [8], [9], [16]. Regarding technology, the rise of social and information networks in the past decades lead to the creation of the *Graph Database* System (GDB), a specifically design database (DB) implementation for managing graphs. Among GDB implementations, the most popular is *pgRouting*, which runs on top of PostgreSQL DB[4]. However, in 2014 according to Miler et al. [16], "graph database management systems [...] are not suitable for full graph traversal purposes". Moreover, they proved that in-memory processing outperforms GDB path finding, but with the disclaimer that the memory needed in real applications may present an issue.

In 2011, Efentakis et al. [9] proposed map partitioning by road hierarchies, and by using real world road maps and DB storage proved that partitioning improved the time for computing a P2PSP. More recently, in 2015 Aridihi et al. [8] proposed a *MapReduced* based strategy for computing P2PSP over large scale networks, showing excellent time performance, but at the cost of not guaranteeing optimal solutions and adding a lot of preprocessing.

[4]https://www.postgresql.org/

In the commercial field, a successful (and popular) SP implementation is Google Maps. However, despite the great results (in terms of user experience), there is almost no information of the techniques used to solve the problem. According to Brumitt[5], the implementation relies on *MapReduce*, but further details have not been unclassified.

Taking into account this ecosystem of approaches, we decided to explore a new partitioning approach based on geographic properties, with the intention to find a balance between computation time and memory usage.

III. Optimal Tiles Size Problem

In this section we describe the *Optimal Tiles Size* (OTS) problem. First, we show how we represented the map of the city in function of tiles, then, we show the problem definition (OTS problem) as well as an analysis of the size of the search space. Finally, we present the Differential Evolution Algorithm for solving the OTS problem. To test our proposal we used the road map of nearly the Province of Málaga, Spain (as shown in Section IV, 4500 km^2).

A. Tile Map Representation

Considering the representation of a road map as a weighted graph $G_m = (V_m, E_m, w_m)$, whose vertices $v \in V_m$ are geolocated in the position (v_{lat}, v_{lon}); we define a *Tile* as the graph $G_t \subseteq G_m$, whose vertices $u \in V_m$ are geolocated within the region described by the tile, i.e. where $min_{lat} \leq u_{lat} < max_{lat}$, and $min_{lon} \leq u_{lon} < max_{lon}$, where $R_t = ((min_{lat}, max_{lat}), (min_{lon}, max_{lon}))$ is the region described by G_t. The size of a tile is defined $s_t = (\mid max_{lat} - min_{lat} \mid, \mid max_{lon} - min_{lon} \mid)$.

We define a *Tile Map* as the set of Tiles $M = \{G_1, ..., G_n\}$ ($n \in \mathbb{N}$), where $G_i \subseteq G_m$ ($1 \leq i \leq n$) are tiles that have the same size $s = s_i$, and do not overlap, that is to say: $R_i \bigcap R_j = \emptyset, \quad \forall i \neq j$, and cover the whole map $\bigcup_i^n G_i = G_m$. In other words, a tile map is made by partitioning a map into n uniform blocks (or tiles).

Figure 1 illustrates a map partitioned into a set of tiles, i.e. a Tile Map. Each region of the map is represented by one tile.

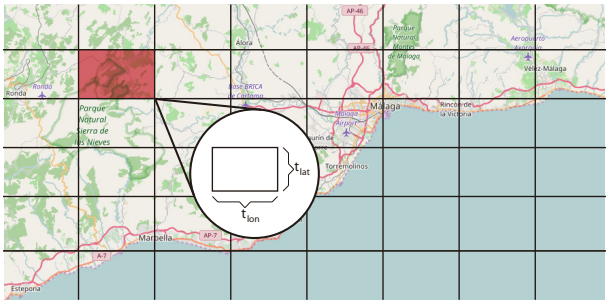

Fig. 1. The region considered to test our approach, partitioned into tiles.

