Applied Soft Computing 24 (2014) 181–195

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Red Swarm: Reducing travel times in smart cities by using bio-inspired algorithms

Daniel H. Stolfi [a,*], Enrique Alba [a,b]

[a] Departamento de Lenguajes y Ciencias de la Computación, University of Malaga, Malaga, Spain
[b] VSB-Technical University of Ostrava, Czech Republic

ARTICLE INFO

Article history:
Received 18 November 2013
Received in revised form 16 May 2014
Accepted 8 July 2014
Available online 16 July 2014

Keywords:
Evolutionary algorithm
Road traffic
Smart city
Smart mobility
WiFi connections
Traffic light

ABSTRACT

This article presents an innovative approach to solve one of the most relevant problems related to smart mobility: the reduction of vehicles' travel time. Our original approach, called Red Swarm, suggests a potentially customized route to each vehicle by using several spots located at traffic lights in order to avoid traffic jams by using V2I communications. That is quite different from other existing proposals, as it deals with real maps and actual streets, as well as several road traffic distributions. We propose an evolutionary algorithm (later efficiently parallelized) to optimize our case studies which have been imported from OpenStreetMap into SUMO as they belong to a real city. We have also developed a Rerouting Algorithm which accesses the configuration of the Red Swarm and communicates the route chosen to vehicles, using the spots (via WiFi link). Moreover, we have developed three competing algorithms in order to compare their results to those of Red Swarm and have observed that Red Swarm not only achieved the best results, but also outperformed the experts' solutions in a total of 60 scenarios tested, with up to 19% shorter travel times.

Introduction

In a world where road traffic is continuously increasing faster than the infrastructure intended to contain it, traffic congestions and accidents are more frequent now than in the past, especially in urban areas. This situation affects different aspects of our society such as health, environmental pollution, and economic development, just to name a few.

Nowadays, we have to address many challenges and do so from a global city perspective [1], this includes surveillance and control, keeping efficiency in terms of performance, cost, maintenance, and support, especially if the opportunities to increase the number and capacity of roads are limited.

Smart City represents a world initiative which is defined by six characteristics [2]: smart economy, smart people, smart governance, smart mobility, smart environment, and smart living. They are related to factors such as innovative spirit, entrepreneurship, level of qualification, creativity, public and social services, pollution control, cultural facilities, health conditions, etc.

On the one hand, *smart mobility* is related to transport and Information and Communication Technologies (ICT). Therefore, it is focused on local, national and international accessibility, the availability of ICT-infrastructure, and transport systems which are sustainable, innovative, and safe. A well accepted belief is that new transport strategies will improve urban traffic as well as citizens' mobility and quality of life.

On the other hand, *smart environment* is related to an intelligent management of natural resources and is focused on pollution reduction, environmental protection, and management of sustainable resources. Apart from the pollutant gas emissions from factories, power plants and waste incinerators, one of the main sources of greenhouse gas emissions are vehicles. Besides reducing travel times, our work is also related to reducing emissions from vehicles' exhaust pipes as well as their fuel consumption.

This is the motivation for this work. Here, we propose a data information system spread throughout the city at a low cost and able to redirect vehicles in movement in the city to finally achieve shorter travel times and fewer traffic jams in urban areas. The WiFi connectivity of drivers' terminals (e.g., smartphones) plus an easy interaction with Red Swarm spots located at traffic lights will define our base scenario for a global intelligent mobile service with benefits for drivers and for general municipal policies. Note that in most cities there already are computers running Windows/Linux, which will support our proposal based on V2I (Vehicle to Infrastructure) communications.

* Corresponding author. Tel.: +34 952133303.
E-mail addresses: dhstolfi@lcc.uma.es (D.H. Stolfi), eat@lcc.uma.es (E. Alba).

http://dx.doi.org/10.1016/j.asoc.2014.07.014
1568-4946/

The deployment of Red Swarm, a real time system for suggesting distributed personalized routes in a modern city, not only provides customized routes to every single vehicle, but it is also able to collect information from the road traffic (anonymously) enabling local authorities to better know the on-line and historical data of the city.

This paper is organized as follows: the next section reviews several publications related to our proposal. The subsequent sections present the Red Swarm architecture, three competing algorithms that we have designed in order to compare their results to those of Red Swarm, and the parameterization of the algorithms and the experimental settings. Section "experimental analysis" focuses on the studies done and a discussion of the results. And finally, in last section, conclusions and future work are presented.

Related work

Out of all the vast amount of literature on this topic, we move far beyond classic ideas like modeling the city with differential equations, considering it as a complex network, or using graphs and traditional operation research techniques. We do so because there are too many assumptions there, too much modeling of what is happening in an actual city. We try to address a complete system with all its components in an environment, as realistic as possible, in a computer. We also focus on strategies that interact with drivers, which is seldom the case in the current literature. Having said that, we are going to revisit approaches that try to address the global city problem on the one hand and interact in some way with drivers (who actually make the decisions in traffic), using some kind of connectivity.

With the intention of improving road traffic, there exist solutions which focus on adjusting the traffic light cycles. In [3] a Particle Swarm Optimization algorithm is presented to do this in an area of Malaga and Seville (Spain), while in [4] the authors use a Genetic Algorithm with a Cellular Automaton based traffic simulator in order to optimize the traffic in the district called "La Almozara" in Saragossa (Spain). In spite of the promising results that they have obtained, these proposals do not suggest alternative routes to the drivers as we do, but they consider the city as a set of traffic lights regulating drivers as a whole.

Moreover, the authors in [5] describe the development and implementation of an Ant Colony Optimization algorithm to solve the traffic signal coordination problem. Experiments show that the ACO algorithm yielded better results when it was compared against a genetic algorithm. Unfortunately, the work mentioned is not based on rerouting vehicles and the traffic networks studied are small (up to 20 traffic lights).

A strategy to reduce traffic congestions by using V2V (Vehicle to Vehicle) communication is presented in [6]. There, a series of beacons are used to analyze the traffic flow and communicate to drivers possible breakdowns, so that they can maintain a bigger gap between themselves and the car in front while they are traveling along the highway. Although the technical requirements are minimal, it is not suitable for an urban area comprising many intersections, traffic lights, and roundabouts.

In [7], unexpected traffic jams provoked by accidents are prevented by using four different strategies which consist of different road bans, which define the kind of control performed. The proposed control strategies could be communicated to drivers by changing electronic signals from a traffic signal controller when a traffic jam is going to occur. The strategies are simulated in two-way rectangular grid networks although neither of them corresponds to real streets of a city, although the authors were interested in using a microsimulator in future work, but we actually use it here as part of our work.

A distributed and cooperative system dedicated to road self-organization is presented in [8] with the aim of detecting traffic jams and transmitting traffic alerts. The authors present a theoretical model based on the FORESEE cooperation model [9] composed of a set of agents which are physically installed in each vehicle. Each agent evaluates the traffic conditions and exchanges information with other agents over wireless media. The working scenario is quite big and the results are achieved by using a traffic simulator. However, as their conclusions depict, a minimum number of vehicles is necessary in order to sense and communicate the traffic state, while in our proposal, communications only depend on a fixed number of spots.

In [10], the authors present an optimization method that determines routes for drivers and then increases the performance of the traffic network via dynamic traffic routing. A novel algorithm, called Ant Colony Routing (ACR), based on Ant Colony Optimization (ACO) with stench pheromone and colored ants, is proposed for the optimization. The different vehicles routes are modeled by using the colored ants so that they are only sensitive to their own color. Moreover, the stench pheromone is used to disperse ants throughout the network thereby preventing traffic jams. This proposal differs from ours not only in the fact we use an evolutionary algorithm but also in the type of scenarios we work with, which consist of real streets (instead of a graph) and we test them by simulating the dynamics of the whole city and players (vehicles, driving rules, red lights, etc.).

An Event-Driven Architecture (EDA) is proposed in [11] that detects different levels of traffic jams taking into account environmental information, as well. The proposal has been designed to run either as part of the on-board equipment in each vehicle or in a mobility management center outside the road. Simulation tests on the VGSim tool show the suitability of the proposal which can detect a wide range of traffic jams with regard to their length and severity. This work principally focuses on the study of motorways, while ours is on urban traffic and the detection of traffic jams, which we try to prevent.

In [12] a proposal based on an integrated macroscopic traffic model (S model) which includes a macroscopic urban traffic flow model and a microscopic traffic emission model (VT-Micro) is presented. While the former provides macroscopic traffic states for each link, the latter evaluates the emissions based not only on the speed of all vehicles but also on the acceleration or the deceleration of each of them. Moreover, a Model Predictive Control (MPC) is applied to urban traffic networks with the aim of reducing both travel delays and gas emissions based on the aforementioned models, by regulating the stop-and-go behavior and distributions of traffic flows within the network with the aid of traffic signals. Although we also reduce travel times, we use a different approach consisting in rerouting vehicles to avoid congested streets in real geographical areas, in real time, and in a customized manner for every driver.

A multi-agent framework for Swarm Intelligence (SI) is presented in [13]. It can be used to solve search problems on a computer network. The authors present initial experimental results that show their approach scales better when implementing the Ant Colony System (ACS) algorithm, but is faster when implementing the Bee Colony Optimization (BCO) algorithm.

The Red Swarm

In order to guide and optimize the traffic in the whole city we propose a new system called Red Swarm. By giving personalized advice to every single driver of a vehicle circulating through an urban area we hope to reduce time, fuel consumption, and gas emissions. We propose a continuous distributed exchange of data between vehicles and spots that will allow us to run intelligent

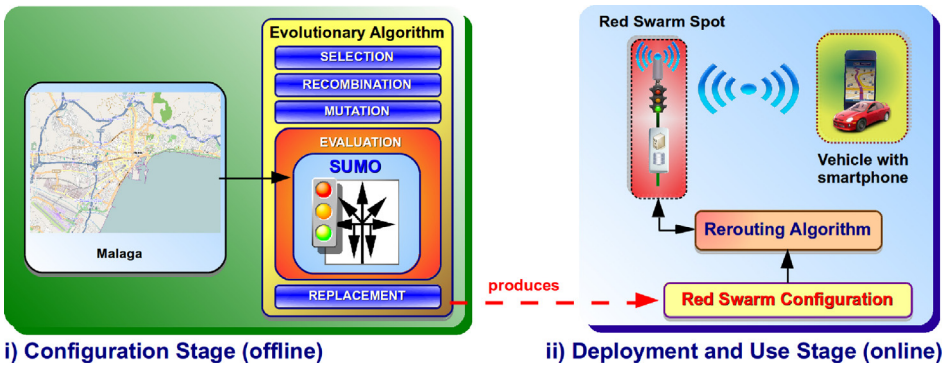


Fig. 1. Red Swarm architecture.

algorithms that compute optimized destinations both at the individual and the ensemble levels.

We have taken some small starting steps in [14,15] which have led us to the proposal in this article, showing the whole system, optimized and compared with expert and alternative techniques in the following sections.

Our Red Swarm architecture consists of:

1. Several spots distributed throughout the city, installed at traffic lights, which use a WiFi connection to suggest new routes to vehicles.
2. The *Rerouting Algorithm* (RA), which selects the route to be suggested based on the configuration of the system and the vehicles' destination.
3. The evolutionary algorithm (EA), which computes the configuration of the system.
4. User Terminal Units (UTU), usually mobile phones or tablets which are able to communicate with the spots, send their data, and receive the new routes suggested. Note that this function could also be fulfilled by On Board Units (OBU) installed in vehicles.

