Applied Mathematical Modelling 49 (2017) 663–679

Contents lists available at ScienceDirect

Applied Mathematical Modelling

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

A new efficient approach for solving the capacitated Vehicle Routing Problem using the Gravitational Emulation Local Search Algorithm

Ali Asghar Rahmani Hosseinabadi[a,*], Najmeh Sadat Hosseini Rostami[b], Maryam Kardgar[a], Seyedsaeid Mirkamali[c], Ajith Abraham[d]

[a] Young Researchers and Elite Club, Ayatollah Amoli Branch, Islamic Azad University, Amol, Iran
[b] Department of Management, University of Guilan, Rasht, Iran
[c] Department of Computer Engineering and IT, Payame Noor University (PNU), Tehran, Iran
[d] Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation, and Research Excellence, Auburn, WA, USA

ARTICLE INFO

Article history:
Received 27 May 2015
Revised 28 January 2017
Accepted 22 February 2017
Available online 3 March 2017

Keywords:
Vehicle Routing Problem (VRP)
CVRP
Optimization
Gravitational Emulation Local Search Algorithm (GELS)

ABSTRACT

Capacitated Vehicle Routing Problem (CVRP) is one of the most famous specialized forms of the VRP, which has attracted considerable attention from scientists and researchers. Therefore, many accurate, heuristic, and meta-heuristic methods have been introduced to solve this problem in recent decades. In this paper, a new meta-heuristic optimization algorithm is introduced to solve the CVRP, which is based on the law of gravity and group interactions. The proposed algorithm uses two of the four basic parameters of velocity and gravitational force in physics based on the concepts of random search and searching agents, which are a collection of masses that interact with each other based on Newtonian gravity and the laws of motion. The introduced method was quantitatively compared with the State-of-the-Art algorithms in terms of execution time and number of optimal solutions achieved in four well-known benchmark problems. Our experiments illustrated that the proposed method could be a very efficient method in solving CVRP and the results are comparable with the results using state-of-the-art computational methods. Moreover, in some cases our method could produce solutions with less number of required vehicles compared to the Best Known Solution (BKS) in a very efficient manner, which is another advantage of the proposed algorithm.

1. Introduction

Distribution of goods is of great importance in logistics and supply chain management, and some problems in the domain of goods distribution may even seem to be modeled as Vehicle Routing Problems (VRP). The VRP plays an important role in reducing the costs of transportation in logistic distribution and it is considered as one of the most important combinatorial optimization problems. In this problem, a set of customers, each with one demand, are scattered in a graph. The vehicle that delivers service to the customers is programmed to achieve one or more goals when its route satisfies the needs of customers along the routes [1,2].

* Corresponding author.
E-mail addresses: A.R.Hosseinabadi@iaubeh.ac.ir, rahmani_aliasghar_1987@yahoo.com (A.A.R. Hosseinabadi), Najmeh_Hr@yahoo.com (N.S.H. Rostami), Kardgar_Maryam@yahoo.com (M. Kardgar), S.Mirkamali@pnu.ac.ir (S. Mirkamali), Ajith.Abraham@ieee.org (A. Abraham).

http://dx.doi.org/10.1016/j.apm.2017.02.042
0307-904X/

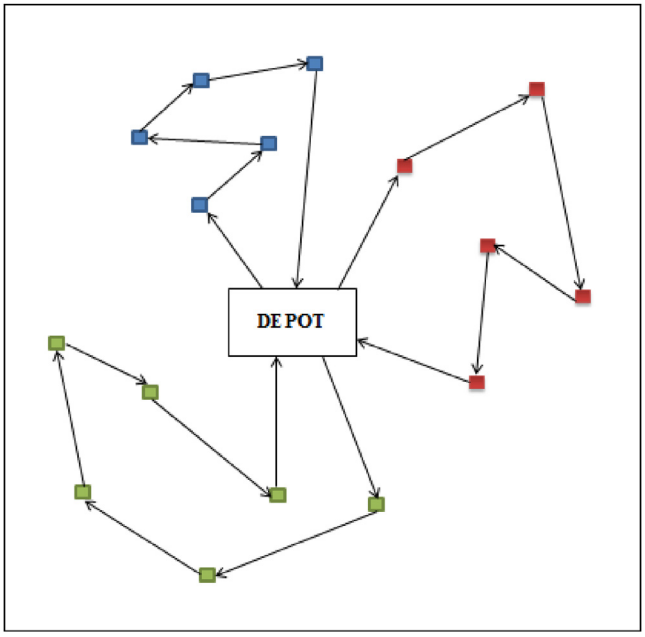

Fig. 1. Example of a group of CVRP routes.

The classical VRP includes determining minimum-cost paths for a set of homogeneous vehicles located in a depot to deliver services to customers that are geographically scattered. These services may be goods delivery, pick-up and delivery of students, or gather the packets to deliver incoming emails, etc. [3,4]. The VRP, which is an extension of the Traveling Salesman Problem (TSP), has many applications in real-world high transportation cost in commerce such as transferring waste materials, newspaper and food delivery, beverages distribution, etc.

CVRP is a specialized form of the VRP in which a fleet of vehicles located in a depot should meet the various demands of a group of customers for goods delivery. Fig. 1 depicts one group of CVRP routes.

The purpose in CVRP is to minimize the total cost (for example, a weighted function of the number of vehicles and the distance to be covered by them) of service delivery to a set of customers with specified demands. Routing must be designed such a way that each customer is met once and only by one vehicle.

CVRP is known as a combinatorial NP-hard problem because it includes a packing problem (in which a container has to be filled with a set of boxes) and the TSP [5,6].

As CVRP [7] can be defined as an undirected graph $G = (V, E)$, where $V = \{v_0, v_1, \ldots, v_n\}$ is a vertex set and $E = \{(v_i, v_j) | v_i, v_j \in V, i < j\}$ is an edge set. The depot is represented by vertex v_0, which uses m independent delivery vehicles, with identical delivery capacity Q, to service demands q_i from n cities or customers, $i = 1, 2, \ldots, n$, represented by the set of n vertices $\{v_1, \ldots, v_n\}$. A solution for the CVRP would be a partition $R_1, R_2, \ldots, R_m$ of V representing the routes of the vehicles.

The cost of the problem solution is the sum of the costs of its routes $R_i = \{v_{i0}, v_{i1}, \ldots, v_{ik+1}\}$, where $v_{ij} \in V$ and $v_{i0} = v_{ik+1} = 0$ (0 denotes the depot), satisfying $\sum_{v_{ij} \in R_i} q_j \leq Q$, which can be presented as:

$$Cost = \sum_{i=1}^{m} Cost(R_i) = \sum_{j=0}^{k} c_{ij+1}, \tag{1}$$

where matrix $C = (c_{ij})$ is a non-negative cost (distance or travel time) between customers v_i and v_j defined on E.

The CVRP consists determining a set of a maximum m routes with minimum total cost, such that each route starts and ends at the depot, each customer is visited exactly once by exactly one vehicle, subject to the restriction that the total demand of any route does not exceed Q. We additionally consider that the total duration of any route is no longer than a preset bound D [7].

Fig. 1 shows an example of CVRP routes in which the middle rectangle represents a depot and the smaller rectangles are different customers [8].

The rest of the paper is organized as follows. Section 2 reviews related works on the subject covered in this paper. The GELS algorithm is introduced in Section 3. Section 4 thoroughly explains the proposed algorithm. We show the results of simulations in Section 5 and conclude the work in Section 6.

2. Related work

VRP with capacity restrictions was first introduced by Dantzig and Ramser [9] in 1959. Following that work, many different algorithms were proposed to solve this problem that can be classified into two broad categories: heuristics and meta-heuristics methods.

Under the category of heuristic methods, we can name Savings, Tabu Search (TS) and Genetic Based Algorithms (GBA), which have successfully solved the VRP.

Clarke and Wright [10] proposed the use of Savings algorithm for solving VRP. The initial conditions were that only one vehicle services each customer and then there is N routes (where N is customer demands). Stanojevic et al. [11] used an advanced savings computations for solving the CVRP problem. They introduced a new method of combining routes to find a formula for calculating the savings, and employed this advanced combination for developing a new, extended, heuristic savings algorithm that recalculated savings through iterations. Computational results indicated that on an average, the extended savings algorithm was a better solution compared to the original savings algorithm.

The Granular Tabu Search (GTS) algorithm was proposed by Toth and Vigo first in 2003 [12] to solve the VRP with the aim of shortening computation time. The idea was implemented in conjunction with a TS method but the principle is general and could be beneficial to other types of algorithms. In this model neighbor solutions were obtained by performing intra-route and inter-route edge exchanges.

The most common heuristic methods that solved the CVRP are GBA. In [13] a Reduced Variable Neighborhood Search Algorithm (VNSA) for the VRP with capacity restrictions was proposed in which a diversion strategy by the crossover operator was designed to help the escape from local minima. This algorithm was tested on 34 CVRP samples and results indicated that its performance was better than other heuristic algorithms. Later, in [14] Genetic Algorithm (GA) was used to solve the CVRP problem. Results indicated that the algorithm was able to determine the optimal routes for the vehicles by considering their capacity restrictions and travel times. Nazif and Lee [7] introduced an Optimized Crossover Genetic Algorithm (OCGA) for the VRP with capacity restrictions. The main feature of the algorithm was that vehicles with similar capacities located in a depot were used in a way to optimize the routes and satisfy the demands of the customers. The proposed algorithm used an optimized crossover operator that employed a directed complete bipartite graph for finding an optimal set of delivery routes, in order to satisfy demands, and minimize the total costs. Considering the slow rate of convergence and the weak search ability of the traditional GA, the combinatorial GA (which has a greater convergence rate and rapid search ability) was developed to simplify the problem and improve its search efficiency [15,16].

Under the category of meta-heuristic methods, Pisinger and Ropke [17] introduced Large Neighborhood Search (LNS) to solve CVRP problem as a case study. Authors in [18,19] reported many VRP variants including the CVRP that can be transformed to a PDPTW and solved using an improved version of the Adaptive Large Neighborhood Search (ALNS) heuristic [20]. They argued that for most of the tested VRP variants the ALNS heuristic must be considered to be on par with or better than competing heuristics.