[5]https://googleblog.blogspot.com.es/2007/11/road-to-better-path-finding.html

B. Tile Size Problem

Considering that a map may be partitioned into several different tile sizes, and that our motivation is to improve the performance of the SP computation, we state the OTS problem. The OTS problem consists in finding the tile size that is optimal in terms of performance (refer to Equation 2). Formally, given a road map represented by the graph G_m, the problem is to find the Tile Map M that minimizes the time needed to compute the P2PSP between two points $\delta(v_i, v_j)$, where $v_i, v_j \in V_m$.

$$\text{minimize } \frac{1}{N}\sum_{k}^{N}(\phi(P_k) + \rho(P_k)) \tag{2}$$

$$\text{subject to } s = s_i, 1 \leq i \leq n \tag{3}$$

$$R_i \bigcap R_j = \emptyset, 1 \leq i, j \leq n, i \neq j \tag{4}$$

$$\bigcup_{i}^{n} G_i = G_m \tag{5}$$

Where ρ is the time for computing an itinerary, ϕ is the time for preparing (i.e. loading from DB into memory) the map, P_k is an origin-destination pair, and N is the number of routes been computed.

C. Proposed Algorithm

In order to solve the OTS problem, we designed an algorithm which is based on "DE/rand/1/bin" Differential Evolution Algorithm. DE [17] is a population-based optimizer for minimizing possibly nonlinear and non-differentiable continuous space functions. Typically, DE encodes all parameters as floating-point numbers. In this work, individuals (x) are encoded as two dimensions vectors which represents the latitude and longitude size of each tile (Equation 6). Therefore, the population (P^g) is represented by a matrix, where the number of rows is equal to the size of the population and each column is a solution, as shown in Equation 7 (in the case of the proposed algorithm, the size of the population is five individuals).

$$x = \{t_{lat}, t_{lon}\}, t_{lat}, t_{lon} \in \mathbb{R}^+ \tag{6}$$

$$P^g_{p,m} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix} \tag{7}$$

Where p corresponds to the size of the population, m is dimensionality of the individuals, and g refers to the (relative) generation. Thus, we may refer to the m-th parameter of the p-th individual of the g-th generation as $x^g_{p,m}$.

To initialize the population (Equation 8), both upper x^{max}_m and lower x^{min}_m bounds for each parameter must be specified. In the case of the algorithm proposed, these values correspond to the tile size (latitude and longitude) of the tile map being represented by the individual (see Figure 1). The function $rand(0,1)$ returns a uniformly distributed random number within the range $[0, 1)$. A new random value is generated for each parameter.

$$x^1_{p,m} = x^{min}_m + rand(0,1)(x^{max}_m - x^{min}_m) \tag{8}$$

To mutate an individual (refer to Equation 9), DE randomly selects three individuals from the population ($a \neq b \neq c$) and adds a scaled vector difference $F(x_a - x_b)$ to a third vector x_c. The scale factor, $F \in (0, 1]$, is a positive real number that controls the rate at which the population evolves.

$$n^g_p = x_c + F(x_a - x_b) \tag{9}$$

DE employs a discrete recombination by crossing each vector with a mutant vector (see Equation 10). The crossover probability, $\lambda \in [0, 1]$, is a user-defined value that controls the portion of parameter values that are copied from the mutant. To determine which source contributes a given parameter, the crossover operator compares λ to the output of a uniform random number $rand(0,1)$. If the $rand(0,1) \leq \lambda$, $n^g_{p,m}$ is inherited from the mutant, otherwise, the parameter $x^g_{p,m}$ is used.

$$t^g_{p,m} = \begin{cases} n^g_{p,m} & if \;\; rand(0,1) \leq \lambda \\ x^g_{p,m} & Otherwise \end{cases} \tag{10}$$

To generate the new population (refer to Equation 11), DE compares the mutated vector x^g_p with the target vector $t^g_{p,m}$. If $t^g_{p,m}$ has an equal or lower fitness value than x^g_p, $t^g_{p,m}$ is preserved; otherwise x^g_p is selected.