The Red Swarm architecture is divided into two stages: (i) the configuration stage, and (ii) the deployment and use stage (Fig. 1).

In the configuration stage the EA calculates the configuration for the spots by using the traffic simulator SUMO (Simulation of Urban MObility) [16] in order to evaluate each solution.

In the deployment and use stage, the aforementioned optimal configuration for the Red Swarm spots is used by the RA (explained later) to suggest new routes to the vehicles that are approaching a junction controlled by a Red Swarm spot by using a WiFi link. Although the configuration is not recalculated in the deployment and use stage, the routes suggested to each vehicle are personalized, so as to split traffic into separate routes that will benefit both the individual vehicles and the overall traffic flow. Regarding the WiFi communication, we have experimented with this technology in [17] and the results observed, support a wide coverage area of 77 m on average which supports the study in this article.

Traffic simulators implement different flow models [18] to specify rules for car following, lane changing, etc. According to the level of granularity, they can be categorized as macroscopic, mesoscopic, and microscopic simulators.

Microscopic models describe the mobility parameters of each vehicle with respect to others in detail, while macroscopic and mesoscopic models work at a higher level of abstraction [19].

As we need a realistic car following model we choose the SUMO microscopic traffic simulator which implements the car following model developed in [20]. While SUMO simulates the traffic flow of each vehicle, TraCI (Traffic Control Interface) [21] makes it possible to control SUMO externally and obtain the state of the simulation as well as change it. When the simulation ends, data from the itinerary of each vehicle such as departure times, travel times, gas emissions, etc., can be obtained by parsing several XML files.

Case study

For this work, we have applied the Red Swarm solution with the aim of reducing the average travel times of vehicles in an area of the city of Malaga which is well-known for suffering from traffic jams at peak times. The area is delimited to the north by Carretería Street, to the south by the Mediterranean Sea, to the east by Gutemberg Street, and to the west by the Guadalmedina River, and encompasses an area of about 2.5 km^2.

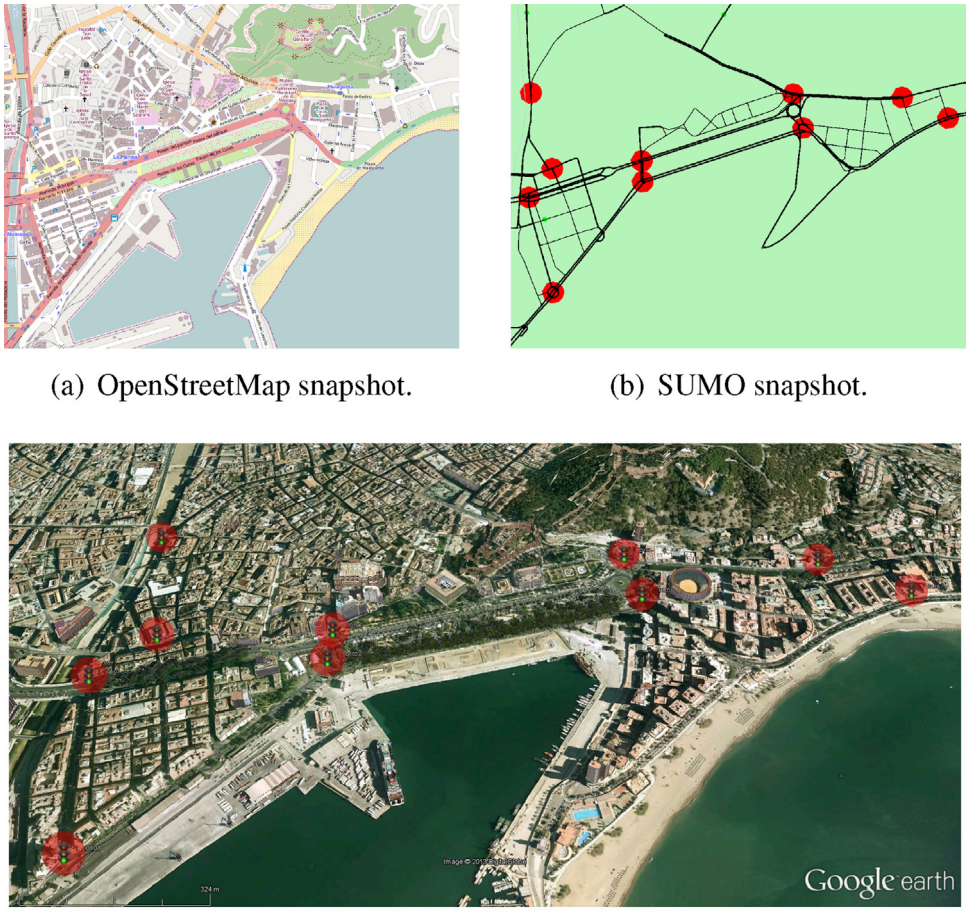
We show in Fig. 2(a) a snapshot of the area analyzed taken from OpenStreetMap, in Fig. 2(b) we can see the same area imported into SUMO; and in Fig. 2(c) a snapshot exported from SUMO to Google Earth™ is presented. In the latter we can see our ten Red Swarm spots placed at strategic junctions of the city represented by red circles. Since most modern cities already have a computer inside traffic lights, the spots just need a WiFi antenna plus our software solution.

The case study was built following the steps presented in Fig. 3 based on the model building principles in SUMO [22]. First, the selected area was downloaded from OpenStreetMap. Second, we added the existing traffic lights and the Red Swarm spots by using the application JOSM (Java OpenStreetMap). We also removed all extra data included in the original map such as POIs, pedestrian crossings, etc. to improve the load time and avoid the possibility that SUMO could misuse them. Third, we translated the map into the SUMO format by using the NETCONVERT utility which is part of the SUMO suite. Finally, we added traffic flows between streets of origin and destination which were calculated by using the DUAROUTER utility, also included in the SUMO suite.

The experts' solution consists of a real scenario of this case study, where vehicles take different routes between their origin and destination (flows). These flows are composed of several routes which have been calculated by using all the weight attributes available in DUAROUTER: travel time, noise, CO, CO_2, PM*x*, HC, NO*x*, and fuel. Although some of these routes are identical (were included once), by using this method we will prevent vehicles in the same flow from taking the same route, which could favor the creation of traffic jams.

In this paper we have worked with two different case studies called *z8* and *z12*. These two share the same characteristics (listed in Table 1) such as 262 traffic lights, ten Red Swarm spots, four vehicle types (listed in Table 2), and nine input and output streets where vehicles arrive at and exit from the analyzed area, respectively.

However, the number of vehicles (800 in *z8* vs. 1200 in *z12*) and the period of time analyzed (2400 in *z8* vs. 3000 in *z12*) are different. The former is to test different traffic density in the same area and the latter is a consequence of the number of vehicles: the



(a) OpenStreetMap snapshot. (b) SUMO snapshot.

(c) Red Swarm spots exported to Google Earth™.

Fig. 2. Area of the city of Malaga analyzed, imported from OpenStreetMap into SUMO, and then exported to Google Earth™. (For interpretation of the references to color in text, the reader is referred to the web version of this article.)

more vehicles that enter the studied geographical zone, the more time they take to abandon the scenario. Further on in the article, we analyze other techniques for rerouting vehicles as well as 30 different scenarios for the two case studies.

Since the arrival rate, vehicle type, arrival speed, etc., are non-deterministic, by changing values in the city we are able to define different scenarios in order to evaluate different traffic distributions in the same case study.

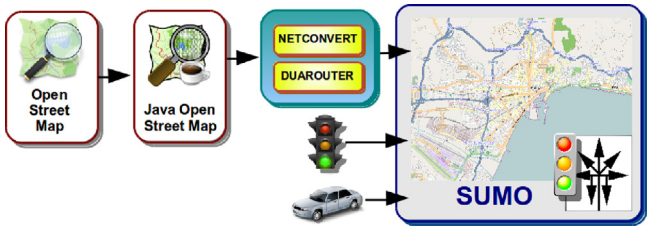


Fig. 3. Scenario building scheme.

Representation

In this section we discuss how we encode the solutions of the problem (routes that maximally split traffic flow in the city with reduced times for travelers) in the evolutionary algorithm used. Each Red Swarm spot is placed at a traffic light situated at a junction of the city. As a consequence, it can be thought of a set of input and output streets (Fig. 4). The input streets are the streets by which the

Table 1
Characteristics of the two case studies.

Case study	*z8*	*z12*
# vehicles	800	1200
Analysis time (s)	2400	3000
# Traffic lights	262	
# Red swarm spots	10	
# Vehicle types	4	
# Input streets	9	
# Output streets	9	

Table 2
Type and characteristics of vehicles.

Type	Arriving (prob.)	MaxSpd. (Km/h)	Accel. (m/s^2)	Decel. (m/s^2)	Length (m)
Sedan	0.50	160	0.9	5.0	3.8
Van	0.25	100	0.8	4.5	4.2
Wagon	0.15	50	0.7	4.0	4.3
Transport	0.10	40	0.6	3.5	4.5

Fig. 4. Representation of a Red swarm spot.

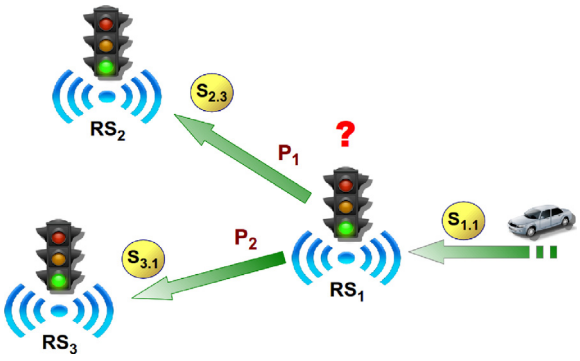


Fig. 5. Rerouting example.

vehicles arrive at the junction and the output streets are the streets by which the vehicles leave that junction.

When a vehicle is approaching the junction by an input street, the WiFi link is established which triggers the rerouting process. Then, this process suggests a new route to the vehicle depending on the configuration of the Red Swarm (probabilities P_1, P_2 and P_3 in Fig. 4) and the vehicle's final destination in the city (the neighborhood or street, which would depend on the final implementation in actual cities).

In Fig. 5 an example of a possible rerouting is illustrated. When the vehicle is detected from the input street $S_{1.1}$ belonging to the Red Swarm spot RS_1, a new route is suggested. The next step in the route of the vehicle might be the input street $S_{2.3}$ (spot RS_2) or the input street $S_{3.1}$ (spot RS_3), depending on the values of the probabilities P_1 and P_2 which have been previously optimized for the traffic in this area in connection to the rest of the areas by the EA.

This behavior is repeated in each Red Swarm spot until the vehicle arrives at its final destination. In this way, the new route of the vehicle is made up of several paths between input streets of Red Swarm spots, from the first spot which has detected and rerouted it, to the last spot which is placed in proximity to the vehicle's final destination (Fig. 6).