Vidal et al. [21] named Hybrid Genetic Search with Adaptive Diversity Control (HGSADC) to solve three important VRP classes: the MDVRP, the PVRP and the MDPVRP, effectively. The meta-heuristic proposed in this paper, combines the exploration breadth of population based evolutionary search, the aggressive-improvement capabilities of neighborhood-based meta-heuristics and advanced population-diversity management schemes. The method performs in terms of both solution quality and computational exigency. Authors tested the method on all available benchmark instances for the CVRP. The meta-heuristic equals or outperforms the current best methods proposed for each particular class and requires a limited computational effort. Moreover, with very limited adaptation, the algorithm also proves extremely competitive for the CVRP.

A number of algorithms used advantages of combining Simulated Annealing (SA) with other optimization methods. In [22] a Hybrid Ant Colony Optimization Routing Algorithm (HACO) was introduced for solving the CVRP problem that was a Hybrid of Simulated Annealing (HSA) and ACO Algorithm. Experimental results showed that it was able to solve the CVRP problem. Lin et al. [23] combined SA Algorithm with Local Search Algorithms (LSA). They reported the results of simulations on 14 classic instances and 20 large-scale samples. Their proposed algorithm found better solutions for 8 of the 14 classic instances.

Local search methods have also been used in some meta-heuristics in-order to solve CVRP. A new hybrid electromagnetism-like algorithm was proposed by Yurtkuran and Emel [24] that were population-based and dependent on the attraction-repulsion mechanism between charged particles. The obtained solutions were improved using a local search method and the iterated Swap Procedure, and the algorithm was tested on several benchmark problems. Computational results showed that the proposed algorithm yielded acceptable results compared to other meta-heuristic methods. The CVRP studied in [25] was an optimization problem in which a set of vehicle tours was employed to satisfy all demands identically. The main procedure introduced in the algorithm was a suitable neighbor research project intending to reduce many of the deviations between minimum and maximum travel times and deviations related to the loading capacity of the fleet of vehicles.

3. Gravitational Emulation Local Search Algorithm (GELS)

In 1995 Voudouris and Tesang [26] proposed the Guided Local Search Algorithm (GLS) for searching and solving NP-complete problems. Later Webster [27] presented this algorithm as a powerful algorithm and called it GELS algorithm. This

algorithm is based on randomization concepts along with two velocity and force parameters, and uses the random numbers provided by existing Local Search Algorithms (LSA) for avoiding the local optimum. The idea of this algorithm is based on the gravitation force principle in nature that causes the objects to attract each other; heavier objects have more gravitational force and apply it to other objects, and attracting lighter objects toward themselves. The distance between objects has strong effect on the gravitational force, by increasing the distance, the gravitational force between objects decreases by the power of 2 and vice versa. In GELS, the Newton gravitational force between two objects is as below:

$$F = \frac{Gm_1m_2}{R^2}. \tag{2}$$

In which m_1 and m_2 are the masses of the first and second object respectively. G is the gravitational force constant 6.672 and R (Radius) is the distance between two objects.

GELS also imitates this process of nature in order to search in a search space. In this algorithm, the search space is considered as a large space and its objects are considered feasible solutions for search. Each of these objects (solutions) has its weight; the weight of each object is its performance and the search criterion is that the best solution has the most weight and no object can have zero weight [28].

In this method, the feasible solutions in search space are classified based on criteria, which depend on the type of the problem. Each of these classes is a dimension of the solution and for each dimension, a value, which is called initial velocity, is considered.

GELS calculates the gravitational force between the solutions in the search space by two methods. In the first method, a solution from the local neighborhood space is selected as the Current Solution (*CU*) and the gravitational force between these two solutions will be calculated. The second method, applies the formula to all the solutions in the neighborhood and the gravitational force between all of them and the *CU* is calculated. GELS can behave in two ways when moving in the search space; in the first method it is possible only to move to available solutions in the current neighborhood space. Each of these motion methods can be used in each calculation for gravitational force and therefore four models for GELS are built [29].

GELS includes a vector that its size indicates the solution dimensions. The vector components indicate the relative velocity in each dimension. The algorithm starts with an initial solution, initial velocity vector and moving direction. For each dimension in velocity vector, a random number between one and maximum velocity is selected and this is the component value in each direction. The moving direction of each dimension in the initial velocity vector is determined with respect to the initial velocity vector of solution dimensions and this direction is equal to the solution dimension with the maximum initial velocity in the initial velocity vector.

In each iteration of the algorithm, based on the current moving direction, a Candidate Solution (*CA*) is selected from the local neighborhood space and the gravitational force between the *CU* and *CA* is calculated and then the related velocity vector is updated. For the next iteration, the velocity vector is checked and a moving direction is selected.

Every iteration of the algorithm in the second method is completely similar to the first method but there is only a minor difference between them. In the second method in the update stage, instead of calculating the gravitational force and updating the velocity vector, in order to achieve the *CA* from the *CU* the gravitational force and initial velocity is calculated for all the *CA*. In the employed Newton formula, the two masses in the numerator of the fraction are replaced by the difference between the *CA* cost and *CU* cost. Thus the gravitational force between two objects is calculated from the following relation [30,31]:

$$F = \frac{G(CU - CA)}{R^2}. \tag{3}$$

In which *CU* and *CA* are the *CU* cost and the *CA* cost respectively. If the *CU* cost is greater than the *CA* cost, the result of this formula is positive and if the *CA* cost is greater, the result of this formula is negative. Then the result of this force (negative or positive) is added to the velocity vector in the current moving direction. If this operation makes the velocity value exceed the maximum limit, the same maximum value is considered and if updating makes the velocity negative, the velocity value is considered to be zero [32,33].

The available velocities in GELS are as follows:

Maximum velocity: The maximum value that can be assigned to each component of the initial velocity vector.

Radius: The radius used in the calculation of the gravitational force.

Iteration: Indicates the maximum number of iterations, which guarantee that algorithm will end.

The parameters can be adjusted by doing repeated experiments and by trial and error.

Algorithm 1 shows the GELS algorithm. This algorithm shows that an initial response to a problem is created, and each mass is evaluated. Next, the problem is updated as *G*, or *Best*, and/or *Worst,* and the parameters *m* and *a* are calculated for each mass. Then *velocity* and the *location* of each mass are also updated. Finally, the algorithm is concluded if the maximum number of iterations meets or all the initial velocity vector elements become zero. Otherwise, the algorithm goes back to Step 2 [34].

Algorithm 1 The GELS Algorithm.

1. Generate initial population
2. **While** (termination condition is not satisfied)
   3. Evaluate the fitness for each agent
   4. Update the *G*, *Best* and *Worst* of the population
   5. Calculate *m* and *a* parameters for each agent
   6. Update velocity and position
7. **End While**
8. **Return** best solution

4. The proposed algorithm

In the proposed method, GELS algorithm is used as a strategy to solve the CVRP problem and we called as *CVRP_GELS*. The objective of this algorithm is to find the shortest path between customers, reducing the number of vehicles and traveling time of all the vehicles.

4.1. Defining the solution dimensions

In the proposed method, every dimension of the solution can be considered as a customer. In fact the number of solution dimensions is equal to the number of customers. The neighbor of the *CU* in the considered dimension is the customer who has the least distance, time and the most velocity with the customer of the *CU* for every vehicle and the vehicle has not met the customer so far.

4.2. Definition of neighborhood

In GELS algorithm unlike other algorithms, searching the neighborhood solution is not carried out randomly but every *CU* has different neighbors and each of them is based on a particular change, which is named the moving direction toward the neighboring solution. All the neighbors that are obtained based on this method, are only based on this neighbor. In the proposed method in order to find the neighboring solution, the following procedure is used: the customer with the least distance, time and the most velocity with respect to the *CU* is selected as the neighbor and *CA* for every vehicle.

4.3. Solution method

In the proposed method, GELS algorithm is applied to solve the CVRP. The algorithm's objective is to reduce the traveling time and distance and also to reduce the number of employed vehicles. Considering the complexity of the problem, achieving a solution is difficult even if there is a few numbers of vehicles and customers. Therefore, considering the characteristics of gravitational force algorithm and its global feature, applying this algorithm is proposed for solving CVRP problem.

Considering the objective of the problem, which is reducing the traveling distance and number of vehicles, this paper introduces the gravitational optimization algorithm (GELS) as a suitable strategy for solving the CVRP problem. Contrary to other algorithms, the gravitational force algorithm (GELS) does not proceed with random solutions, but it proceeds by examining the present solutions and provides distinct final conditions that can complete the previous direction for a specific iterate to continue counting. Thus, although the GELS algorithm has a number of elements based on random operators, it does not proceed purely based on random operators. Although it employs the local neighborhood search method for solving the problem, it does not move in the same way between them and although it includes a particular behavior of greedy algorithms in itself, it does not find the best path for searching. The GELS algorithm uses the same law that guides the motion of objects in physical space in order to control their motion in a complex search space.

Unlike other algorithms that start from an initial population and then generate the next generations, the GELS algorithm starts from an initial or *CU* and in the next step the secondary or *CA* is generated based on the neighbors, customers and vehicles that can satisfy the problem constraints. In this step, the gravitational force between these two masses is calculated and it is considered as a solution for the problem and then velocity and gravitational mass are updated.

One of the advantages of this algorithm when compared to other algorithms is the presence of velocity factor, which makes this algorithm to search only for the best and optimum solutions and attract them. This process leads to the growth of object's mass and as we know in the gravitational law, as the object mass increases the gravitational force increases as well and thus the object attracts more optimum solutions toward itself. This process is repeated and in every iteration, by attracting the optimum solutions, the object's mass is added and therefore the algorithm achieves the best solution.