$$t^{g+1}_p = \begin{cases} t^g_p & if \;\; fitness(t^g_p) < fitness(x^g_p) \\ x^g_p & Otherwise \end{cases} \tag{11}$$

The *fitness* function (refer to Equation 12) used in Equation 11 is defined as the mean time for computing a predefined set of N routes (P2PSP), where the time for computing a route (or itinerary) includes the time for SP finding (ρ function), as well the time for preparing (i.e. loading from DB into memory) the map (ϕ function).

$$fitness = \frac{1}{N}\sum_{i}^{N}(\phi(P_i) + \rho(P_i)) \tag{12}$$

Once the new population is generated, the process of mutation, recombination and selection is repeated until the optimum is located, or a specified termination criterion is satisfied. In Algorithm 1 the pseudo-code of DE is shown.

IV. EXPERIMENTAL STUDY

In order to recreate a real world scenario, we extracted from OSM the area comprehended by the latitudes 36.47 and 36.86, and the longitudes -4.99 and -4.04, which is nearly the Province of Málaga, Spain. This region includes more than 46,000 intersections and almost 64,000 streets (more than $4500.0\ Km^2$ in total); which are distributed in a dozen cities,

Algorithm 1 Proposed DE/rand/1/bin

g ← 1
P(g) = GenerateInitialPopulation(*NP*)
Evaluate(*P(g)*)
while not TerminationCriterion(*P(g)*) **do**
  for (i=1; i ≤ *NP*; i++) **do**
    Select $x_a \neq x_b \neq x_c$ from *P(g)*
    $n_p^i \leftarrow$ Mutate(x_a, x_b, x_c)
    $t_p^i \leftarrow$ Recombine(n_p^i, x_p^i)
  end for
  for (i=1; i ≤ *NP*; i++) **do**
    $t_g^i \leftarrow$ Select(t_p^i, x_p^i)
  end for
  $g \leftarrow g + 1$
end while

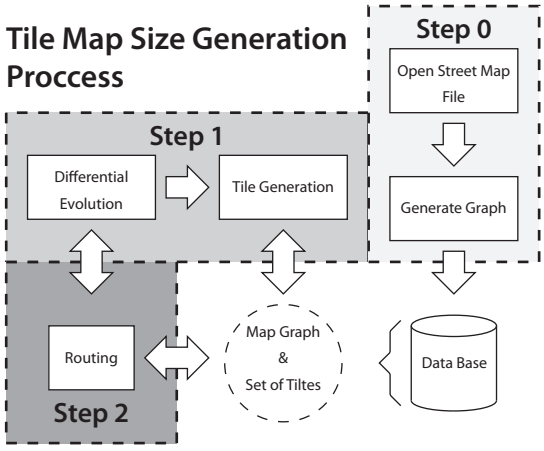

Fig. 2. Obtaining the size of the tiles by using a DE algorithm

including: Málaga, Marbella, Torremolinos, Mijas, Fuengirola, among others. Figure 1 shows the selected region at a high level.

By combining 14 points (or places) located along the main cities, 182 itineraries were generated to test the performance of the routing process. The referred itineraries include urban, as well as interurban travels, and for all itineraries there is at least one connecting path.

We set the search boundaries (length and width of the tiles) to be equal to $s \in [0.001, 5.000]$. This range encompasses from block size (approximately 100 m at the Equator) to wide areas (up to approximately 550 km). The precision of $s_i \in s$ is set to 7 decimal digits (similar to the precision used by OSM). In despite of the relative small search space (smaller than 10^{16} solutions), solving this problem by brute force is not an option, since computing the set of tiles (for this particular map) ranges from half a minute to 24 hours (in the worst case).

A. Solution Stack

We implemented a full solution stack using Java, MySQL, JMetal [18] and Python technologies. Figure 2 shows a high level description of the implemented system. At a glance, the system extracts the data from OSM files (intersections and streets) and loads this data into a custom data model (DB): Step 0. Then, the data (map) is partitioned into a specified tile size, generating and storing in the DB those tiles: Step 1. On each routing request, the system calculates the set of tiles needed to compute the route, loads the set from the DB into memory, and performs the path finding by using an implementation of the algorithm of Dijkstra: Step 2.