As there are ten Red Swarm spots in our case studies each with several input streets, there are a total of 28 input streets in the area under analysis. Furthermore, there are also nine possible destinations, so that the different routes from one spot's input street to its reachable ones are arranged in nine chunks, thus each destination has its own configuration inside Red Swarm. Therefore, each reachable input street has a probability value associated with it which defines its chances of being suggested to a vehicle as the next step in its personalized route to destination.

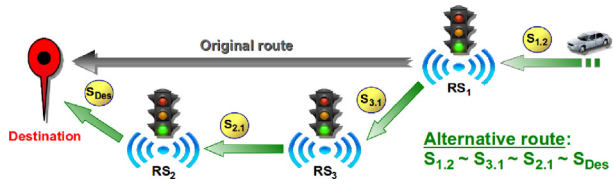


Fig. 6. Rerouting of a vehicle through Red Swarm spots toward its final destination.

Fig. 7 illustrates the schematic representation of the problem as well as the probabilities for each reachable street of being selected, mapped into a tentative solution vector. Note that the solution vector is made of 1098 floating-point numbers (the probability values) which reveals the complexity of this problem.

The probabilities included in each destination chunk are normalized by following Eq. (1), so that each value is in the range of [0, 1]. So, the sum of all the probabilities for the K_N reachable streets in the destination chunk M, which is part of the configuration of the street N, is equal to 1.

$$P_{S_N D_M} = P_{(N,M)_1} + \cdots + P_{(N,M)_{K_N}} = \sum_{i=1}^{K_N} P_{(N,M)_i} = 1,\ P_{(N,M)_i} \in [0,1] \quad (1)$$

Evaluation function

The evaluation of scenarios takes into account the average travel time of vehicles as well as the number of vehicles inside the area analyzed, both of which are taken after studying several scenarios in the city.

In Eq. (2) we propose an evaluation function which computes a real number indicating the fitness of the configuration being evaluated. As we want to minimize this value, the lower it is the better.

$$F = \omega_1(N - n) + \omega_2 \frac{1}{n} \sum_{i=1}^{n} (delay + duration)_i \quad (2)$$

In the first term, we denote with N the total number of vehicles and with n the number of vehicles that have arrived at their destination when the analyzed period of time ends. We have used this term to preserve the completeness of the study as we wanted all vehicles to complete their itinerary.

In the second term, the travel time of a vehicle is calculated by adding the time that it has waited before entering the area due to a congested input street (*delay*) and the time that it has spent in arriving at its destination (*duration*). Note that only the vehicles which arrive at their destination (n) are included in the summation.

Both terms are weighted by two constants (ω_1 and ω_2). The value of the first one depends on the degree of penalization wanted. We assume that the vehicles which are in the city at the end of the analysis would have spent (on average) half of the analysis time in completing their itinerary ($\omega_1 = \frac{1}{2} analysistime$). The second term is weighted by one ($\omega_2 = 1$), so that both terms are in the same scale of time (seconds).

Rerouting Algorithm (RA)

The Rerouting Algorithm (RA) is part of our Red Swarm architecture. By using our RA, each Red Swarm spot is able to suggest a new route to vehicles which are approaching it. The pseudocode

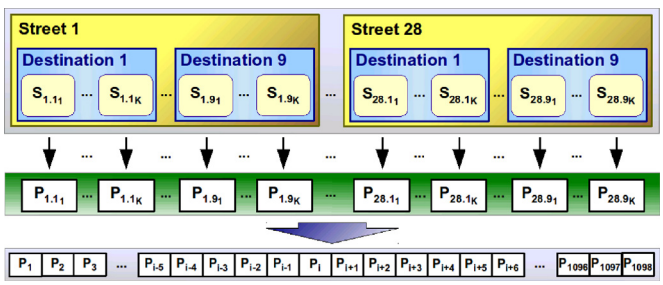


Fig. 7. Schematic representation of the configuration of the Red Swarm mapped into a probability solution vector of floats.

of the RA is presented in Algorithm 1 and the flow chart in Fig. 8 describes this.

Algorithm 1. Rerouting Algorithm

```
procedure REROUTING(vehicle)
  current ← getStreet(vehicle)
  if isDestination(current, vehicle) then
    nextDest ← current
  else
    nextDest ← getDestination(current)
    if nextDest = [] then
      nextStreets ← getReachableStreets(current)
      nextDest ← getStByProbability(nextStreets)
    end if
  end if
  setNextDestinationStreet(nextDest, vehicle)
end procedure
```

First, the current street is obtained from the data sent by the vehicle and the final destination is checked so that a vehicle in the last street of its itinerary will not be rerouted at all. To the contrary, if the vehicle has not yet reached its destination, all the routes from the current street to the destination of the vehicle are obtained from the configuration previously calculated by the EA. If there are no direct routes to the final destination (it is not directly reachable from the current street), the algorithm will obtain all the input streets which are directly reachable from the current street so that the vehicle can be sent to another Red Swarm spot.

Then, one of the routes from the current street to another one is chosen based on the probabilities stored in the configuration of the spot.

Finally, the next destination street (thus the route) of the vehicle is set to the chosen one (here we have supposed that all drivers accept the change suggested) and the process ends.

Note that if there is more than one route to the destination suggested, one of these routes will be randomly selected (we presume it to be the driver's decision).

Evolutionary algorithm (EA)

In this work, we use an EA in the offline stage of the Red Swarm architecture. Its purpose is to find the optimal arrangement of routes in the city to then spread the traffic out in a way that is efficient for both drivers and municipal policies.

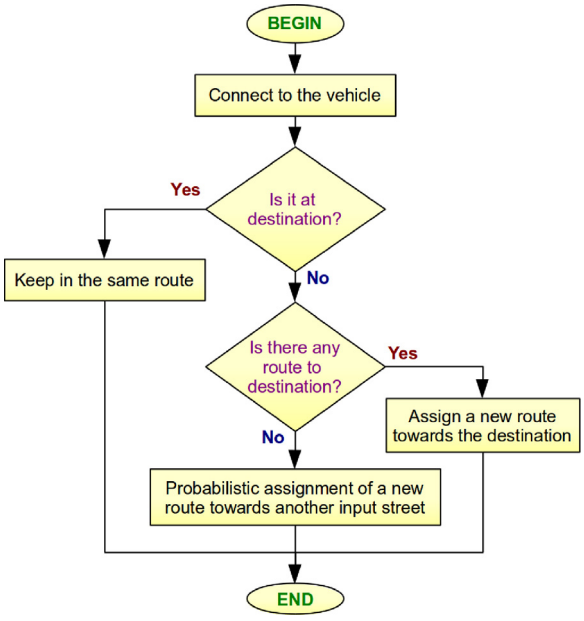


Fig. 8. Flow chart describing the Rerouting Algorithm.

Evolutionary algorithms are inspired by the evolution of individuals which are well adapted to their environment. They are a population-based method and at each iteration several operators are applied to the μ individuals of the population. After each iteration, the λ new individuals are obtained in order to be incorporated in the next generation. Recombination, mutation, selection, and replacement operators are commonly found in EAs as a way of producing new individuals which can experiment self-adaptation and be naturally selected based on their fitness value which is provided by the objective function. GAs are a very popular subclass of EAs with proven efficacy in solving combinatorial optimization problems [23], either static or dynamic versions [24].

We have designed our evolutionary algorithm (EA) which is a steady state (10 + 2)-EA. We have chosen to work with a small population only creating two new individuals in each generation in order for it to be more efficient, because our fitness function takes, on average, about 20 s to be computed, because of the complexity involved in analyzing the traffic distribution in the city.

Selection operator

The selection operator implemented chooses two individuals for reproduction by using a uniform probability distribution.

Recombination operator

We have designed and tested two different recombination operators: The Street Two Point Crossover (STPX) and the Destination Crossover (DESX).

On the one hand, STPX consists of a standard two point crossover in which two individuals are crossed by swapping their contents between two randomly selected points, to produce two descendants. In our case we exchange all the probabilities in range of the input street blocks selected (the complete blocks including the destination chunks of probabilities). Fig. 9(a) shows an example of STPX where the probability values of the street blocks from six to twelve are exchanged.

On the other hand, DESX exchanges the configuration of destination chunks throughout all the street blocks of the parents instead of exchanging the configuration of streets, as STPX does. Fig. 9(b) shows an example of DESX where the probability values of destination chunks three and four are exchanged in all the street blocks of the solution vector.

The former keeps the configuration of each input street intact, while the latter does the same with each destination chunk.

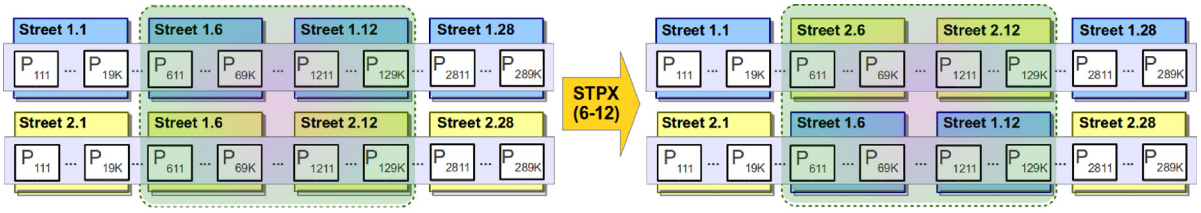
Mutation operator

In [14] we proposed and tested several mutation operators. There, we decided to use two different operators for the mutation of the individuals in the EA.

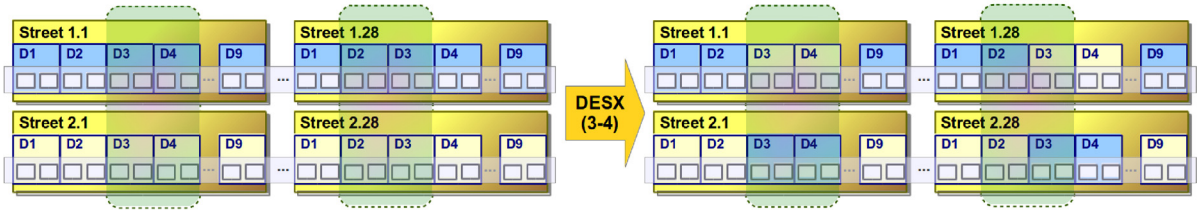
First, the ADOS operator, which assigns new values to the probabilities of routes, in all the destination chunks of one street block, was used as the mutation operator until the fitness value of the best individual of the population was lower than a defined threshold (θ).

Then, the ODOS operator was used in the next iterations of the algorithm. This operator assigns new values to the probabilities of routes in just one destination chunk in one street block. The former is meant to explore the search space and the latter, to exploit the accumulated search experience.

Here we have decided to unify these two operators in just one, called the Variable Mutation Operator (VMO). In order to preserve the variability presented by the two former mutation operators in VMO, we have added two parameters to the new operator: π_1 and π_2. When the fitness value of the best individual of the population is greater than θ, π_1 will be the probability of changing the values in a destination chunk corresponding to a single street block. Otherwise, π_2 will be the probability used. We have also kept θ as a parameter of the operator, thus, if $\pi_1 = 1$, the behavior of this



(a) Street Two Point Crossover (STPX).



(b) Destination Crossover (DESX).