To solve the CVRP problem, we first consider three matrices: distance, initial velocity and time. The distance matrix is taken from standard problems in [35] and is calculated for every sample separately. Based on gravitational algorithm, the initial velocity is a constant value at first, in this problem the initial velocity of all the masses is considered equal to 100. The initial velocity is updated in every stage. As it was explained, in the initial velocity matrix, an initial velocity equal of

Table 1
CVRP_GELS Steps.

1.	To create initial response by GELS algorithm
2.	To determine the perimeter of system and initialize customers who are considered as bodies.
3.	To arrange solutions created by customers based on each customer's weight.
4.	To select the first solution as the best one.
5.	To evaluate the weight of each customer.
6.	To calculate *G(t)*.
7.	To calculate the Gravity force imposed on each customer.
8.	To calculate acceleration, time and speed of each customer.
9.	To update parameters *T* and *V*.
10.	Displace solutions in each dimension based on the force exerted on them in different dimensions.
11.	If stop condition is satisfied, run steps 12; otherwise, go to step 4.
12.	Turn the best response.

Table 2
Algorithm 2: CVRP_GELS Algorithm.

```
Data: I, K, M, A, B, R, F, V, NBListSize
Result: Distance, Time, BestFit A, BestFit B
initialization;
Parameters I, K, M, NBListSize, Distance, Speed, Time, A, B, R, F, V, BestFit A,
BestFit B.
//Generate a feasible solution using the GELS algorithm
Distance= Create Matrix Distance()
Speed= Create Matrix Speed()
Time= Create Matrix Time()
A= Create Parente()
B= Create Child()
BestFitA= Fitness A
BestFitB= Fitness B
while Number Of City's Region do
    Empty B
    B= Create Child()
End
```

100 is assigned to each customer who is considered as a mass and then in next stages, velocity will be changed and also time (mass) matrix is obtained based on distance matrix and velocity matrix according to the following equation:

$$T = \frac{\sqrt{(Y_B - Y_A)^2 + (X_B - X_A)^2}}{V_{inA,B}}. \tag{4}$$

In (4), T indicates mass of the customer, $(Y_B - Y_A)^2 + (X_B - X_A)^2$ is the distance between the two customers and $V_{in_{A,B}}$ shows the velocity between the two customers.

Using the GELS algorithm in these conditions, an appropriate reserve factor should be defined. The reserve factor is equal to the number of reserved customers for cars in the future. The set of assigned customers to the cars is a solution. This method can be used to represent a solution in the form of two (n^*n) matrices, which are equal to the number of cars and customers. Every row and column indicates one of the customers in the group and the number in each row and column indicates the number of cars that customers belong to them. When the algorithm is completed, the reserved cars for the customers with the reservation factors are presented.

The algorithm stops when the assigned speed is zero or the number of iterations of the algorithm reaches the maximum value determined by the problem. In the following tables, we simply explain the steps of our algorithm (Table 1) and the pseudo code of *CVRP_GELS* in detail (Table 2).

5. Simulation results

Experimental results of the proposed algorithm *(CVRP_GELS)* are presented with details in this section, and are compared with those of other popular meta-heuristic methods. We use four different benchmarks in order to test and comparer our method. A large number of algorithms were executed on these benchmarks and, hence, a good comparison could be made between the proposed *CVRP_GELS* algorithm and other algorithms. The first benchmark with 14 instances is proposed by Christofides [36], the second benchmark that has 12 different problems is proposed by Taillard [37], the third benchmark which is our main benchmark is introduced by Uchoa et al. [38] and the fourth benchmark is proposed by Golden et al. [39] and compared with State-of-the-Art methods.

It is obviously challenging to compare the performance of algorithms whose testing was done using different computers. However, a very simple hint of the machine relative speed may be derived from the flops measure using a benchmark as a classic numerical application. Within this simplified setting, the flops ratios may be used to convert CPU times of different

Table 3
Instances' characteristics of Christofides et al. [41].

Instance	C	K	Q	m.t.l	s.t
C1	50	5	160	∞	–
C2	75	10	140	∞	–
C3	100	8	200	∞	–
C4	150	12	200	∞	–
C5	199	17	200	∞	–
C6	50	6	160	200	10
C7	75	11	140	160	10
C8	100	9	200	230	10
C9	150	14	200	200	10
C10	199	18	200	200	10
C11	120	7	200	∞	–
C12	100	10	200	∞	–
C13	120	11	200	720	50
C14	100	11	200	1040	90

C: number of costumer, **K:** minimum number of used vehicle, **Q:** capacity of vehicle, **m.t.l.:** maximum tour length, **s.t.:** service time.

machines [12]. In our computational testing we implemented our codes using Matlab Language and run the codes on Intel(R) Core(TM) i7 CPU 950 @ 3.07 GHz (24,596 Mflops), and we use a conversion factor in order to make our algorithm compatible to other algorithms in different tables of this section according to Dongarra benchmarks [40].

5.1. Experiments on Christofides [35] benchmark

In this benchmark, 14 standard instances were considered that has a combination of problems with 50–199 nodes (in addition to the depot). Moreover, among these 14 instances, the first 10 (C1–C10) include customers who were randomly distributed around the depot, while the customers in the other 4 instances belonged to more distant classes and the depot was not located in their centers. Furthermore, all considered instances had capacity restriction, but instances C6–C10, C13, and C14 had the additional route length and service time restrictions. In other words, no vehicle in these instances could travel more than a specific length of the routes.

Table 3 lists the characteristics of these instances. In this table, **C** indicates the number of costumer, **K:** minimum number of used vehicle, **Q:** capacity of vehicle, **m.t.l.:** maximum tour length, **s.t.:** service time.

Tables 4 and 5 list execution times (in seconds) obtained from the comparison of the proposed algorithm with a large number of available methods for the VRP problem. It must be noted that the instances shown in these tables can be divided into three classes based on the number of the customers: small (fewer than 100), medium (100–150), and large (more than 150) customers. Therefore, results of applying the algorithm on these three classes can be studied separately.

We compare our proposed algorithm against the BKS results summarized in Table 4. Concerning the instances of Christofides et al. [41], we have selected some algorithms namely: Tabu Search (TS) and Savings Ants (SA) by Osman [42], and (SS_ACO) algorithm, which is to hybridize the solution construction mechanism of ACO and Scatter Search (SS) to solve the Vehicle Routing Problem by Zhang and Tang [43], Particle Swarm Optimization (PSO) by Ai and Kachitvichyanukul [44], and Hybrid Genetic–PSO–GRASP–ENS for the Vehicle Routing Problem (Hyb GENPSO) by Marinakis and Marinaki [45]. For the proposed algorithm we show three columns. The first column shows the average solution quality (averaged over the 10 experiments), and the second column shows the best result out of 10 experiments. The third column shows the root CPU time (in seconds) used for finding the solutions. Bold entries mark the best solution quality obtained among the heuristics in the comparison.

As per Table 4, the first class contains 4 instances, in which, the proposed algorithm was almost completely successful and returned the best solutions found so far in the 75% of the instances. This indicates that the proposed algorithm has a good efficiency for small instances. In the second class for the next 6 instances, the proposed algorithm could find the best solutions for 83% of the instances. Therefore, it can be concluded that the proposed algorithm was highly efficient in solving medium size instances. Finally, in the third class with 4 instances, it exhibited good efficiency the same as first class, and could return the best solutions found so far for 75% of the instances. Moreover, we assumed that the two algorithms SA and TS can obtain the best solutions found so far if they found the correct parts of the solutions returned by BKS. With this assumption, SA was able to find optimal solutions for only 1 of the instances, while TS could obtain the answers found by BKS in 3 of the instances. Another algorithm that could obtain solutions similar to those of the SA algorithm was the (SS-ACO) algorithm. It was able to return BKS solutions in only 1 of the instances, while the PSO algorithm, found the best solutions in 4 of the instances and could find best solutions compared to the other one mentioned algorithms. Furthermore, this algorithm obtained solutions that were weaker compared to the (Hyb GENPSO) algorithm that returned BKS results in 6 of the instances. It must be noted that in this table the *CVRP_GELS* algorithm obtained the best solutions and, as mentioned earlier, found the best solutions for 8 of the instances.

Table 4
Computational results for the benchmark of Christofides et al. [41]. The bold values mean the best-so-far results that are found by the algorithms.

Instance	SA [42]		TS [42]		SS-ACO [43]		PSO [44]		Hyb GENPSO [45]		CVRP_GELS			BKS
	Best val.	Time (s)	Best val.	Time (s)	Best val.	Time (s)	Best val.	Time (s)	Best val.	Time (s)	Avg. of 10	Best of 10	Time (s)	
C1	528.5	0.33	**524.5**	0.22	524.61	32.39	524.61	–	524.61	0.70	528	524.61	51	524.61[a]
C2	838.10	12.56	844.10	0.35	**835.26**	41.23	844.42	–	**835.26**	15.33	838	**835.26**	68	**835.26**[a]
C3	829.8	18.22	838.8	3.01	830.14	70.67	829.40	–	**826.14**	17.42	830	**826.14**	136	**826.14**[a]
C4	1058.12	9.78	1044.12	6.95	1038.20	147.83	1048.89	–	**1028.42**	43.21	1035	**1028.42**	173	**1028.42**[a]
C5	1378.16	4.52	1334.16	6.33	1307.18	416.98	1323.89	–	1294.21	126.14	1302	1294.21	541	**1294.29**[b]
C6	555.6	6.65	555.6	0.34	559.12	38.28	**555.43**	–	**555.43**	0.70	559	**555.43**	58	**555.43**[a]
C7	**909.11**	1.22	911.11	2.06	912.68	53.01	917.68	–	909.68	13.94	920	914.13	89	909.68[a]
C8	866.9	1.87	878.9	5.85	869.34	123.68	867.01	–	**865.94**	43.90	876	869.34	192	**865.94**[a]
C9	1164.14	164.52	1184.14	9.28	1179.4	306.85	1181.14	–	1163.41	75.96	1174	**1162.55**	437	**1162.55**[a]
C10	1417.18	11.14	1441.18	8.90	1410.26	596.03	1428.46	–	1397.51	152.62	1405	**1395.85**	997	**1395.85**[c]
C11	1176.7	0.62	1043.7	2.82	1044.12	136.64	1051.87	–	**1042.11**	10.45	1058	**1042.11**	225	**1042.11**[a]
C12	826.10	1.23	**819.10**	1.74	824.31	91.88	819.56	–	819.56	18.12	829	819.56	149	819.56
C13	1545.11	14.88	1545.11	5.53	1556.52	275.04	1546.20	–	1544.57	21.60	1564	**1541.14**	451	**1541.14**[a]
C14	890.11	0.60	**866.11**	2.30	870.26	217.33	866.37	–	866.37	16.73	873	866.37	357	866.37[a]