For each individual the DE performs the partitioning of the map (Step 1) and compute the P2PSP for all the itineraries defined above (Step 2). Then, the fitness of an individual is calculated as the average time for computing the itineraries.

B. Manual and Random Search

To benchmark the results obtained by the DE algorithm presented in Section III, and as a sanity check, we defined two alternate methods: Manual and Random search. Both methods measure the quality of the solution being evaluated by using the fitness function defined by the DE algorithm (refer to Section III).

The Manual search consists in testing the tile map partitioning with a set of user-defined tile sizes. The selection of the candidates is based on the distance covered by one geographic degree in the latitude and longitude of the region covered by the map, and on the total region covered by the map. The set is specified in Section IV-D. Random search is defined as generating a random tile size within the range s.

C. Baseline Performance

In order to have a baseline performance to compare with, we calculated the fitness of P2PSP finding for the defined set of itineraries without using tile map partitioning. In other words, since we are trying to improve the performance of SP finding in real applications, we set as the baseline the performance of a system that computes the SP over a given map.

Particularly, we implemented two variations of this approach: *Dij-DB* and *Dij-Mem*. Dij-DB implementation stores the map in a DB and loads the data into main memory for computing SP. Dij-Mem variation retains the whole map into main memory. It is important to notice that the last approach is feasible because of memory availability, however this is not a scalable approach (a bigger map may overflow memory).

D. Optimal Tile Size

All experiments have been performed on a PC running 64 bits Ubuntu Linux v16.04 operating system, having an Intel(R) Core(TM) i5 Quad Core processor running at 2.6 GHz with 8 GB of RAM. For the three solvers (DE, Random and Manual) and the two baseline performance approaches, thirty independent runs were performed for the defined map and using the 182 pairs of source-destination, amounting to a grand total of 150 experiments with the real map. The value of each parameter was set to its recommended value (in literature) as shown in Table I.

The Manual search was executed for the combinations of the following values (tile sizes): 0.01, 0.1, 0.2, 0.5, 1, and 10 (36

TABLE I
DE PARAMETERS

Parameter	Value
Population size	5
Crossover probability	0.5
F factor	0.5
Number of routes	182
Map size	4500 km^2

tile size pairs), and the best 30 results were selected. Table II presents a summary of the results (fitness) obtained, while Figure 3 illustrates the performance (fitness) of the approaches tested.

TABLE II
COMPARISON OF THE FITNESS OF THE SOLUTIONS.

Execution	DE	Random	Manual	Dij-DB	Dij-Mem
1	331	800	2696	814	373
2	295	788	618	817	354
3	330	812	478	819	353
4	301	792	588	808	353
5	293	770	618	815	354
6	301	683	618	816	356
7	334	750	466	818	349
8	293	658	438	807	350
9	286	713	581	802	350
10	305	752	738	803	352
11	301	748	801	818	350
12	299	789	795	805	350
13	302	667	618	804	349
14	309	750	437	821	350
15	291	702	569	804	350
16	294	662	726	821	350
17	297	634	863	802	351
18	301	808	861	811	349
19	303	433	458	815	349
20	300	664	449	823	350
21	298	689	599	810	348
22	320	799	775	824	360
23	305	693	967	829	360
24	303	802	950	807	352
25	309	676	458	819	360
26	325	814	447	819	366
27	297	484	600	823	365
28	308	803	770	815	360
29	309	798	971	807	373
30	298	804	935	806	360
Median	**301**	750	618	815	352
Mean	**304.6**	724.7	729.6	813.4	354.9
SD	12.1	92.6	410.3	7.66	**7.0**
Min	**286**	433	437	802	348
Max	**334**	814	2696	829	373

In order to be able to compare the experimental results from all the tested algorithms two statistical analyses were done. The first one (refer to Table II) involves all classic statistical indicators (minimum, maximum, mean, median, and standard deviation); and the second analysis involves non-parametric indexes based on hypothesis tests: Aligned Friedman test and Holm test.