Fig. 9. Recombination operators.

operator will be the same as the ADOS operator, while if $\pi_2 = \frac{1}{9}$, it will resemble the ODOS (one chunk of nine).

Note that we are changing only the probabilities in destination chunks of just one street block whether it is π_1 or π_2 used for the computations.

In Algorithm 2 the pseudocode of the VMO is presented. First, the input street s is randomly chosen from the individual ones. Then, the list of destinations is obtained from the individual and some of the probabilities in each destination chunk are changed depending on the value returned by the *random* function and the value of π_k (which is equal to π_1 or π_2).

Finally, when all the destinations in the street block have been processed, the individual mutated is returned.

Algorithm 2. Variable Mutation Operator (VMO)

```
procedure VMO(individual)
  s ← getRndStreet(individual)
  destinations ← getDestinations(individual)
  for all d ∈ destinations do
    if random() < π_k then                ▷ k ∈ {1, 2}
      AssignNewProbabilities(d)
    end if
  end for
  return individual
end procedure
```

For example, Fig. 10 represents the VMO applied to an individual when *Street 4* has been randomly selected as the target street of the mutation. We can also see that the destination chunks *Destination 1*, *Destination 4*, and *Destination 5* have also been randomly selected to be mutated. As a consequence, in this example VMO changes the probability values of the ranges $P_{4.1.1}$ to $P_{4.1.K}$, $P_{4.4.1}$ to $P_{4.4.K}$, and $P_{4.5.1}$ to $P_{4.5.K}$ corresponding to the K input streets which are reachable from *Street 4*, when the final destination of the vehicle is either *Destination 1*, *Destination 4* or *Destination 5*.

In this article, due to the characteristics of the problem, we have worked with two fixed rates for assigning (and mutating) the probabilities in a destination chunk: 0.5 (EA05) and 1.0 (EA10). By using the former, up to two reachable input streets (if there are two or more) have an equal probability of being suggested ($P_1 = 0.5$, $P_2 = 0.5$), while by using the latter only one can be selected ($P = 1$).

Replacement operator

We have proposed two different replacement strategies: (i) Elitist Replacement (ER) in which the λ worse individuals of the population are replaced only if they have a fitness value worse than an offspring individual; and (ii) Random Replacement (RR) in which the offspring always replaces λ individuals of the current population which are randomly selected, except for the best individual in the population which is always preserved (elitism).

In Section "Parameterization of the EA" the evaluation of these operators as well as the rest of the parameterization of our EA is presented.

Parallel EA (pEA)

In order to achieve a more general configuration for the Red Swarm spots, we have addressed the optimization of several scenarios in the same run of the EA by doing multiple evaluations (one per scenario) of the same individual in order to calculate its fitness. As this implies an increment in the run time, we have developed a new parallel EA (pEA) to tackle it. Fig. 11 shows the block diagram of our pEA, which calculates the fitness function of an individual over n scenarios as the average fitness of them all.

Competitor techniques for our EA

In order to evaluate our Red Swarm architecture and our EA, we propose here three other competing algorithms instead of our EA that could reduce travel times by configuring the system in the configuration stage: (i) The DJK algorithm, based on the Dijkstra shortest path algorithm [25]; (ii) the DV algorithm, based on the Bellman-Ford algorithm [26]; and (iii) the ACO algorithm, which is an adaptation of the Ant System algorithm presented in [27]. All of them have been implemented in Python as well as in the EA and the RA.

Fig. 10. Variable mutation operator (VMO).

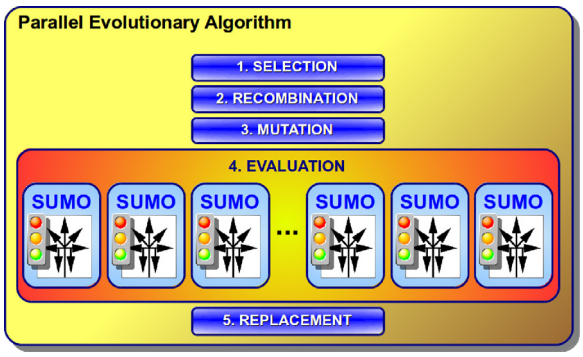


Fig. 11. Parallel evolutionary algorithm (pEA).

Fig. 12 shows the block diagram of the configuration of a rerouting when DJK, DV and ACO algorithms are used.

Dijkstra (DJK)

The Dijkstra algorithm (DJK) proposed here is based on the implementation of the well-known Dijkstra shortest path algorithm [25] included in the DUAROUTER utility, which is part of the SUMO suite. The DJK algorithm will be used as an alternative method of calculating the solution vector to be used by the RA which runs in each Red Swarm spot.

This algorithm conceives the scenario as a graph in which inputs, the spots' input streets, and destinations are all the nodes, while the edges are the different paths between them. The weight values for each edge are calculated by counting the number of preplanned routes that include each street of that edge.

Then, the solution vector is obtained based on these weights by converting them to probability values so that they can be managed by the RA.

To do that, the streets involved in the routes between Red Swarm spots (calculated by DUAROUTER) and the vehicles' flows are counted. Then the weights are assigned to the edges of the graph based on the number of routes in which the streets in those edges are included. Finally, the weights are converted into probabilities by calculating the inverse value of the weight of each edge. In Eq. (3) the probability for the route between the input street S_N and the destination D_M is calculated. Therefore, $\omega_{S_N D_M}$ is the weight of the edge between S_N and D_M, and $\sum_i \frac{1}{\omega_{S_N D_i}}$ is the summation of all the destinations reachable from S_N. As we want to distribute the road traffic through different streets, the more routes use a street, the less likelihood of being chosen the street has.

$$P_{S_N D_M} = \frac{1}{\omega_{S_N D_M} \sum_i 1/(\omega_{S_N D_i})} \tag{3}$$

Distance Vector (DV)

The Distance Vector (DV) algorithm that we propose here as a basic competitor to our EA, is based on the implementation included in the RIP Version 2 protocol (RFC 2453) of the Distance Vector algorithm, which in turn is based on the Bellman-Ford algorithm [26].

This algorithm and its implementation are found in most of the current Internet routers and its main characteristic is that each node of the network knows the length of the shortest path from itself to all the other destination routers.

In this case, instead of calculating routes between source and destination networks, we calculate routes between the input streets of the Red Swarm spots of the city and the final destinations of vehicles.

The routing tables of each input street contain an entry for each destination in the city, which are updated with the ID of the next input street in the route. This input street is selected depending on the number of spots which will be in this journey to the vehicle destination when it takes this route, the fewer the better.

In this case, instead of the RA we run a simplified version of the Rerouting Algorithm (RA') so that the next street suggested to vehicles is taken directly from the routing tables calculated by the DV algorithm, thus we are not calculating a solution vector of probabilities but actual rerouting tables.

When the rerouting takes place in each spot, the next street suggested to vehicles will be obtained directly from the routing tables calculated for each input street, instead of using the RA based on probabilities.

Ant Colony Optimization (ACO)

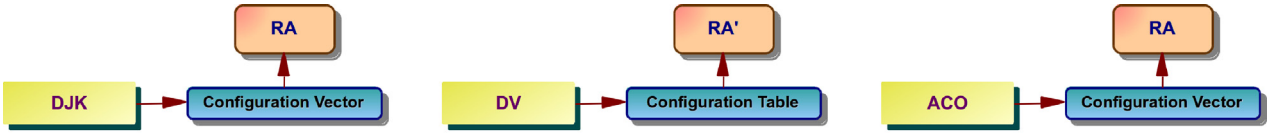
Ant Colony Optimization (ACO) [28] is an optimization technique inspired by the natural behavior of ants. It is a general-purpose heuristic method for identifying efficient paths through a graph and has been successfully applied to solve different combinatorial optimization problems with discrete representations.

Inspired by the Ant System model [27], we here provide a new algorithm to suggest routes in our Red Swarm system in a new way.

Our case study is represented by nine graphs (one per destination), which nodes are the 28 input streets and which edges are the routes between them. In ACO, a set of ants construct a solution by traveling through a graph which represents the environment. So, we have created 56 ants per graph (twice the number of input streets) and placed them randomly in the nodes which are different from the destination ones.

Then, each ant makes a sequence of probabilistic decisions when arriving at each Red Swarm spot in order to choose the next input street to visit. This series of decisions represents the path that the ant has followed to reach its target destination. Thus, we build the ants' tours across the nodes of the graph (input streets) until the destination is reached.

When all the ants have ended their individual journey, a new iteration begins after marking each edge of the graph with artificial pheromones. These pheromones influence the ants' decisions in following iterations in such a way as to increase the likelihood of the paths which have been transited most, being chosen in the current iteration. There also exists an evaporation coefficient which prevents the pheromone values from having an influence for too long on one iteration propagation, as happens in the natural ant system.



(a) Dijkstra (DJK). (b) Distance Vector (DV). (c) Ant Colony Optimization (ACO).

Fig. 12. Configuration and rerouting when using DJK, DV and ACO algorithms.

The pheromone trail update is done as explained in Eq. (4), where $\tau_{ij}(t+1)$ is the intensity of the trail in the next iteration, $\tau_{ij}(t)$ is the current intensity, $(1-\rho)$ represents the evaporation of the trail in the current iteration, and $\Delta\tau_{ij}$ is the sum of the quantity per unit of length of pheromones laid on the edge (i, j) by the k-th ant (Eq. (5)). Finally, the $\Delta\tau_{ij}^{k}$ value is calculated by following Eq. (6), where Q is the relative importance of the trail and L_k is the tour length of the k-th ant.

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij} \tag{4}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m}\Delta\tau_{ij}^{k} \tag{5}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & \text{if the } k\text{-th ant travels from street } i \text{ to street } j \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Furthermore, the probability of transit from street i to street j is formalized in Eq. (7), where η_{ij} is the heuristic value which defines the visibility of the next street, and $allowed_k$ is the set of the direct reachable streets. We have used the same heuristic as in DJK and DV, i.e., the number of routes that contain the streets, as we have discussed in Section "Dijkstra". Moreover, both α and β are the parameters that control the relative importance of trail versus visibility in the transition equation.

$$p_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}]^{\beta}}{\sum_{k\in allowed_k}[\tau_{ik}(t)]^{\alpha}\cdot[\eta_{ik}]^{\beta}} & \text{if } j \in \text{allowed}_k \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

As we need probability values to configure the system, when all the ants have ended their tours we evaluate the solution by mapping the paths (tours) into probabilities in order to get the status vector to configure the Red Swarm spots. Since we have a set of routes from the ACO system, at this point we count the number of paths transited by the ants, for each route, from the nine graphs that include this route. Then, we calculate the normalized solution vector of probabilities as we did in DJK. Thus, the more transited a route is, the less likely it is to be selected.

Initial parameterization and experimental settings

Now, we describe the experiments that have been conducted in order to parameterize the ACO algorithm (Section "Parameterization of the ACO algorithm") and the EA algorithm (Section "Parameterization of the EA"). As we are dealing with a very complex problem and consequently with long execution times, we cannot follow an exhaustive method of parameterization. Instead, we have tested several parameter values for the operators and made decisions based on the data collected.

In all the experiments we have applied a Friedman test to determine the best parameter and operator, and then we have studied the statistical significance of these data by using the Wilcoxon test to compare the best ranked distribution to the rest (pairwise comparisons). All the experiments were conducted by performing 30 independent runs on the same scenario.