The column Best val. indicates the number of vehicles used in the best found solution. The column Time indicates time of the best run of tested algorithms respectively.
Bold: Indicates the solution that minimizes the number of vehicles with least distance traveled.
SA: Savings Ants (SA) algorithm from Osman [42]; VAX 8600 Computer using Fortran 77 language, Performance = 48 Mflops.
TS: Tabu Search (TS) algorithm from Osman [42]; VAX 8600 Computer using Fortran 77 language, Performance = 48 Mflops.
SS_ACO: Hybridize the solution construction mechanism of ACO and Scatter Search (SS) to solve the Vehicle Routing Problem from Zhang and Tang [43]; IBM computer with 512 MB RAM and 1600 MHz CPU using visual C++ language, Performance = 14,486 Mflops.
PSO: Particle Swarm Optimization (PSO) from Ai and Kachitvichyanukul [44]; Intel Pentium 4, CPU 3.4 GHz and 1 GB RAM using Microsoft Visual Studio C#.Net language.
Hyb GENPSO: Hybrid Genetic–PSO–GRASP–ENS for the Vehicle Routing Problem (Hyb GENPSO) from Marinakis and Marinaki [45]; Intel Pentium M 750 at 1.86 GHz, using Fortran 90 and was compiled using the Lahey f95 compiler, Linux 9.1, Performance = 17,141 Mflops.
CVRP_GELS: Capacitated Vehicle Routing Problem Using the Gravitational Emulation Local Search Algorithm; Pentium core i7 950, at 3.07 GHz with 16 GB RAM using Matlab language, Performance = 24,596 Mflops.
BKS: Best Known Solution.
–: No value is calculated for this case.
Note. Computing times are expressed in seconds based on a machine with the performance of 24,596 Mflops.
[a] Taillard [37].
[b] Mester and Braysy [50].
[c] Rochat and Taillard [56].

Table 5 compared the proposed algorithm with other five meta-heuristic algorithms on the same benchmark as mentioned in Table 4. Namely, Optimized Crossover Genetic Algorithm (OCGA) by Nazif and Lee [7], an Active Guided Evolution Strategies Meta-heuristic Called (AGES) proposed by Mester and Braysy [46], adaptive memory programming method called (SEPAS) by Tarantilis [47], and GAs that are those of Prins (GA_P) [48] and Berger and Barkaoui (GA_BB) [49] and also against the BKS results reported in the literature concerning the instances of Christofides et al. [41] benchmark. For the proposed algorithm we show three columns. The first column shows the average solution quality (averaged over the 10 experiments), the second column shows the best result out of 10 experiments and the three column shows the root CPU time required to run every algorithm on an instance (in seconds) to find the solutions. In this set, our algorithm could find the best solution for C10, which could not solve by other compared algorithms.

5.2. Experiments on Taillard [37] benchmark

The second benchmark includes 12 instances, introduced by Taillard [37]. The instances, divided into three groups that have 75, 100, or 150 nodes. There were 4 instances in each group, the number of used vehicles was 9 or 10 in group 1, 11 or 12 in group 2, and 14 or 15 in group 3, and an algorithm could set to solve the problem using both number of vehicles. Table 6 presents the complete characteristics of these groups.

Table 7 shows the instances suggested by Taillard [37] and the execution times (in seconds) of the proposed algorithms for these instances. The instances suggested by Taillard have been tested on a limit number of algorithms compared to the benchmark introduced by Christofides et al. [41]. We include three most important algorithms in Table 7. Namely, Optimized Crossover Genetic Algorithm (OCGA) by Nazif and Lee [7], an Active Guided Evolution Strategies Meta-heuristic Called (AGES) proposed by Mester and Braysy [50], and a Cellular Genetic Algorithm Called (JCell2o1i) by Alba and Dorronsoro [51]. Like in Tables 4 and 5, for the proposed algorithm we show three columns. The first column shows the average solution quality (averaged over the 10 experiments), and the second column shows the best result out of 10 experiments. The third column shows the CPU time (in seconds) used for finding the solutions. Bold entries mark the best solution quality obtained among the heuristics in the comparison.

In these tests, the proposed algorithm was able to prove its efficiency. It could successfully find 8 out of 12 solutions and obtained good solutions for three other instances. On the other hand, OCGA [7] found 6 out of 12 solutions successfully,

Table 5
Computational results for the benchmark of Christofides et al. [41]. The bold values mean the best-so-far results that are found by the algorithms.

	OCGA [7]	AGES [46]	SEPAS [47]	GA_P [48]	GA_BB [49]	CVRP_GELS			BKS
Instance	Best val.	Best val.	Best val.	Best val.	Best val.	Avg. of 10	Best of 10	Time (s)	
C1	**524.61**	**524.61**	**524.61**	**524.61**	**524.61**	528	**524.61**	51	524.61[a]
C2	**835.26**	**835.26**	**835.26**	**835.26**	**835.26**	838	**835.26**	68	835.26[a]
C3	**826.14**	**826.14**	**826.14**	**826.14**	827.39	830	**826.14**	136	826.14[a]
C4	**1028.42**	**1028.42**	**1028.42**	1030.46	1036.16	1035	**1028.42**	173	1028.42[a]
C5	1299.64	**1291.29**	1311.48	1296.39	1324.06	1302	1294.21	541	1291.29[b]
C6	**555.43**	**555.43**	**555.43**	**555.43**	**555.43**	559	**555.43**	58	555.43[a]
C7	**909.68**	**909.68**	**909.68**	**909.68**	**909.68**	920	914.13	89	909.68[a]
C8	**865.94**	**865.94**	**865.94**	**865.94**	868.32	876	869.34	192	865.94[a]
C9	1163.38	**1162.55**	**1162.55**	**1162.55**	1169.15	1174	**1162.55**	437	1162.55[a]
C10	1406.23	1401.12	1407.21	1402.75	1418.79	1405	**1395.85**	997	1395.85[c]
C11	**1042.11**	**1042.11**	**1042.11**	**1042.11**	1043.11	1058	**1042.11**	225	1042.11[a]
C12	**819.56**	**819.56**	**819.56**	**819.56**	**819.56**	829	**819.56**	149	819.56[a]
C13	1542.25	**1541.14**	1544.01	1542.86	1553.12	1564	**1541.14**	451	1541.14[a]
C14	**866.37**	**866.37**	**866.37**	**866.37**	**866.37**	873	**866.37**	357	866.37[a]

The column Best val. indicates the number of vehicles used in the best found solution. The column Time indicates time of the best run of tested algorithms respectively.
Bold: Indicates the solution that minimizes the number of vehicles with least distance traveled.
OCGA: Optimized Crossover Genetic Algorithm (OCGA) from Nazif and Lee [7]; Pentium 4, 2.0 GHz computer with 2.0 GB RAM using C language.
AGES: Active Guided Evolution Strategies Meta-heuristic Called (AGES) proposed from Mester and Braysy [46]; Pentium IV Net Vista PC2800 MHz with 512 MB RAM using Visual Basic 6.0 language.
SEPAS: Adaptive memory programming method called (SEPAS) from Tarantilis [47]; Pentium II/400 with 128 MB RAM using Visual C++ language.
GA_P: GAs that are those of Prins (GA_P) [48]; Pentium-3PC clocked at 1 GHz under the operating system Windows 98 using Pascal-like, Delphi 5 language.
GA_BB: Berger and Barkaoui (GA_BB) [49]; Pentium 400 MHz using C++ language.
CVRP_GELS: Capacitated Vehicle Routing Problem Using the Gravitational Emulation Local Search Algorithm; Pentium 7, at 3.07 GHz with 16 GB RAM using Matlab language.
BKS: Best Known Solution.
Note. Computing times are expressed in seconds based on a machine with the performance of 24,596 Mflops.
[a] Taillard [37].
[b] Mester and Braysy [50].
[c] Rochat and Taillard [56].

Table 6
Instances' characteristics of Taillard [37].

Instance	C	K	Q	m.t.l	s.t
Tai75a	75	9 or 10	1445	∞	–
Tai75b	75	9 or 10	1679	∞	–
Tai75c	75	9 or 10	1122	∞	–
Tai75d	75	9 or 10	1699	∞	–
Tai100a	100	11 or 12	1409	∞	–
Tai100b	100	11 or 12	1842	∞	–
Tai100c	100	11 or 12	2043	∞	–
Tai100d	100	11 or 12	1297	∞	–
Tai150a	150	14 or 15	1544	∞	–
Tai150b	150	14 or 15	1918	∞	–
Tai150c	150	14 or 15	2021	∞	–
Tai150d	150	14 or 15	1874	∞	–

C: number of costumer, **K:** minimum number of used vehicle, **Q:** capacity of vehicle, **m.t.l.:** maximum tour length, **s.t.:** service time.

AGES [50] found 7 successful solutions and JCell2oil [51] could find only 4 out of 12 solutions. Comparing the results, confirm that the proposed *CVRP-GELS* algorithm works best to solve the problems proposed in these 12 instances.

5.3. Experiments on Uchoa et al. [38] benchmark

In order to compare our method with the state of the art methods, we used a benchmark introduced by Uchoa et al. [38]. The introduced benchmark contains a set of instances running from 100 to 1000 customers. We report results of our experiment jointly in Tables 8–10, which compared our method with other methods based on the average solution quality, best solutions quality and average CPU time of every method, the lower bound, root CPU time, number of search nodes and overall time, and finally the cost and number of vehicles of the BKS ever found since the instances were created, including preliminary runs of those heuristics with alternative parameterizations.