After performing Aligned Friedman test, as it can be seen in Table III, the best rank is for the DE algorithm, which means that the DE algorithm is the best one (among the tested algorithms) for solving this problem. Nevertheless, Friedman test could sometimes fail in identifying the particular

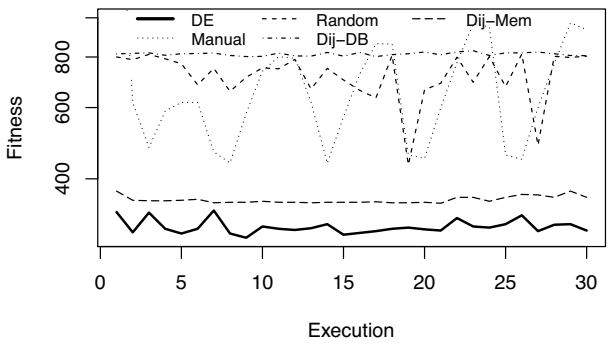

Fig. 3. Fitness comparison of the tested approaches.

differences between the best ranked algorithm (DE) and the other ones. Then, a Holm analysis is used for comparing each pair of algorithms. The confidence level for all comparisons is set to 95% ($p_{value} = 0.05$), which allows us to ensure that algorithms are statistically different if the result lies in $p_{value} < 0.05$. Table IV shows the resulting p_{value} for each set of compared algorithms. The results indicate that p_{value} is lower than 0.05 for all the algorithms, except for Dij-Mem. Therefore, refuting the (null) hypothesis; the DE is then different and better with respect to Manual, Random and Dij-DB algorithms.

TABLE III
AVERAGE RANKINGS OF THE ALGORITHMS (ALIGNED FRIEDMAN)

Algorithm	Ranking
DE	**22.5000**
Dij-Mem	39.2333
Manual	94.3167
Random	97.4167
Dij-DB	124.0333

TABLE IV
ADJUSTED p-VALUES TAKING DE AS CONTROL ALGORITHM

Algorithm	Unadjusted p
Dij-DB	0
Random	0
Manual	0
Dij-Mem	0.135775

E. Comparing the Best Solutions

Since evaluating a tile configuration (computing P2PSP for the defined itineraries) requires the execution of several processes (including reading data from a DB) in a computer, it could happen that other alien running processes could exist, and thus the fitness (time) may be perturbed by these external processes. This is important to consider, since we are talking on small time values. Thus, to compare two *similar* solutions in term of fitness, we have to verify if the difference (in terms of fitness) is significant or not. In other words, we want to assess the actual statistical validity of the differences in times.

We selected three solutions (given by the DE algorithm) whose fitness is in the first quartile. Table V presents the

solutions selected, where *solution* corresponds to the number of the execution informed on Table II, t_{lat} and t_{lon} correspond to the tile size, *Fitness* corresponds to the fitness informed in the latter mentioned table, and *Tiles* shows the number of tiles in which the map was partitioned.

TABLE V
DE SOLUTIONS SELECTED FROM THE FIRST QUARTILE.

Solution	t_{lat}	t_{lon}	Fitness	Tiles
Sol. 2	2.4362724	0.0355379	295	27
Sol. 4	0.2372263	0.0289512	286	66
Sol. 8	1.4609874	0.0270191	293	36

We executed 30 independent times the evaluation of each solution mentioned above. Table VI summarizes the results (fitness) obtained for each independent evaluation.

TABLE VI
FITNESS VARIATIONS OF THE BEST SOLUTIONS.