In Section "Experimental settings", the experimental settings are discussed.

Parameterization of the ACO algorithm

Based on the values proposed in [27], we have conducted several experiments in order to perform the parameterization that better suits our problem. Table 3 shows the results of 30 runs of the algorithm using different values for the quantity of trail laid by the ants (Q), the relative importance of the trail (α) and the visibility (β). The number of ants (m) was set to 504 (twice the number of streets multiplied by the nine destinations, thus, the nine graphs), the trail evaporation in each iteration $(1-\rho)$ was set to 0.25, and the initial value for trails ($\tau_{ij}(0)$) was set to 0.50.

Table 3
Parameter tuning of the ACO algorithm. Note that the best values are in bold.

Parameters			Best Fitness		Friedman rank	Wilcoxon p-value
Q	α	β	Avg.	StdDev.		
	0.5	1.0	5215.6	141.6%	3.73	0.19
25	1.0	0.5	7613.4	216.1%	**2.00**	–
	1.0	**1.0**	**4317.3**	183.7%	2.33	0.99
	0.5	1.0	7552.4	**121.1%**	4.97	0.05
50	1.0	0.5	8840.0	225.8%	2.70	0.07
	1.0	1.0	13853.8	147.7%	5.27	0.00

We have explored a large set of potential sets of parameters for ACO (six in total) adding up to a final large number of 180 experiments. As we can see, the best ranked algorithm, $ACO_{25,1.0,0.5}$ ($Q=25, \alpha=1.0, \beta=0.5$) and the second best, $ACO_{25,1.0,1.0}$ ($Q=25$, $\alpha=1.0$, $\beta=1.0$), both share the same statistical benefits ($p-$value = 0.99), so we have chosen the latter because its average fitness and standard deviation are lower than the former(4317.3 vs. 7613.4 and 183.7% vs. 216.1%, respectively).

Parameterization of the EA

First of all, we have performed the tuning of the recombination operator. We have tested the STPX and DESX operators for recombination probabilities (P_c) of 0.2, 0.4, 0.6, 0.8, and 1.0, by doing 30 runs for each operator and probability values (300 runs). Table 4 shows the results of the experiments conducted as well as the statistical analysis performed. As we can see, STPX has outperformed DESX for all the recombination probability values tested.

Then, by using STPX as recombination operator ($P_c=0.6$), we have performed the tuning of the mutation and replacement operators. Again we have tested probability values between 0.2 and 1.0 by doing another 300 runs (30 by case), but in this occasion we have varied the mutation probability (P_m) as well as the two different replacement strategies: Elitist Replacement (ER) and Random Replacement (RR).

Table 4 denotes that ER behaves much better than RR, especially for mutation probabilities of 1.0 and 0.8, where the best Friedman ranks are achieved (1.43 and 2.10, respectively). Note that these mutation probabilities are the probability of mutating an individual in each iteration of the algorithm while the amount of mutation on the individual itself is governed by the π_1 and π_2 values.

Finally, in order to parametrize the mutation operator VMO, we have conducted several experiments which are also listed in Table 4. We have tested π_1 and π_2 for probability combinations of $\frac{1}{9}$ (one over the number of destinations), $\frac{1}{3}$, $\frac{1}{6}$ and 1.

We have chosen $\pi_1=0.33$ and $\pi_2=0.11$ based on their average fitness value in spite of the fact that it is the second best ranked case (5.30 vs. 5.27) Moreover, the Wilcoxon p-value denotes that this configuration (0.33,0.11) and the best ranked one (0.66,0.33) are not so different (p-value = 0.91), which allows us to make this decision.

All in all, the mutation probability of a route will depend on P_m, the number of input streets, π_1, and π_2, so that it will be $\frac{1}{1}\frac{1}{28}\frac{1}{3}=\frac{1}{84}$ for π_1 and $\frac{1}{1}\frac{1}{28}\frac{1}{9}=\frac{1}{252}$ for π_2.

The rest of the parameters of the EA are: $P_C=0.6$, $P_M=1.0$, $\theta=1500$, and a maximum of 5000 generations.

Table 4
Tuning of recombination, mutation and replacement operators as well as the Variable Mutation Operator (VMO). Note that the best values of each operator are in bold.

Recombination operators					Mutation and replacement operators ($P_c = 0.6$)				Variable Mutation Operator (ER, $P_c = 0.6$, $P_m = 1.0$)				
P_c	Fitness		Friedman Rank	Wilcoxon p-value	P_m	Fitness		Friedman Rank	Wilcoxon p-value	π_1, π_2	Best Fitness		Friedman Rank	Wilcoxon p-value
	Average	StdDev				Average	StdDev				Average	StdDev		
Street Two Point Crossover (STPX)					Elitist Replacement (ER)									
0.2	917.3	8.3%	4.07	0.27	0.2	2879.6	352.8%	7.30	0.00	0.11, 0.33	35957.2	352.6%	5.57	0.47
0.4	3930.3	398.0%	3.53	0.00	0.4	3332.8	396.9%	5.13	0.00	0.11, 0.66	12707.1	484.9%	6.77	0.34
0.6	**887.5**	**5.7%**	2.37	0.50	0.6	12865.3	388.2%	4.17	0.00	0.11, 1.00	9827.1	275.1%	8.90	0.05
0.8	941.8	12.0%	4.80	0.00	0.8	842.6	**5.8%**	2.10	0.00	**0.33, 0.11**	**987.0**	**12.4%**	5.30	0.91
1.0	1068.5	84.7%	**2.17**	–	**1.0**	**835.4**	51.8%	**1.43**	–	0.33, 0.66	34553.9	371.2%	6.67	0.25
Destination Crossover (DESX)					Random Replacement (RR)					0.33, 1.00	2527.9	315.5%	7.40	0.23
										0.66, 0.11	10245.8	354.6%	5.37	0.99
0.2	964.0	14.6%	6.00	0.00	0.2	13805.0	227.5%	9.30	0.00	0.66, 0.33	4664.1	318.0%	**5.27**	–
0.4	1301.2	151.3%	5.17	0.00	0.4	3432.8	311.8%	7.25	0.00	0.66, 1.00	4999.7	427.9%	7.83	0.15
0.6	14711.5	213.1%	8.00	0.00	0.6	996.0	9.1%	6.47	0.00	1.00, 0.11	13455.8	397.6%	5.33	0.50
0.8	20301.9	179.1%	8.93	0.00	0.8	4093.1	100.0%	6.43	0.00	1.00, 0.33	10833.4	230.4%	6.67	0.27
1.0	36507.2	140.0%	9.97	0.00	1.0	965.1	9.8%	5.42	0.00	1.00, 0.66	11510.1	321.3%	6.93	0.08

Table 5
Average fitness, average number of iterations and statistical tests for the optimization of one scenario of the case studies *z8* and *z12*. The best values of each case study are in bold.

Alg.	Fitness		# iterations		Friedman Rank	Wilcoxon p-value
	Avg.	StdDev	Avg.	StdDev		
z8						
EA10	**658.0**	**7.1%**	2531.6	36.4%	**1.00**	–
EA05	760.0	8.1%	3485.0	32.5%	2.67	0.00
ACO	1610.4	172.7%	**573.7**	**12.2%**	2.33	0.00
z12						
EA10	**855.7**	**8.5%**	3699.0	30.5%	**1.00**	–
EA05	1037.8	9.7%	3411.7	30.5%	2.67	0.00
ACO	7613.4	216.1%	**579.8**	**15.3%**	2.33	0.00

Experimental settings

The experiments conducted were executed in the cluster belonging to the NEO (Networking and Emerging Optimization) group, which consists of 16 nodes (64 cores) equipped with an Intel Core2 Quad CPU (Q9400) @ 2.66 GHz and 4 GB of RAM, 14 nodes (28 cores) equipped with an Intel Pentium D @ 2.8 GHz and 1 GB of RAM, one node (eight cores) equipped with two Intel Xeon (E5405) @ 2.00 GHz and 8 GB of RAM, and one node (eight cores) equipped with an Intel Core i7 (920) @ 2.67 GHz and 4 GB of RAM.

The time-dependent experiments such as the ones targeted to measure speedup of the parallel algorithms have been conducted in the same kind of nodes in order to fairly evaluate each execution, while the parallel optimization of eight scenarios have been conducted in the eight core machines. Moreover, we have used HTCondor Version 7.8.4 as the software responsible for the management of these heterogeneous resources.

We have carried out 1140 independent runs for the parameterization of the algorithms and another 180 independent runs for the optimization of the case studies performed by the EA100, EA50 and ACO algorithms.

We have also performed 240 independent runs of the parallel optimization of 2, 4 and 8 scenarios, and finally in the calculation of the speedup we have carried out 100 runs (ten per algorithm).

All in all, the total time spent by the algorithms in the cluster was about 34 days.

Experimental analysis

In the following sections we address the optimization of the case studies *z8* and *z12*. First, we have run our algorithms to compare their performance as well as the solutions achieved. Second, we have tested the best solution from each algorithm in 30 different scenarios in order to discover how scalable the solutions are. Finally, we have identified the best performing algorithms and reported the improvement achieved in the average travel time of vehicles.

Case study z8

In this section we have performed the optimization of *z8* in order to achieve an optimum for the configuration of the Red Swarm spots. This case study consists of 800 vehicles that arrive in the analyzed zone of the city via nine input streets and that drive through the city until reaching their destination.

First, as DJK and DV are deterministic, we only need one execution to get the configuration vector from those algorithms. In the case of the EA10, EA05 and ACO algorithms, we have carried out 30 runs as they are nondeterministic in order to analyze their performance as well as achieve the best configuration for the Red Swarm spots. In Table 5 we present the results of the optimization of one

Table 6
Fitness comparative and statistical test of 30 scenarios of *z8* and *z12*. Note that the best values of each case study are in bold.

Alg.	Fitness		Best in 30	Friedman rank	Wilcoxon *p*-value
	Average	StdDev			
z8					
EXP	**623.5**	**4.3%**	–	**1.50**	–
DJK	222588.6	8.3%	0.0%	6.00	0.00
DV	50229.1	15.2%	0.0%	5.00	0.00
EA10	658.1	34.9%	53.3%	1.53	0.75
EA05	761.8	23.6%	0.0%	3.90	0.00
ACO	837.6	71.3%	3.3%	3.07	0.00
z12					
EXP	821.1	3.5%	–	1.83	0.00
DJK	1048164.2	12.8%	0.0%	6.00	0.00
DV	146557.1	60.6%	0.0%	4.97	0.00
EA10	**788.0**	**3.3%**	**83.3%**	**1.17**	–
EA05	14494.9	510.0%	0.0%	3.83	0.00
ACO	923.2	6.8%	0.0%	3.20	0.00

scenario of *z8* performed by our algorithms. As we can see, it consists of the average of the best fitness achieved in the 30 runs as well as the standard deviation. Furthermore, the average number of iterations and the standard deviation are also provided together with the Friedman Rank and the Wilcoxon *p*-value.