We selected two of the successful meta-heuristics that have been tested on the same benchmark. The first method that we choose is an efficient neighborhood-based method, the Iterated Local Search based meta-heuristic algorithm (ILS-SP)

Table 7
Computational results for the benchmark of Taillard [38]. The bold values mean the best-so-far results that are found by the algorithms.

Instance	OCGA [7]	AGES [50]	JCell2o1i [51]	CVRP_GELS			BKS
				AVG. of 10	Best of 10	Time (s)	
Tai75a	**1618.36**	**1618.36**	**1618.36**	1625	**1618.36**	39	**1618.36**[a]
Tai75b	1344.63	1344.64	**1344.62**	1348	**1344.62**	48	**1344.62**[b]
Tai75c	**1291.01**	**1291.01**	**1291.01**	1298	**1291.01**	41	**1291.01**[a]
Tai75d	**1365.42**	**1365.42**	**1365.42**	1369	**1365.42**	63	**1365.42**[a]
Tai100a	2050.64	**2041.34**	2047.90	2056	**2041.34**	75	**2041.34**[c]
Tai100b	**1939.90**	**1939.90**	1940.36	1967	1947.07	89	**1939.90**[d]
Tai100c	1408.40	**1406.20**	1411.66	1429	**1406.20**	84	**1406.20**[c]
Tai100d	**1581.22**	1581.25	1584.20	1609	1581.25	92	1581.25[c]
Tai150a	**3055.23**	**3055.23**	3056.41	3099	3069.14	117	**3055.23**[a]
Tai150b	2755.09	2727.67	2732.75	2769	**2656.47**	114	**2656.47**[c]
Tai150c	2352.86	2343.11	2364.08	2394	**2341.84**	138	**2341.84**[a]
Tai150d	2660.33	2645.40	2654.69	2702	2659.02	129	**2645.39**[e]

Bold: Indicates the solution that minimizes the number of vehicles with least distance traveled.
OCGA: Optimized Crossover Genetic Algorithm (OCGA) from Nazif and Lee [7]; Pentium 4, 2.0 GHz computer with 2.0 GB RAM using C language.
AGES: Active Guided Evolution Strategies Meta-heuristic Called (AGES) proposed from Mester and Braysy [46]; Pentium IV Net Vista PC2800 MHz with 512 MB RAM using Visual Basic 6.0 language.
JCell2o1i: Cellular Genetic Algorithm Called (JCell2o1i) from Alba and Dorronsoro [51]; 2.8 GHz computer using Java language.
CVRP_GELS: Capacitated Vehicle Routing Problem Using the Gravitational Emulation Local Search Algorithm; Pentium 7, at 3.07 GHz with 16 GB RAM using Matlab language.
BKS: Best Known Solution.
Note. Computing times are expressed in seconds based on a machine with the performance of 24,596 Mflops.
[a] Taillard [37].
[b] Alba and Dorronosoro [51].
[c] Gambardella et al. [57].
[d] Mester and Braysy [50].
[e] Rochat and Taillard [56].

of Subramanian et al. [52]. The second method that we consider is a recent population-based method, the Unified Hybrid Genetic Search (UHGS) of Vidal et al. [21,53].

As reported in [21,52,53], ILS-SP and UHGS have been tested on a Xeon CPU with 3.07 GHz and 16 GB of RAM, running under Oracle Linux Server 6.4. The average and best results on 50 runs are reported in Tables 8–10, as well as the average computational time per instance similar to reported tables of benchmark Uchoa et al. [38]. Table 10 also reports additional statistics on instances characteristics and results, such as the minimum, maximum, average and median results on the instance set for n, Q, r, n/K_{min}, as well as for the Gaps (%) and CPU time of ILS-SP, UHGS and *CVRP_GELS*.

As shown in Table 10, all the tested methods namely, ILS-SP, UHGS and *CVRP_GELS*, find solutions of consistent quality, with an average gap of 0.52%, 0.19% and 0.16%, respectively, with respect to the BKS ever found during all experiments.

CVPR_GELS finds solutions of generally higher quality than both UHGS and ILS for a comparable amount of CPU time specially when comparing its execution time with execution of the UHGS method. In some cases our method could find solutions with less number of required vehicles compare to BKS in a very efficient manner which is another advantage of this algorithm.

Overall, these experiments revealed that the proposed method could be a very efficient method in order to solve CVRP and its results are comparable with the results of state-of-the-art methods.

5.4. Experiments on Christofides et al. [41] and Golden et al. [39] benchmarks

In order to compare our method with more methods, in Table 11 we compare the results of GTS proposed by [12], XK algorithm proposed by Xu and Kelly [54] and with those of a deterministic annealing heuristic, called RTR, described in Golden et al. [39]. In this test we use fifteen CVRP instances proposed by Christofides et al. [41] and Golden et al. [39] which are generally used as a standard benchmark for CVRP algorithms. The results of the GTS, XK and RTR algorithms for the CVRP are those reported in [12]. On these widely used test instances, *CVRP_GELS* proved to solve the problems, as fast as GTS with better solutions found in average. The *CVRP_GELS* solution is worse than the GTS only in two out of the fifteen CVRP instances and found better solutions than the BKS in nine cases.

Table 12 summarizes the results of comparing four well-known algorithms that solved CVRP problem. In this test, we compared our results using 14 instances (P01–P14) of Christofides et al. [41] ranging from 50 to 199 customers and 20 large-scale instances (Pr01–Pr20) of Golden et al. [39], ranging from 200 to 483 customers with average results of other four algorithms based on the report given by Vidal et al. in [21].

In Table 12 the first two columns display the instance identifier and the number of customers respectively, while the next columns compare the average performance of *CVPR_GELS* to the performance of Hybrid Genetic Algorithm of Prins [6] (Hybrid GA), the guided evaluation strategies of Mester et al. [46] (AGES), edge-assembly crossover based metric algorithm of Nagata et al. [55] (EAX) and Hybrid Genetic Search Adaptive Diversity Control of Vidal et al [21] (HGSADC). We also

Table 8
New set of benchmark instances: characteristics and results of current state-of-the-art algorithms (Part I). The bold values mean the best-so-far results that are found by the algorithms.