Execution	Sol. 2	Sol. 4	Sol. 8
1	309	292	299
2	306	294	293
3	304	296	297
4	300	295	296
5	304	291	298
6	303	293	298
7	303	290	295
8	298	290	298
9	299	293	296
10	298	287	299
11	299	293	297
12	301	291	298
13	298	291	297
14	298	290	299
15	299	291	297
16	298	290	297
17	299	291	299
18	298	292	297
19	300	289	295
20	297	312	299
21	295	292	295
22	299	292	297
23	297	294	295
24	299	294	298
25	297	292	297
26	298	298	297
27	296	292	296
28	300	289	295
29	299	289	301
30	302	290	297
Median	299.0	**292.0**	297.0
Mean	299.7	**292.4**	297.1
SD	3.1	4.3	**1.7**
Min	295.0	**287.0**	293.0
Max	309.0	312.0	**301.0**

TABLE VII
AVERAGE RANKINGS OF THE ALGORITHMS

Solution	Ranking
4	**1.0536**
8	2.1250
2	2.8214

As a conclusion from the statistical analysis, Solution 4 is regarded as the best solution, in other words there is a significant difference in terms of fitness (time) among the solutions analyzed. Figure 4 shows the fitness (refer to Table VI), excluding the maximum and minimum values for each solution. The behavior of Solution 4 in terms of fitness is *stable*, and consistently beats solutions 2 and 8.

TABLE VIII
ADJUSTED p-VALUES

Hypothesis	Unadjusted p	p_{Holm}
Sol. 2 vs Sol. 4	0.000000	0.000000
Sol. 4 vs Sol. 8	0.000061	0.000122
Sol. 2 vs Sol. 8	0.009166	0.009166

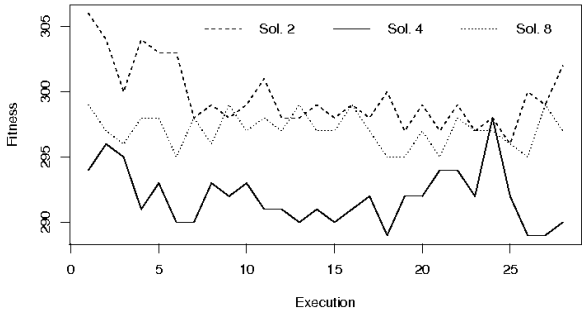

Fig. 4. Fitness variation for the best solutions analyzed.

F. Tile Cache

As mentioned in Section II-B, Miller et al. [16] concluded that the in-memory SP routing outperforms DB routing in terms of time. According to this, we implemented a *Tile Cache*, i.e. a temporal in-memory tile storage, for the calculation of new routes. Using the best solution found by the proposed DE algorithm (best tile map size) to partition the map, we tested this proposal, and as it can be seen in Table IX, the performance of the P2PSP routing improves up to 50% in terms of time.

TABLE IX
BEST TILE MAP SIZE WITH AND WITHOUT CACHE

Property	Cache	No Cache
Median	168.0	292.0
Mean	168.7	292.4
SD	1.1	4.3
Min	167.0	287.0
Max	171.0	312.0

Figure 5 compares same solution with and without cache. The implementation of the tile cache improves the time performance of the system.

V. CONCLUSIONS

In this work we propose a strategy for improving the performance (in terms of time) for the shortest path calculation problem in real applications: the *Tile Map Partitioning*, including not only a real road map, but also considering a real implementation (full solution stack).

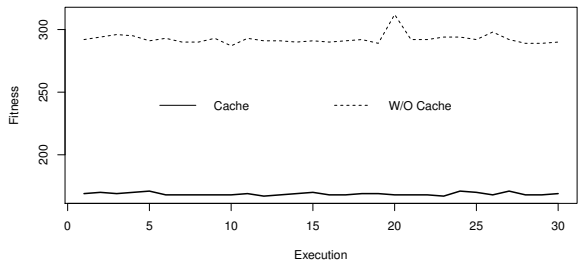

Fig. 5. Fitness variation for the best solutions analyzed.

By using a Differential Evolution algorithm we calculate the *optimal tile size*, demonstrating that changes in the size of the tiles produce significant improvements in the time needed for computing the shortest path between two points.