The first conclusion is that EA10 has achieved the lowest average fitness value (658.0) and standard deviation (7.1%). It is also notable the small number of iterations (573.7) performed by the ACO algorithm (thus a faster convergence) despite its high average fitness (1610.4). EA10 also presents the best Friedman rank (1.00) in *z8* and a comparison with EA05 and ACO by using the Wilcoxon test confirms that the differences are statistically significant. Based on the fitness values and the statistical analysis we have selected EA10 as the best of the optimization algorithms in the scenario of *z8*.

In Fig. 13(a) the distribution of the fitness values from the 30 runs performed by the algorithms is shown in a box plot where the better solutions (lower fitness value) achieved by EA10 are clearly depicted. Furthermore, the best fitness value of the algorithms analyzed plus the experts' solution in the optimization scenario are shown in the bar graph of Fig. 13(b) where we can see that DJK and DV present the worst results of the comparison.

Second, we have configured the Red Swarm spots with the solution obtained from the five algorithms in order to analyze how they behave in the 30 different scenarios of *z8* as well as to compare them with the experts' solution (EXP). Table 6 and Fig. 13(c) and (d) present the results where the high fitness values of DJK and DV confirm they are not capable of rerouting vehicles to their destinations, while avoiding traffic jams in the period of time analyzed.

Although EA10, EA05 and ACO are capable of rerouting all vehicles to their destinations, neither of them is able to achieve an average fitness (and travel time) lower than the experts' solution (623.5).

We have chosen EA10 as the best solution of the algorithms although its average fitness in the 30 scenarios is higher than EXP. The Friedman rank confirms that EA10 is the closest algorithm to EXP but it cannot outperform EXP in *z8* (travel times in EA10 are about 25 s longer in average). That is mainly explained by the fact that EA10 is better than EXP in 53.3% of those scenarios, as can be seen in the comparison of the fitness value in 30 different scenarios depicted in Fig. 13(e).

Therefore, in order to analyze how Red Swarm behaves when compared with the experts' solution (EXP) in *z8* when it is configured by EA10, we have illustrated the travel time vs. the number of vehicles in the city in Fig. 13(f). Although we have optimized a case study of 800 vehicles, we have tested the configuration obtained by EA10 with up to 2400 vehicles. As can be seen, EA10 becomes effective when there are more than 560 vehicles in *z8* and it is always better than EXP beyond 880 vehicles. This is an interesting and high impact conclusion, since it means that, as long as we have around 1000 vehicles in a city our solution is more efficient for drivers and modern urban policies. Furthermore, considering that any modern city reaches well over 1000 vehicles, we can claim that our system is both cheap and effective for any normal city; in fact, the larger the city, the better.

Moreover, in Fig. 13(g) and (h) we present the traffic density and the travel times of vehicles in *z8*. We can see in both graphs the effects of the best solution of each algorithm on the road traffic, which confirm that neither DJK nor DV are capable of managing such a number of vehicles and that EA10 routes vehicles out of the city faster than the experts' solution.

Finally, the average travel times in the 30 scenarios tested as well as the average route length of vehicles are listed in Table 7. Note that the average distance traveled by vehicles when they are being rerouted by Red Swarm is always longer than in the experts' solution. This was to be expected because we are rerouting vehicles via alternative streets which are not part of the shortest path, with the aim of reducing traffic jams. This extra length is however minimal (10.8% on average) and has a huge advantage in reducing times for drivers and the city.

Case study z12

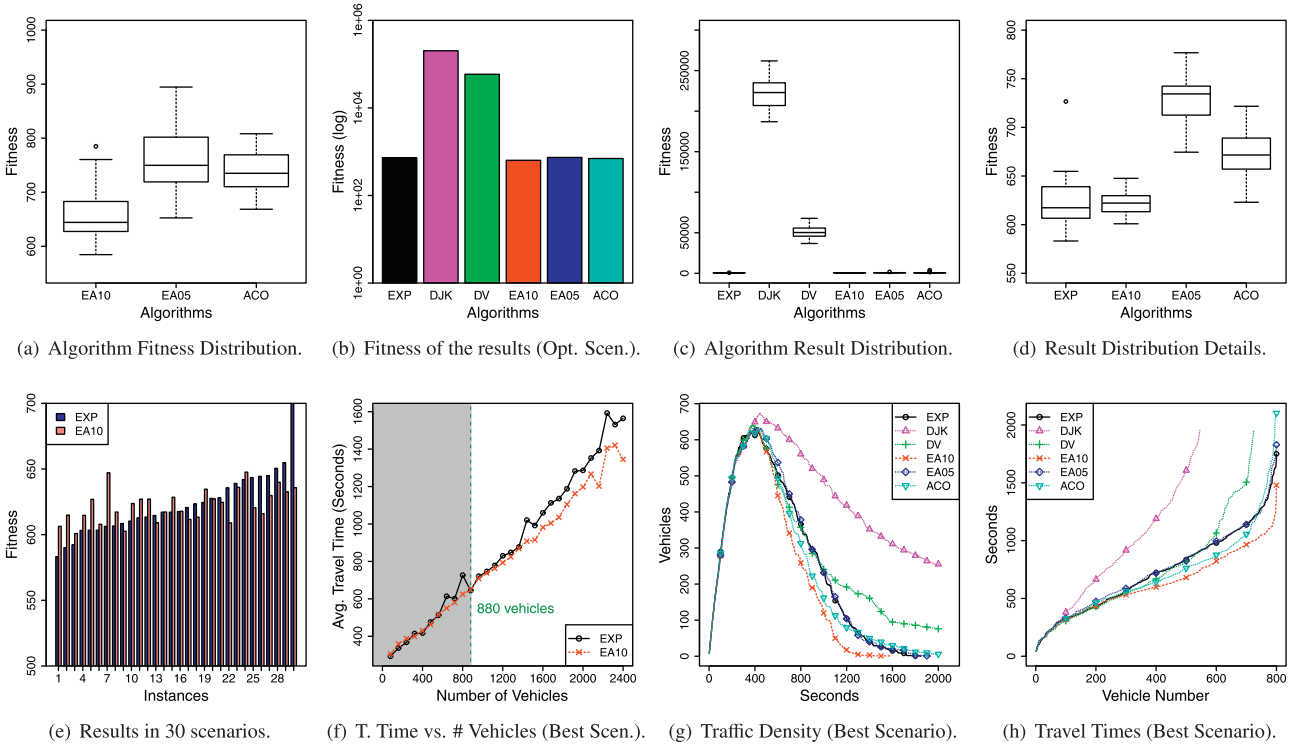
As the next step, we have addressed the optimization of *z12* which consists of 1200 vehicles which arrive in the first 100 s of the analyzed period of time, thus it is a harder case study.

First, we have optimized one scenario by using EA10, EA05, and ACO. The results from the 30 independent runs of the optimization algorithms are presented in Table 5. We can observe in the table that EA10 has a lower average fitness value (855.7) than EA50 (1037.8) and ACO (7613.4) (the lower the better). EA10 is also the best Friedman ranked algorithm, while the Wilcoxon *p*-value indicates significant differences from the others, which made us select EA10 again as the best performing of all our algorithms in *z12*. In Fig. 14(a) the distribution of the fitness values from the 30 independent runs is represented as a box plot. Note that ACO presents quite an asymmetric distribution which has penalized its average fitness value.

Second, the best solution achieved by each algorithm in the previous step (training), was used to configure the Red Swarm spots in 30 different scenarios of *z12* in order to evaluate how robust each solution is (testing). Fig. 14(b–d) and Table 6 show the results of these 30 runs. EA10 achieves better results than EXP in 25 of 30 scenarios (83.3%, see Fig. 14(e)) while the rest of algorithms are unable to do that in any of the scenarios of *z12*. The average fitness values also indicate that the EA10's solution (788.0) is better than EA05 (14494.9), ACO (923.2), and also EXP (821.1).

The study on how EA10 scales in *z12* is depicted in Fig. 14(f). There we can see that the solution achieved by EA10 in *z12* is effective when there are more than 960 vehicles in the area analyzed, which is lower than the total number of vehicles in that case study. Furthermore, in Fig. 14(g) and (h) we can observe the traffic density and the travel times of vehicles in *z12*. It is notable how DJK and DV have real problems in route vehicles to their destinations, as their configurations do not depend on the number of vehicles in the case study as the rest of algorithms do.

Table 7 indicates the average travel time and the average route length of vehicles in the 30 different scenarios tested as well as the best scenario (the scenario in which each algorithm achieves its best results). By using the solution computed by EA10 to configure the Red Swarm spots in *z12*, we achieve an improvement of 4.0% in the average travel time of vehicles and a maximum



(a) Algorithm Fitness Distribution. (b) Fitness of the results (Opt. Scen.). (c) Algorithm Result Distribution. (d) Result Distribution Details.

(e) Results in 30 scenarios. (f) T. Time vs. # Vehicles (Best Scen.). (g) Traffic Density (Best Scenario). (h) Travel Times (Best Scenario).

Fig. 13. Figures for case study *z8*, where we can observe the fitness distribution of the algorithms and the best values achieved in the optimizing scenario (a and b, respectively); the fitness distribution of the solution achieved, applied to 30 different scenarios (c and d); the comparison between the fitnesses of EA10 and the experts' solution (e); and the graphs which depict how vehicles behave in *z8* when are routed by using Red Swarm (f–h).

Table 7
Results of the optimization of the vehicles' average travel time, divided in the average of 30 scenarios and the best of them. Note that the best values of each case study are in bold.