#	Name	Instance Characteristics							ILS-SP [52]			UHGS [21, 53]			CVRP_GELS			BKS	
		n	Dep	Cust	Dem	Q	r	n/K_{min}	Avg	Best	T(min)	Avg	Best	T(min)	Avg	Best	T(min)	Value	NV
1	X-n101-k25	100	R	RC (7)	1–100	206	4.0	4.0	27,591.0	**27,591**	0.13	27,591.0	**27,591**	1.43	27,591.0	**27,591**	1.24	27,591	26
2	X-n106-k14	105	E	C (3)	50–100	600	8.0	7.5	26,375.9	**26,362**	2.01	26,381.8	26,378	4.04	26,375.9	**26,362**	3.87	26,362	14
3	X-n110-k13	109	C	R	5–100	66	8.8	8.4	14,971.0	**14,971**	0.20	14,971.0	**14,971**	1.58	14,971.0	**14,971**	1.35	14,971	13
4	X-n115-k10	114	C	R	SL	169	12.5	11.4	12,747.0	**12,747**	0.18	12,747.0	**12,747**	1.81	12,747.0	**12,747**	1.72	12,747	10
5	X-n120-k6	119	E	RC (8)	U	21	21.8	19.8	13,337.6	**13,332**	1.69	13,332.0	**13,332**	2.31	13,332.0	**13,332**	2.21	13,332	6
6	X-n125-k30	124	R	C (5)	Q	188	4.2	4.1	55,673.8	**55,539**	1.43	55,542.1	**55,539**	2.66	55,542.1	**55,539**	2.57	55,539	30
7	X-n129-k18	128	E	RC (8)	1–10	39	7.4	7.1	28,998.0	28,948	1.92	28,948.5	**28,940**	2.71	28,944.7	**28,940**	2.63	28,940	18
8	X-n134-k13	133	R	C (4)	Q	643	10.4	10.2	10,947.4	**10,916**	2.07	10,934.9	**10,916**	3.32	10,938.2	**10,916**	3.11	10,916	13
9	X-n139-k10	138	C	R	5–10	106	14.0	13.8	13,603.1	**13,590**	1.60	13,590.0	**13,590**	2.28	13,586.4	**13,590**	2.17	13,590	10
10	X-n143-k7	142	E	R	1–100	1190	22.6	20.3	15,745.2	15,726	1.64	15,700.2	**15,700**	3.10	15,743.1	15,709	3.02	15,700	7
11	X-n148-k46	147	R	RC (7)	1–10	18	3.2	3.2	43,452.1	**43,448**	0.84	43,448.0	**43,448**	3.18	43,445.6	**43,448**	3.08	43,448	47
12	X-n153-k22	152	C	C (3)	SL	144	7.1	6.9	21,400.0	21,340	0.49	21,226.3	**21,220**	5.47	21,223.1	**21,220**	5.34	21,220	23
13	X-n157-k13	156	R	C (3)	U	12	12.0	12.0	16,876.0	**16,876**	0.76	16,876.0	**16,876**	3.19	16,876.0	**16,876**	3.10	16,876	13
14	X-n162-k11	161	C	RC (8)	50–100	1174	15.5	14.6	14,160.1	**14,138**	0.54	14,141.3	**14,138**	3.32	14,152.7	**14,138**	3.14	14,138	11
15	X-n167-k10	166	E	R	5–10	133	17.8	16.6	20,608.7	20,562	0.86	20,563.2	**20,557**	3.73	20,563.2	**20,557**	3.63	20,557	10
16	X-n172-k51	171	C	RC (5)	Q	161	3.4	3.4	45,616.1	**45,607**	0.64	45,607.0	**45,607**	3.83	45,610.8	**45,607**	3.79	45,607	53
17	X-n176-k26	175	E	R	SL	142	6.8	6.7 4	48,249.8	48,140	1.11	47,957.2	**47,812**	7.56	47,951.4	**47,812**	7.47	47,812	26
18	X-n181-k23	180	R	C (6)	U	8	8.4	7.8	25,571.5	**25,569**	1.59	25,591.1	**25,569**	6.28	25,583.7	**25,569**	6.16	25,569	23
19	X-n186-k15	185	R	R	50–100	974	13.0	12.3	24,186.0	**24,145**	1.72	24,147.2	**24,145**	5.92	24,159.1	**24,145**	5.82	24,145	15
20	X-n190-k8	189	E	C (3)	1–10	138	25.0	23.6	17,143.1	17,085	2.10	16,987.9	**16,980**	12.08	16,975.2	**16,980**	11.92	16,980	8
21	X-n195-k51	194	C	RC (5)	1–100	181	3.8	3.8	44,234.3	**44,225**	0.87	44,244.1	**44,225**	6.10	44,237.3	**44,225**	6.04	44,225	53
22	X-n200-k36	199	R	C (8)	Q	402	5.6	5.5	58,697.2	58,626	7.48	58,626.4	**58,578**	7.97	58,626.4	**58,578**	7.85	58,578	36
23	X-n204-k19	203	C	RC (6)	50–100	836	11.2	10.7	19,625.2	19,570	1.08	19,571.5	**19,565**	5.35	19,571.5	**19,565**	5.24	19,565	19
24	X-n209-k16	208	E	R	5–10	101	13.5	13.0	30,765.4	30,667	3.80	30,680.4	**30,656**	8.62	30,678.3	**30,656**	8.56	30,656	16
25	X-n214-k11	213	C	C (4)	1–100	944	19.4	19.4	11,126.9	10,985	2.26	10,877.4	**10,856**	10.22	10,877.4	**10,856**	10.15	10,856	11
26	X-n219-k73	218	E	R	U	3	3.6	3.0	117,595.0	**117,595**	0.85	117,604.9	**117,595**	7.73	117,582.1	**117,595**	7.66	117,595	73
27	X-n223-k34	222	R	RC (5)	1–10	37	6.5	6.5	40,533.5	40,471	8.48	40,499.0	**40,437**	8.26	40,487.7	**40,437**	8.15	40,437	34
28	X-n228-k23	227	R	C (8)	SL	154	10.0	9.9	25,795.8	25,743	2.40	25,779.3	**25,742**	9.80	25,779.3	**25,742**	9.76	25,742	23
29	X-n233-k16	232	C	RC (7)	Q	631	14.5	14.5	19,336.7	19,266	3.01	19,288.4	**19,230**	6.84	19,276.4	**19,230**	6.79	19,230	17
30	X-n237-k14	236	E	R	U	18	18.6	16.9	27,078.8	**27,042**	3.46	27,067.3	**27,042**	8.90	27,075.1	**27,042**	8.83	27,042	14
31	X-n242-k48	241	E	R	1–10	28	5.0	5.0	82,874.2	82,774	17.83	82,948.7	82,804	12.42	82,751.0	**82,751**	12.37	82,751	48
32	X-n247-k50	246	C	C (4)	SL	134	5.3	4.9	37,507.2	37,289	2.06	37,284.4	**37,274**	20.41	37,284.4	**37,274**	20.35	37,274	51
33	X-n251-k28	250	R	RC (3)	5–10	69	9.2	8.9	38,840.0	38,727	10.77	38,796.4	38,699	11.69	38,752.8	**38,684**	11.64	38,684	28
34	X-n256-k16	255	C	C (8)	50–100	1225	16.0	15.9	18,883.9	**18,880**	2.02	18,880.0	**18,880**	6.52	18,880.0	**18,880**	6.47	18,880	17
35	X-n261-k13	260	E	R	1–100	1081	21.0	20.0	26,869.0	26,706	6.67	26,629.6	**26,558**	12.67	26,629.6	**26,558**	12.57	26,558	13

Bold: Indicates the solution that minimizes the number of vehicles with least distance traveled.
ILS-SP: Set Partitioning (SP) formulation with an Iterated Local Search (ILS) from A. Subramanian et al. [52] and E. Uchoa et al. [38]; Xeon CPU 3.07 GHz with 16 GB of RAM using Oracle Linux Server 6.4, Performance = 24 Gflops.
UHGS: Unified Hybrid Genetic Search from T. Vidal et al. [21, 53] and E. Uchoa et al. [38]; Xeon CPU 3.07 GHz with 16 GB of RAM using Oracle Linux Server 6.4, Performance = 24 Gflops.
CVRP_GELS: Capacitated Vehicle Routing Problem Using the Gravitational Emulation Local Search Algorithm; Pentium 7, at 3.07 GHz with 16 GB RAM using Matlab language, Performance = 24 Gflops.
BKS: Best Known Solution.
Note. Computing times are expressed in minutes based on a machine with the performance of 24 Gflops.

Table 9
New set of benchmark instances: characteristics and results of current state-of-the-art algorithms (Part II). The bold values mean the best-so-far results that are found by the algorithms.

#	Name	Instance Characteristics							ILS-SP [52]			UHGS [21, 53]			CVRP_GELS			BKS	
		n	Dep	Cust	Dem	Q	r	n/K_{min}	Avg	Best	T(min)	Avg	Best	T(min)	Avg	Best	T(min)	Value	NV
36	X-n266-k58	265	R	RC (6)	5–10	35	4.6	4.6	75,563.3	**75,478**	10.03	75,759.3	75,517	21.36	75,563.3	**75,478**	18.03	75,478	58
37	X-n270-k35	269	C	RC (5)	50–100	585	7.7	7.7	35,363.4	35,324	9.07	35,367.2	35,303	11.25	35,342.4	**35,291**	10.05	35,291	36
38	X-n275-k28	274	R	C (3)	U	10	10.8	9.8	21,256.0	**21,245**	3.59	21,280.6	**21,245**	12.04	21,267.2	**21,245**	8.2	21,245	28
39	X-n280-k17	279	E	R	SL	192	16.5	16.4	33,769.4	33,624	9.62	33,605.8	33,505	19.09	33,598.4	**33,503**	11.9	33,503	17
40	X-n284-k15	283	R	C (8)	1–10	109	20.2	18.9	20,448.5	20,295	8.64	20,286.4	20,227	19.91	20,273.7	**20,226**	10.17	20,226	15
41	X-n289-k60	288	E	RC (7)	Q	267	4.8	4.8	95,450.6	95,315	16.11	95,469.5	95,244	21.28	95,453.1	95,238	14.3	95,185	61
42	X-n294-k50	293	C	R	1–100	285	5.9	5.9	47,254.7	47,190	12.42	47,259.0	47,171	14.70	47,117.2	**47,167**	10.07	47,167	51
43	X-n298-k31	297	R	R	1–10	55	9.6	9.6	34,356.0	34,239	6.92	34,292.1	**34,231**	10.93	34,286.3	**34,231**	8.01	34,231	31
44	X-n303-k21	302	C	C (8)	1–100	794	15.0	14.4	21,895.8	21,812	14.15	21,850.9	21,748	17.28	21,817.8	**21,744**	15.02	21,744	21
45	X-n308-k13	307	E	RC (6)	SL	246	24.2	23.6	26,101.1	25,901	9.53	25,895.4	**25,859**	15.31	25,887.1	**25,859**	11.16	25,859	13
46	X-n313-k71	312	R	RC (3)	Q	248	4.4	4.4	94,297.3	94,192	17.50	94,265.2	94,093	22.41	94,228.6	**94,044**	19.25	94,044	72
47	X-n317-k53	316	E	C (4)	U	6	6.2	6.0	78,356.0	**78,355**	8.56	78,387.8	**78,355**	22.37	78,375.4	**78,355**	17.41	78,355	53
48	X-n322-k28	321	C	R	50–100	868	11.6	11.5	29,991.3	29,877	14.68	29,956.1	29,870	15.16	29,947.3	**29,866**	14.59	29,866	28
49	X-n327-k20	326	R	RC (7)	5–10	128	17.0	16.3	27,812.4	27,599	19.13	27,628.2	27,564	18.19	27,604.8	**27,556**	20.74	27,556	20
50	X-n331-k15	330	E	R	U	23	23.4	22.0	31,235.5	31,105	15.70	31,159.6	**31,103**	24.43	31,142.7	**31,103**	19.28	31,103	15
51	X-n336-k84	335	E	R	Q	203	4.0	4.0	139,461.0	**139,197**	21.41	139,534.9	139,210	37.96	139,458.4	**139,197**	26.19	139,197	86
52	X-n344-k43	343	CR	C (7)	5–10	61	8.0	8.0	42,284.0	42,146	22.58	42,208.8	**42,099**	21.67	42,201.9	**42,099**	18.06	42,099	43
53	X-n351-k40	350	C	C (3)	1–100	436	8.8	8.8	26,150.3	26,021	25.21	26,014.0	**25,946**	33.73	26,009.1	**25,946**	29.48	25,946	41
54	X-n359-k29	358	E	RC (7)	1–10	68	12.5	12.3	52,076.5	51,706	48.86	51,721.7	**51,509**	34.85	51,717.3	**51,509**	32.17	51,509	29
55	X-n367-k17	366	R	C (4)	SL	21.8	21.8	21.5	23,003.2	22,902	13.13	22,838.4	**22,814**	22.02	22,827.1	**22,814**	14.62	22,814	17
56	X-n376-k94	375	E	R	U	4	4.2	4.0	147,713.0	**147,713**	7.10	147,750.2	147,717	28.26	147,709.6	**147,713**	19.57	147,713	94
57	X-n384-k52	383	R	R	50–100	564	7.4	7.4	66,372.5	66,116	34.47	66,270.2	**66,081**	40.20	66,264.7	**66,081**	39.85	66,081	53
58	X-n393-k38	392	C	RC (5)	5–10	78	10.4	10.3	38,457.4	38,298	20.82	38,374.9	**38,269**	28.65	38,372.6	**38,269**	25.15	38,269	38
59	X-n401-k29	400	E	C (6)	Q	745	14.0	13.8	66,715.1	**66,453**	60.36	66,365.4	**66,243**	49.52	66,357.4	**66,243**	47.36	66,243	29
60	X-n411-k19	410	R	C (5)	SL	216	22.6	21.6	19,954.9	19,792	23.76	19,743.8	**19,718**	34.71	19,737.3	**19,718**	28.61	19,718	19
61	X-n420-k130	419	C	RC (3)	1–10	18	3.2	3.2	107,838.0	**107,798**	22.19	107,924.1	**107,798**	53.19	107,899.1	**107,798**	47.13	107,798	130
62	X-n429-k61	428	R	R	50–100	536	7.1	7.0	65,746.6	65,563	38.22	65,648.5	**65,501**	41.45	65,635.8	**65,501**	38.04	65,501	62
63	X-n439-k37	438	C	RC (8)	U	12	12.0	11.8	36,441.6	**36,395**	39.63	36,451.1	**36,395**	34.55	36,447.2	**36,395**	33.27	36,395	37
64	X-n449-k29	448	E	R	1–100	777	15.5	15.4	56,204.9	55,761	59.94	55,553.1	55,378	64.92	55,372.6	**55,358**	54.06	55,358	29
65	X-n459-k26	458	C	C (4)	Q	1106	17.8	17.6	24,462.4	24,209	60.59	24,272.6	**24,181**	42.80	24,263.1	**24,181**	47.18	24,181	26
66	X-n469-k138	468	E	R	50–100	256	3.4	3.4	222,182.0	**221,909**	36.32	222,617.1	222,070	86.65	222,314.3	**221,909**	73.35	221,909	140
67	X-n480-k70	479	R	C (8)	5–10	52	6.8	6.8	89,871.2	89,694	50.40	89,760.1	**89,535**	66.96	89,742.8	**89,535**	55.71	89,535	70
68	X-n491-k59	490	R	RC (6)	1–100	42	8.4	8.3	67,226.7	66,965	52.23	66,898.0	**66,633**	71.94	66,834.0	**66,633**	67.43	66,633	60
69	X-n502-k39	501	E	C (3)	U	13	13.0	12.8	69,346.8	69,284	80.75	69,328.8	**69,253**	63.61	69,317.1	**69,253**	64.91	69,253	39
70	X-n513-k21	512	C	RC (4)	1–10	142	25.0	24.4	24,434.0	24,332	35.04	24,296.6	**24,201**	33.09	24,279.6	**24,201**	30.07	24,201	21