The results show that efficiently managing the data (map) results into great time improvements. This conclusion is inherent to the algorithm used in this study for routing (Dijkstra) and the fact that the partitioning strategy prunes the graph used for finding the shortest path. However, as a remarkable result, we show that there are significant differences among possible tile sizes.

Since the results show that our proposed approach improves the performance of the system, we move further by implementing a *tile cache* system. As shown in the results of this study, this cache helps to decrease the time needed for calculating the routes by more than a 50%, which we consider to be a clear step towards a real implementation.

VI. Future Work

As part of future work we envision the following roadmap of enhancements: i) design and implement *macro tiles* structures, i.e. generate tile bundles with the tiles that are used together more often; and ii) create a *multidimensional* tile map partitioning, since the density of intersections/streets by tile (as defined by this study) may vary roughly (since the partitioning size is fixed), we envision that having tiles of different size may improve the overall performance, i.e. having tiles that are more tight to the map density.

ACKNOWLEDGMENTS

The Mexican author wishes to express his gratitude to "CONACyT, Consejo Nacional de Ciencia y Tecnología de México" for its economical support in the program "Estancias Posdoctorales Internacionales 2015-2016" (project number 263564), and to Cesar Bonavides-Martinez, from the Center for Genomic Sciences, UNAM, for his technical support.

This research was partially funded by the University of Málaga, Andalucía Tech, the Spanish Ministry of Economy and Competitiveness, and FEDER (grants TIN2014-57341-R and 8.06/5.47.4142).

References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[3] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, pp. 345–, Jun. 1962. [Online]. Available: http://doi.acm.org/10.1145/367766.368168

[4] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis, "Fast shortest path distance estimation in large networks," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 867–876.

[5] T. Breugem, T. Dollevoet, and W. van den Heuvel, "Analysis of fptases for the multi-objective shortest path problem," *Computers & Operations Research*, vol. 78, pp. 44–58, 2017.

[6] A. Sedeno-Noda and A. Raith, "A dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem," *Computers & Operations Research*, vol. 57, pp. 83–94, 2015.

[7] Q. Li, B. Y. Chen, Y. Wang, and W. H. Lam, "A hybrid link-node approach for finding shortest paths in road networks with turn restrictions," *Transactions in GIS*, vol. 19, no. 6, pp. 915–929, 2015.

[8] S. Aridhi, P. Lacomme, L. Ren, and B. Vincent, "A mapreduce-based approach for shortest path problem in large-scale networks," *Engineering Applications of Artificial Intelligence*, vol. 41, pp. 151–165, 2015.

[9] A. Efentakis, D. Pfoser, and A. Voisard, "Efficient data management in support of shortest-path computation," in *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science*. ACM, 2011, pp. 28–33.

[10] J. Santos, "Real-world applications of shortest path algorithms," *The Shortest Path Problem: 9th DIMACS Implementation Challenge*, vol. 74, pp. 1–19, 2009.

[11] R. K. Ahuja, K. Mehlhorn, J. Orlin, and R. E. Tarjan, "Faster algorithms for the shortest path problem," *Journal of the ACM (JACM)*, vol. 37, no. 2, pp. 213–223, 1990.

[12] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[13] R. J. Gutman, "Reach-based routing: A new approach to shortest path algorithms optimized for road networks." *ALENEX/ANALC*, vol. 4, pp. 100–111, 2004.

[14] N. Jing, Y.-W. Huang, and E. A. Rundensteiner, "Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 3, pp. 409–432, 1998.

[15] H. N. Djidjev, "Efficient algorithms for shortest path queries in planar digraphs," in *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer, 1996, pp. 151–165.

[16] M. Miler, D. Medak, and D. Odobašić, "The shortest path algorithm performance comparison in graph and relational database on a transportation network," *Promet-Traffic&Transportation*, vol. 26, no. 1, pp. 75–82, 2014.

[17] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[18] J. Durillo, A. Nebro, and E. Alba, "The jmetal framework for multi-objective optimization: Design and architecture," in *CEC 2010*, July 2010, pp. 4138–4325.