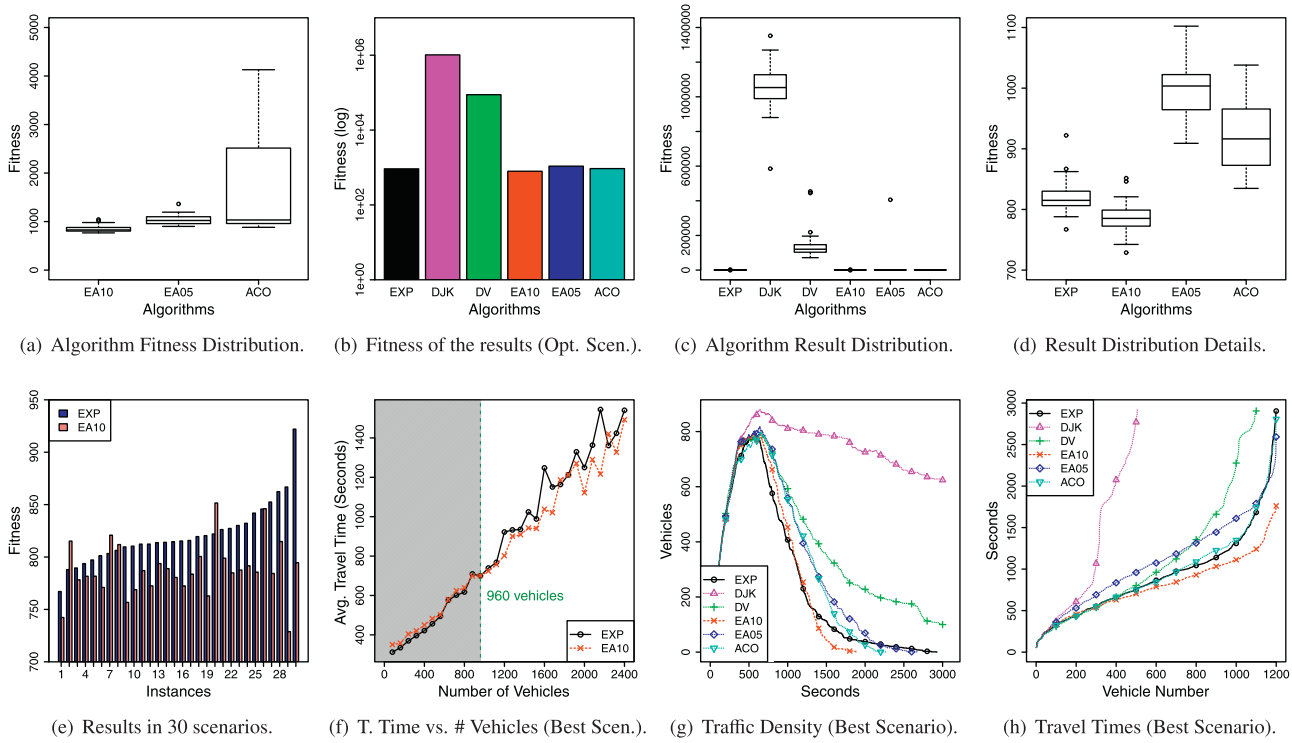
	Alg.	Average 30 scenarios						Best scenario					
		Travel time			Route length			Travel time			Route length		
		Avg.	StdDev	Improv.	Avg.	StdDev	Rate	Avg.	StdDev	Improv.	Avg.	StdDev	Rate
z8	EXP	623.5	4.3%	–	**1770.2**	**1.2%**	–	726.6	48.2%	–	**1771.3**	**42.0%**	–
	EA10	658.1	34.9%	−5.5%	1960.6	1.6%	10.8%	624.8	42.0%	14.0%	1945.9	45.9%	9.9%
	EA05	761.8	23.6%	−22.2%	2076.9	1.4%	17.3%	739.5	74.5%	−1.8%	2097.9	49.1%	18.4%
	ACO	837.6	71.3%	−34.3%	2066.4	1.8%	16.7%	699.0	51.5%	3.8%	2066.7	48.4%	16.7%
	pEA10.2	595.7	2.1%	4.5%	1895.8	1.4%	7.1%	599.0	41.1%	17.6%	1933.6	43.0%	9.2%
	pEA10.4	599.5	2.4%	3.9%	1948.3	1.4%	10.1%	598.2	40.8%	17.7%	1972.5	44.2%	11.4%
	pEA10.8	**584.3**	**1.8%**	**6.3%**	1901.4	1.6%	7.4%	**587.1**	**40.7%**	**19.2%**	1899.8	43.3%	7.3%
z12	EXP	821.1	3.5%	–	**1746.4**	1.2%	–	922.0	55.0%	–	**1716.5**	**43.5%**	–
	EA10	788.0	3.3%	4.0%	2029.0	3.5%	16.2%	794.6	42.4%	13.8%	2017.3	47.6%	17.5%
	EA05	14494.9	510.0%	−1665.3%	2183.6	2.3%	25.0%	1085.7	46.7%	−17.7%	2180.4	50.2%	27.0%
	ACO	923.2	6.8%	−12.4%	1927.0	1.9%	10.3%	935.1	55.1%	−1.4%	1919.1	50.8%	11.8%
	pEA10.2	779.3	2.1%	5.1%	2047.7	1.8%	17.3%	770.2	**41.8%**	16.5%	1903.9	45.1%	10.9%
	pEA10.4	768.5	2.0%	6.4%	1973.5	**1.0%**	13.0%	776.4	44.5%	15.8%	1978.4	45.7%	15.3%
	pEA10.8	**744.8**	**1.7%**	**9.3%**	1921.1	1.2%	10.0%	**749.0**	43.4%	**18.8%**	1931.7	45.3%	12.5%

improvement of 13.8% in the best scenario. These results are close to the experts' solution (travel times 33 s shorter on average) so that, we have concentrated on improving the solution achieved by EA10 even more in both case studies by optimizing more than just one scenario at the same time, as we describe in the next section.

Parallel EA (pEA)

Parallelism is central to carry out our experiments. We need to simulate the whole city, with thousands of cars following driving directions, interacting with each other, communicating, considering statistics and a large amount of data. Parallelism has been essential in reducing the study time from several years of computation to five weeks. As we wanted to optimize more than one scenario in the same run, we used a parallel version of our EA. Thus we have decided to calculate the average fitness value of two, four and even eight scenarios of *z8* and *z12* as the fitness value of an individual by using our pEA10.2, pEA10.4, and pEA10.8, respectively. So, we have named as pEA10.*x*, the parallel version of the EA10 algorithm which evaluates *x* scenarios in parallel and obtains the individual's fitness value by averaging them.

We have tested our parallel algorithms by optimizing one scenario of *z8* and one of *z12* in 30 independent runs. As can be clearly deduced from the values in Table 8, the more scenarios are optimized, the better the solution achieved. Consequently, pEA10.8



(a) Algorithm Fitness Distribution. (b) Fitness of the results (Opt. Scen.). (c) Algorithm Result Distribution. (d) Result Distribution Details.

(e) Results in 30 scenarios. (f) T. Time vs. # Vehicles (Best Scen.). (g) Traffic Density (Best Scenario). (h) Travel Times (Best Scenario).

Fig. 14. Figures for case study *z12*, where we can observe the fitness distribution of the algorithms and the best values achieved in the optimizing scenario (a and b, respectively); the fitness distribution of the solution achieved, applied to 30 different scenarios (c and d); the comparison between the fitnesses of EA10 and the experts' solution (e); and the graphs which depict how vehicles behave in *z12* when are routed by using Red Swarm (f–h).

presents the best results in the optimization scenarios of *z8* and also in the *z12* ones, as depicted in Fig. 15(a and b). Moreover, in Fig. 15(c) we provide a bar graph showing the fitness value comparative between the algorithms analyzed in the optimization scenario.

Next, we tested our solutions in 30 different scenarios for each case study. In Fig. 15(d) the distribution of the fitness values in those scenarios are depicted. The results show that all the parallel algorithms have achieved better fitness values than EXP in all the scenarios of *z12* and in mostly all of them in *z8* (Table 9 and Fig. 15(e)).

We can observe that the results seem to depend not only on the number of scenarios but also which scenarios are selected for optimization. Thus, the more scenarios that are optimized, the more robust the solution achieved.

Table 7 shows an average improvement of 9.3% and a maximum of 18.8% on the average travel time of the vehicles in *z12* when we configure the Red Swarm spots with the best solution obtained in this case from the pEA10.8 algorithm. As has happened in the rest of the experiments, routes are a bit longer than in the experts' solution (10% on average) while the average travel time has been shortened by a maximum of 173 s (76 s on average). Fig. 15(f) shows that the minimum number of vehicles from which Red Swarm is effective has been reduced to 560 vehicles as the behavior of the configuration computed by pEA10.8 is more linear with respect to the number of vehicles than the EA10 one. In addition, Fig. 15(g) and (h) shows the improvement in traffic density and travel times of the vehicles achieved by configuring the RA with the solution of the algorithms tested.

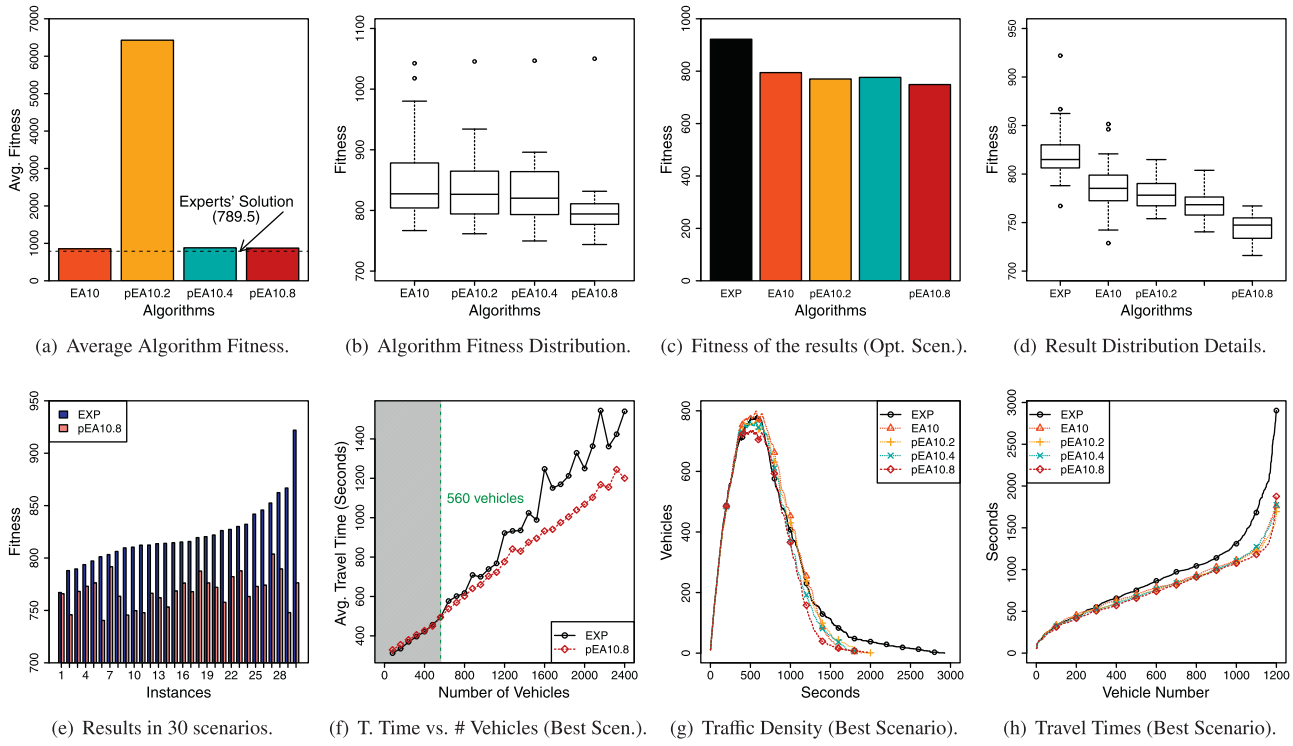
Finally, we have calculated the *weak orthodox speedup* [29] of our parallel algorithms by carrying out 10 runs of each algorithm with a different random seed. Because of the huge demand on resources that each parallel execution requires we have not done 30 runs in this case.

Table 10 lists the different experiments and results of the execution of the algorithm optimizing one scenario in a one core machine (sEA10.1), two scenarios in a one core machine (sEA10.2) and also in two core machines (pEA10.2), and so on. Note that we have named the sequential calculation of n fitness functions sEA10.n and the parallel calculation pEA10.n.

The results show that both pEA10.2 and pEA10.4 have nearly reached a linear speedup by using parallelism (1.9 and 3.9

Table 8
Average fitness, average number of iterations and statistical tests for the parallel optimization of two, four, and eight scenarios of the case studies *z8* and *z12*. Note that the best values of each case study are in bold.