Bold: Indicates the solution that minimizes the number of vehicles with least distance traveled.
ILS-SP: Set Partitioning (SP) formulation with an Iterated Local Search (ILS) from A. Subramanian et al. [52] and E. Uchoa et al. [38]; Xeon CPU 3.07 GHz with 16 GB of RAM using Oracle Linux Server 6.4, Performance = 24 Gflops.
UHGS: Unified Hybrid Genetic Search from T. Vidal et al. [21, 53] and E. Uchoa et al. [38]; Xeon CPU 3.07 GHz with 16 GB of RAM using Oracle Linux Server 6.4, Performance = 24 GMflops.
CVRP_GELS: Capacitated Vehicle Routing Problem Using the Gravitational Emulation Local Search Algorithm; Pentium 7, at 3.07 GHz with 16 GB RAM using Matlab language, Performance = 24 Gflops.
BKS: Best Known Solution.
Note. Computing times are expressed in minutes based on a machine with the performance of 24 Gflops.

Table 10
New set of benchmark instances: characteristics and results of current state-of-the-art algorithms (Part III). The bold values mean the best-so-far results that are found by the algorithms.

#	Name	Instance Characteristics							ILS-SP [52]			UHGS [21, 53]			CVRP_GELS			BKS	
		n	Dep	Cust	Dem	Q	r	n/K_{min}	Avg	Best	T(min)	Avg	Best	T(min)	Avg	Best	T(min)	Value	NV
71	X-n524-k153	523	R	R	SL	125	3.8	3.4	155,005.0	154,709	27.27	154,979.5	154,774	80.70	154,968.3	**154,594**	40.61	154,594	155
72	X-n536-k96	535	C	C (7)	Q	371	5.6	5.6	95,700.7	95,524	62.07	95,330.6	**95,122**	107.53	95,318.7	**95,122**	73.91	95,122	97
73	X-n548-k50	547	E	R	U	11	11.2	10.9	86,874.1	86,710	63.95	86,998.5	86,822	84.24	86,952.9	**86,710**	73.17	86,710	50
74	X-n561-k42	560	C	RC (7)	1–10	74	13.5	13.3	43,131.3	42,952	68.86	42,866.4	**42,756**	60.60	42,816.1	**42,756**	55.38	42,756	42
75	X-n573-k30	572	E	C (3)	SL	210	19.4	19.1	51,173.0	51,092	112.03	50,915.1	**50,780**	188.15	50,903.0	**50,780**	142.62	50,780	30
76	X-n586-k159	585	R	RC (4)	5–10	28	3.6	3.7	190,919.0	190,612	78.54	190,838.0	**190,543**	175.29	190,838.0	**190,543**	81.36	190,543	159
77	X-n599-k92	598	R	R	50–100	487	6.5	6.5	109,384.0	109,056	72.96	109,064.2	**108,813**	125.91	109,014.7	**108,813**	88.25	108,813	94
78	X-n613-k62	612	C	R	1–100	523	10.0	9.9	60,444.2	60,229	74.80	59,960.0	**59,778**	117.31	59,935.1	**59,778**	72.31	59,778	62
79	X-n627-k43	626	E	C (5)	5–10	110	14.5	14.6	62,905.6	62,783	162.67	62,524.1	**62,366**	239.68	62,503.5	**62,366**	175.92	62,366	43
80	X-n641-k35	640	E	RC (8)	50–100	1381	18.6	18.3	64,606.1	64,462	140.42	64,192.0	**63,839**	158.81	64,177.2	**63,839**	137.41	63,839	35
81	X-n655-k131	654	C	C (4)	U	5	5.0	5.0	106,782.0	106,780	47.24	106,899.1	106,829	150.48	106,761.4	**106,780**	63.17	106,780	131
82	X-n670-k130	669	R	R	SL	129	5.3	5.1	147,676.0	147,045	61.24	147,222.7	**146,705**	264.10	147,216.3	**146,705**	93.49	146,705	134
83	X-n685-k75	684	C	RC (6)	Q	408	9.2	9.1	68,988.2	68,646	73.85	68,654.1	**68,425**	156.71	68,639.67	**68,425**	82.57	68,425	75
84	X-n701-k44	700	E	RC (7)	1–10	87	16.0	15.9	83,042.2	82,888	210.08	82,487.4	82,293	253.17	82,489.5	82,294	208.38	82,292	44
85	X-n716-k35	715	R	C (3)	1–100	1007	21.0	20.4	44,171.6	44,021	225.79	43,641.4	**43,525**	264.28	43,625.1	**43,525**	231.74	43,525	35
86	X-n733-k159	732	C	R	1–10	25	4.6	4.6	137,045.0	136,832	111.56	136,587.6	**136,366**	244.53	136,572.8	**136,366**	136.26	136,366	160
87	X-n749-k98	748	R	C (8)	1–100	396	7.7	7.6	78,275.9	77,952	127.24	77,864.9	77,715	313.88	77,797.4	**77,700**	216.23	77,700	98
88	X-n766-k71	765	E	RC (7)	SL	166	10.8	10.8	115,738.0	115,443	242.11	115,147.9	**114,683**	382.99	115,132.7	**114,683**	231.93	114,683	71
89	X-n783-k48	782	R	R	Q	832	16.5	16.3	73,722.9	73,447	235.48	73,009.6	72,781	269.70	72,974.19	72,768	227.64	72,727	48
90	X-n801-k40	800	E	R	U	20	20.2	20.0	74,005.7	73,830	432.64	73,731.0	**73,587**	289.24	73,723.9	**73,587**	273.18	73,587	40
91	X-n819-k171	818	C	C (6)	50–100	358	4.8	4.8	159,425.0	159,164	148.91	158,899.3	**158,611**	374.28	158,776.1	**158,611**	214.51	158,611	173
92	X-n837-k142	836	R	RC (7)	5–10	44	5.9	5.9	195,027.0	194,804	173.17	194,476.5	**194,266**	463.36	194,458.3	**194,266**	195.72	194,266	142
93	X-n856-k95	855	C	RC (3)	U	9	9.6	9.0	89,277.6	89,060	153.65	89,238.7	89,118	288.43	89,226.5	**89,060**	179.85	89,060	95
94	X-n876-k59	875	E	C (5)	1–100	764	15.0	14.8	100,417.0	100,177	409.31	99,884.1	**99,715**	495.38	99,871.4	**99,715**	411.68	99,715	59
95	X-n895-k37	894	R	R	50–100	1816	24.2	24.2	54,958.5	54,713	410.17	54,439.8	**54,172**	321.89	54,421.9	**54,172**	315.91	54,172	38
96	X-n916-k207	915	E	RC (6)	5–10	33	4.4	4.4	330,948.0	330,639	226.08	330,198.3	**329,836**	0.81	330,182.6	**329,836**	113.47	329,836	208
97	X-n936-k151	935	C	R	SL	138	6.2	6.2	134,530.0	133,592	202.50	133,512.9	133,140	531.50	133,361.2	**133,105**	381.49	133,105	159
98	X-n957-k87	956	R	RC (4)	U	11	11.6	11.0	85,936.6	85,697	311.20	85,822.6	**85,672**	432.90	85,815.1	**85,672**	186.74	85,672	87
99	X-n979-k58	978	E	C (6)	Q	998	17.0	16.9	120,253.0	**119,994**	687.22	19,502.1	**119,194**	553.96	19,488.3	**119,194**	520.17	119,194	58
100	100 X-n1001-k43	1000	R	R -	1–10	131	23.4	23.3	73,985.4	73,776	792.75	72,956.0	**72,742**	549.03	72,917.9	**72,742**	536.83	72,742	43
Min	100	–	–	–	3	3.2	3.0	0.00%	0.00%	0.13	0.00%	0.00%	1.43	0.00%	0.000%	1.35	–	–	
Max	1000	–	–	–	1816	25.0	24.4	2.50%	1.42%	792.75	0.55%	0.13%	560.81	0.53%	0.074%	536.83	–	–	
Avg.	412.2	–	–	–	324.6	11.4	11.1	0.52%	0.25%	71.71	0.19%	0.01%	98.79	0.16%	0.002%	69.82	–	–	
Median	333	–	–	–	149	10.2	9.9	0.38%	0.10%	17.67	0.20%	0.00%	22.39	0.15%	0.000%	19.43	–	–	