Alg.	Fitness		# iterations		Friedman Rank	Wilcoxon p-value
	Avg.	StdDev	Avg.	StdDev		
z8						
pEA10.2	746.3	95.6%	**2647.5**	34.8%	2.73	0.00
pEA10.4	608.6	**2.8%**	2940.3	32.8%	2.07	0.00
pEA10.8	**603.3**	2.9%	3140.9	**31.9%**	**1.20**	–
z12						
pEA10.2	6428.3	332.2%	2866.2	38.0%	2.67	0.02
pEA10.4	881.4	**22.8%**	**2717.2**	35.4%	2.57	0.00
pEA10.8	**874.0**	41.8%	3496.5	**32.4%**	**1.43**	–



(a) Average Algorithm Fitness. (b) Algorithm Fitness Distribution. (c) Fitness of the results (Opt. Scen.). (d) Result Distribution Details.

(e) Results in 30 scenarios. (f) T. Time vs. # Vehicles (Best Scen.). (g) Traffic Density (Best Scenario). (h) Travel Times (Best Scenario).

Fig. 15. Figures for case study *z12*, where we can observe a graph bar showing the average fitness values of the parallel algorithms in the optimization scenario (a); the fitness distribution of the parallel algorithms and the best values achieved in the optimizing scenario (b and c, respectively); the fitness distribution of the solution achieved, applied to 30 different scenarios (d); the comparison between the fitnesses of pEA10.8 and the experts' solution (e); and the graphs which depict how vehicles behave in *z12* when are routed by using Red Swarm (f–h).

Table 9
Fitness comparative and statistical test of 30 scenarios of *z8* and *z12*. Note that the best values of each case study are in bold.

Alg.	Fitness		Best in 30	Friedman Rank	Wilcoxon *p*-value
	Average	StdDev			
z8					
EXP	623.5	4.3%	–	2.30	0.01
EA10	658.1	34.9%	53.3%	3.93	0.00
pEA10.2	595.7	2.1%	93.3%	3.33	0.00
pEA10.4	599.5	2.4%	90.0%	3.73	0.00
pEA10.8	**584.3**	**1.8%**	**100.0%**	**1.70**	–
z12					
EXP	821.1	3.5%	–	4.83	0.00
EA10	788.0	3.3%	83.3%	3.53	0.00
pEA10.2	779.3	2.1%	**100.0%**	3.13	0.00
pEA10.4	768.5	2.0%	**100.0%**	2.50	0.00
pEA10.8	**744.8**	**1.7%**	**100.0%**	**1.00**	–

Table 10
Average execution times and speedup of ten independent runs of the sequential algorithm (sEA10.*n*) and parallel (pEA10.*n*). The best speedup reached is in bold.

# Scenarios	Alg.	# Cores	Avg. time (s)	Speedup
1	sEA10.1	1	68617.4	1.0
2	sEA10.2	1	128722.4	1.0
	pEA10.2	2	67620.5	1.9
4	sEA10.4	1	144014.8	1.0
	pEA10.4	4	36881.6	3.9
8	sEA10.8	1	334714.7	1.0
	pEA10.8	8	49511.4	**6.8**

respectively). Moreover, when using 8 cores (pEA10.8) we got a very good speedup of 6.8 (an efficiency of 85%), not really the perfect 8 value because of the overload that the execution of eight parallel threads in the same computer represents.

Further interpretations of the solutions

The solutions achieved when we tested the configuration obtained by the EA in 30 different scenarios show that we have outperformed all of them when we use pEA10.8 as the optimization algorithm. This indicates that our solution is robust enough in such a complex problem like the one analyzed here where it is almost impossible to address a real traffic distribution.

We have also tested Red Swarm in different traffic conditions (number of vehicles) and we found a threshold representing the minimum number of vehicles from which Red Swarm becomes effective as a system for preventing traffic jams. In each case study, the threshold is quite a bit lower than the usual average number of vehicles in the real geographical area. However, with a small number of vehicles Red Swarm still works fine, but the enhancements to the city are less noticeable.

Finally, we observed lower traffic densities and shorter travel times when we applied Red Swarm to the most likely traffic situations in a modern city. This represents valuable aid to the citizens of a *Smart City* providing they become users of the Red Swarm.

Conclusions

In this article we have described our Red Swarm architecture and developed several algorithms in order to compare their solutions and performances to the Red Swarm ones.

Results show that Red Swarm configured by our EA always outperforms the other algorithms, and also outperforms the experts' solution when there are more than 560 vehicles in the scenario. This is welcome news: whenever we have more than 600 vehicles per 2.5 km^2 in a city our solution clearly beats the experts' one.

In our work we have assumed that all drivers have a terminal (e.g. smartphone) and they follow the routes suggested by the system. We expect a reduction in the average improvement on travel times if a significant percentage of drivers do not follow the indications of Red Swarm; the gradual penetration of the system will be the next step in our research. Moreover, unexpected changes in the state of the city, such as accidents or suddenly closed streets, have to be addressed in future work so as to be able to change the active configuration of the spots according to the new situation. A city is a very complex organism, and so we need to move step by step, gradually incorporating layers of human behavior and technologies.

Altogether, we think that our Red Swarm is a solution which could be used in current modern cities in order to reduce traffic jams and improve the citizens' quality of life. Moreover, it could be utilized to collect anonymous information from cars allowing local authorities to better understand the on-line and historical state of the city.

As a matter of future work, we are working on reducing emissions of greenhouse gases as we have evidence from previous work that this can be achieved by rerouting vehicles and avoiding traffic jams. We are also working on extending the geographical area to be analyzed to include the entire city of Malaga by adding more Red Swarm spots.

We will probably attempt another sophisticated problem definition, such as a multiobjetive modeling. Due to its computational complexity, we will need to carry out careful analysis in terms of metrics and Pareto fronts obtained.

Acknowledgments

The authors acknowledge funds from the Ministry of Economy and Competitiveness and FEDER under contract TIN2011-28194 (roadME http://roadme.lcc.uma.es). This research is also partially funded by project number 8.06/5.47.4142 in collaboration with the VSB-Technical University of Ostrava.

References

[1] C. Harrison, B. Eckman, R. Hamilton, P. Hartswick, J. Kalagnanam, J. Paraszczak, P. Williams, Foundations for smarter cities, IBM J. Res. Dev. 54 (2010) 1–16.
[2] R. Giffinger, N. Pichler-Milanović, Smart cities: Ranking of European Medium-sized Cities, Technical Report, 2007, October.
[3] J. García-Nieto, E. Alba, A.C. Olivera, Swarm intelligence for traffic light scheduling: application to real urban areas, Eng. Appl. Artif. Intell. 25 (2012) 274–283.
[4] J. Sánchez-Medina, M. Galán-Moreno, E. Rubio-Royo, Traffic signal optimization in "La Almozara" district in Saragossa under congestion conditions, using genetic algorithms, traffic microsimulation, and cluster computing, IEEE Trans. Intell. Transp. Syst. 11 (2010) 132–141.
[5] R. Putha, L. Quadrifoglio, E. Zechman, Comparing ant colony optimization and genetic algorithm approaches for solving traffic signal coordination under oversaturation conditions, Comput.-Aided Civil Infrastruct. Eng. 27 (2012) 14–28.
[6] F. Knorr, D. Baselt, M. Schreckenberg, M. Mauve, Reducing traffic jams via VANETs, IEEE Trans. Veh. Technol. 61 (2012) 3490–3498.
[7] J. Long, Z. Gao, P. Orenstein, H. Ren, Control strategies for dispersing incident-based traffic jams in two-way grid networks, IEEE Trans. Intell. Transp. Syst. 13 (2012) 469–481.
[8] P. Thomin, A. Gibaud, P. Koutcherawy, Deployment of a fully distributed system for improving urban traffic flows: a simulation-based performance analysis, Simul. Modell. Pract. Theory 31 (2013) 22–38.
[9] A. Gibaud, P. Thomin, Y. Sallez, Foresee, a fully distributed self-organized approach for improving traffic flows, Simul. Modell. Pract. Theory 19 (2011) 1096–1117.
[10] Z. Cong, B. De Schutter, R. Babuška, Ant colony routing algorithm for freeway networks, Transp. Res. C: Emerg. Technol. 37 (2013) 1–19.
[11] F. Terroso-Saenz, M. Valdes-Vela, C. Sotomayor-Martinez, R. Toledo-Moreo, A.F. Gómez-Skarmeta, A cooperative approach to traffic congestion detection with complex event processing and VANET, IEEE Trans. Intell. Transp. Syst. 13 (2012) 914–929.
[12] S. Lin, B. De Schutter, Y. Xi, H. Hellendoorn, Integrated urban traffic control for the reduction of travel delays and emissions, IEEE Trans. Intell. Transp. Syst. 14 (2013) 1609–1619.
[13] S. Ilie, C. Bădică, Multi-agent distributed framework for swarm intelligence, Proc. Comput. Sci. 18 (2013) 611–620.
[14] D.H. Stolfi, E. Alba, Red Swarm: smart mobility in cities with EAs, in: Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference, GECCO'13, ACM, 2013, pp. 1373–1380.
[15] D.H. Stolfi, E. Alba, Reducing gas emissions in smart cities by using the Red Swarm architecture, in: Advances in Artificial Intelligence, volume 8109 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2013, pp. 289–299.
[16] M. Behrisch, L. Bieker, J. Erdmann, D. Krajzewicz, SUMO – Simulation of Urban MObility: An Overview, in: SIMUL 2011: The Third International Conference on Advances in System Simulation, Barcelona, Spain, 2011, pp. 63–68.
[17] J. Toutouh, E. Alba, Performance analysis of optimized VANET protocols in real world tests, in: 7th International Wireless Communications and Mobile Computing Conference (IWCMC), 2011, IEEE, 2011, pp. 1244–1249.
[18] H. Hartenstein, K. Laberteaux, VANET: Vehicular Applications and Inter-networking Technologies, Wiley Online Library, Chichester, UK, 2009.
[19] M. Maciejewski, A comparison of microscopic traffic flow simulation systems for an urban area, Trans. Probl. 5 (2010).
[20] S. Krauß, Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics, 1998 (Ph.D. thesis).
[21] A. Wegener, M. Piórkowski, TraCI: an interface for coupling road traffic and network simulators, in: Proceedings of the 11th Communications and Networking Simulation Symposium, CNS'08, ACM, New York, NY, 2008, pp. 155–163.
[22] D. Krajzewicz, Traffic simulation with sumo - simulation of urban mobility, in: J. Barceló (Ed.), Fundamentals of Traffic Simulation, volume 145 of International Series in Operations Research & Management Science, Springer, New York, 2010, pp. 269–293.
[23] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, MI, 1975.
[24] E. Alba, A. Nakib, P. Siarry, Metaheuristics for Dynamic Optimization, Springer Berlin Heidelberg, 2012.
[25] E. Dijkstra, A Note on Two Problems in Connexion with Graphs, Numerische mathematik (1959) 269–271.
[26] D. Bertsekas, R. Gallager, Data Networks, vol. 2, 2nd ed., Prentice Hall, Englewood Cliffs, NJ, 1987.
[27] M. Dorigo, V. Maniezzo, A. Colorni, Ant System: optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybernet. B: Cybernet. 26 (1996) 29–41.
[28] M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, Cambridge, MA, 2004.
[29] E. Alba, Parallel Evolutionary algorithms can achieve super-linear performance, Inform. Process. Lett. 82 (2002) 7–13.