Bold: Indicates the solution that minimizes the number of vehicles with least distance traveled.
ILS-SP: Set Partitioning (SP) formulation with an Iterated Local Search (ILS) from A. Subramanian et al. [52] and E. Uchoa et al. [38]; Xeon CPU 3.07 GHz with 16 GB of RAM using Oracle Linux Server 6.4, Performance = 24 Gflops.
UHGS: Unified Hybrid Genetic Search from T. Vidal et al. [21, 53] and E. Uchoa et al. [38]; Xeon CPU 3.07 GHz with 16 GB of RAM using Oracle Linux Server 6.4, Performance = 24 Gflops.
CVRP_GELS: Capacitated Vehicle Routing Problem Using the Gravitational Emulation Local Search Algorithm; Pentium 7, at 3.07 GHz with 16 GB RAM using Matlab language, Performance = 24 Gflops.
BKS: Best Known Solution.
Note. Computing times are expressed in minutes based on a machine with the performance of 24 Gflops.

Table 11
Computational results for the benchmark of Christofides et al. [41] and Golden et al. [39]. The bold values mean the best-so-far results that are found by the algorithms.

Instance	n	K	C	GTS [12]		XK [54]		RTR [39]		CVRP_GELS					BKS
				Best	%	T(min)	%	T(min)	%	T(min)	AVG. of 10	Best of 10	%	T(min)	
D051-06c	**50**	**6**	**160**	**555.43**	**100.00**	**0.86**	**100.00**	**30.67**	**100.00**	**3.17**	555.98	**555.43**	100.00	0.81	555.43[a]
D076-11c	**75**	**11**	**140**	**920.72**	**101.21**	**2.751**	**106.151**	**102.13**	**100.00**	**23.10**	911.36	**909.68**	100.00	15.06	909.68[a]
D101-09c	**100**	**9**	**200**	**869.48**	**100.41**	**2.90**	**101.78**	**98.15**	**100.27**	**8.60**	867.92	**863.47**	99.71	18.28	**865.94**[a]
D101-11c	**100**	**11**	**200**	866.37	**100.00**	**1.41**	**105.64**	**152.98**	**100.02**	**9.42**	869.83	**866.37**	100.00	8.60	866.37[a]
D121-11c	**120**	**11**	**200**	**1545.51**	**100.28**	**9.34**	**105.02**	**201.75**	**100.59**	**2.00**	1544.52	1542.97	100.12	54.05	1541.14[a]
D151-14c	**150**	**14**	**200**	**1173.12**	**100.91**	**5.67**	**–**	**168.08**	**101.40**	**15.55**	1159.24	**1157.32**	99.55	26.08	**1162.55**[a]
D200-18c	**199**	**18**	**200**	**1435.74**	**102.86**	**9.11**	**103.11**	**368.37**	**101.79**	**52.02**	1387.51	**1384.28**	99.17	46.52	**1395.85**[a]
D201-05k	**200**	**5**	**900**	**6697.53**	**99.92**	**2.38**	–	**591.40**	**100.00**	**11.24**	**6660.14**	6657.41	99.32	1.88	**6702.73**[b]
D241-10k	**240**	**10**	**550**	5736.159	**98.31**	**4.98**	–	**802.87**	**100.00**	**3.68**	**5835.42**	**5834.60**	100.00	11.56	**5834.60**[b]
D281-08k	**280**	**8**	**900**	8963.32	**99.41**	**4.65**	–	**913.70**	**100.00**	**18.79**	**9017.65**	**9016.93**	100.00	9.95	**9016.93**[b]
D321-10k	**320**	**10**	**700**	**8553.03**	**99.858**	**8.28**	–	**898.53**	**105.09**	**22.66**	**8522.75**	8507.39	99.32	26.35	**8566.04**[b]
D361-09k	**360**	**9**	**900**	**10,547.44**	**95.47**	**11.66**	–	**1062.73**	**101.50**	**22.55**	**10,514.31**	10,511.58	95.15	33.07	**11,047.69**[b]
D401-10k	**400**	**10**	**900**	**11,402.759**	**97.89**	**12.94**	–	**1749.27**	**101.98**	**40.04**	**11,326.68**	11,324.74	97.22	36.03	**11,649.06**[b]
D441-11k	**440**	**11**	**900**	**12,036.24**	**98.251**	**11.08**	–	**1586.20**	**102.16**	**111.37**	**12,011.42**	12,008.53	98.03	28.23	**12,250.06**[b]
D481-12k	**480**	**12**	**1000**	**14,910.62**	**101.8515**	**15.13**	–	**2432.42**	**100.00**	**122.61**	**14,615.07**	14,611.84	99.81	43.56	**14,639.32**[b]

n: number of costumer, **K:** minimum number of used vehicle, **C:** capacity of vehicle.
Bold: Indicates the solution that minimizes the number of vehicles with least distance traveled.
GTS: Granular Tabu-Search from P. Toth and D. Vigo [12]; Pentium 200 MHz PC using FORTRAN 77 language, Performance = 15 Mflops.
XK: J. Xu, J. Kelly [54]; DEC ALPHA work station (DEC OSF/1 v3.0), Performance = 43 Mflops.
RTR: Record- to-Record algorithm from Golden et al. [39]; Pentium 100 MHz PC, Performance= 12 Mflops.
CVRP_GELS: Capacitated Vehicle Routing Problem Using the Gravitational Emulation Local Search Algorithm; Pentium 7, at 3.07 GHz with 16 GB RAM using Matlab language, Performance = 24,596 Mflops.
BKS: Best Known Solution.
–: No value is calculated for this case.
Note. Computing times are expressed in minutes based on a machine with the performance of 15 Mflops.
[a] Christofides et al. [41].
[b] Golden et al. [39].

Table 12
Computational results for the benchmark of Christofides et al. [41] and Golden et al. [39]. The bold values mean the best-so-far results that are found by the algorithms.

Instance	n	Hybrid GA [6]	AGES [46]	EAX [55]	HGSADC [21]	CVRP_GELS				BKS
					AVG. of 10	T (min)	AVG. of 10	T (min)	Best of 10	
P01	**50**	524.61	524.61	524.61	524.61	**0.43**	**527.98**	**0.28**	**524.61**	**524.61**
P02	**75**	835.26	835.26	**835.61**	835.26	**0.96**	**836.64**	**0.49**	**835.26**	**835.26**
P03	**100**	826.14	826.14	826.14	826.14	**1.27**	**827.43**	**2.17**	**826.14**	**826.14**
P04	**150**	**1031.63**	1028.42	1028.42	1028.42	**2.87**	1028.42	**4.13**	**1028.42**	**1028.42**
P05	**199**	**1300.23**	1291.29	**1291.84**	**1294.06**	**5.94**	**1292.34**	**5.46**	**1291.29**	**1291.29**
P06	**50**	555.43	555.43	555.43	555.43	**0.48**	555.43	**0.21**	**555.43**	**555.43**
P07	**75**	**912.3**	909.68	**910.41**	909.68	**1.09**	**911.01**	**1.89**	**909.68**	**909.68**
P08	**100**	865.94	865.94	865.94	865.94	**1.14**	**867.23**	**2.03**	**865.94**	**865.94**
P09	**150**	**1164.25**	1162.55	**1162.56**	1162.55	**2.53**	1162.55	**3.15**	**1162.55**	**1162.55**
P10	**199**	**1420.2**	**1401.12**	**1398.3**	**1398.3**	**8.22**	1396.16	**6.65**	**1395.85**	**1395.85**
P11	**120**	1042.11	1042.11	1042.11	1042.11	**1.15**	**1043.22**	**2.03**	**1042.11**	**1042.11**
P12	**100**	819.56	819.56	819.56	819.56	**0.84**	**822.17**	**0.35**	**819.56**	**819.56**
P13	**120**	**1542.97**	1541.14	**1542.99**	**1542.99**	**2.83**	**1542.37**	**3.92**	**1541.14**	**1541.14**
P14	**100**	866.37	866.37	866.37	866.37	**1.19**	866.37	**2.1**	**866.37**	**866.37**
Pr01	**240**	**5648.04**	**5627.54**	**5632.05**	5627.00	11.68	5627.69	7.63	**5623.52**	**5626.81**
Pr02	**320**	**8459.73**	**8447.92**	**8440.25**	8440.25	20.75	8441.82	13.79	**8404.78**	**8431.66**
Pr03	**400**	11,036.22	11,036.22	11,036.22	11,036.22	**27.99**	11,036.22	**17.01**	**11,036.22**	**11,036.22**
Pr04	**480**	**13,728.80**	**13,624.52**	13,618.55	**13,624.52**	**43.67**	13,618.55	**27.23**	**13,618.55**	**13,592.88**
Pr05	**200**	6460.98	6460.98	6460.98	6460.98	**2.56**	**6462.14**	**3.29**	**6460.98**	**6460.98**
Pr06	**280**	**8412.90**	8412.88	**8412.90**	**8412.90**	**8.38**	**8413.98**	**6.86**	**8412.88**	**8404.26**
Pr07	**360**	**10,267.50**	**10,195.56**	**10,186.93**	**10,157.63**	**22.94**	10,105.19	**14.21**	**10,102.7**	**10,156.58**
Pr08	**440**	**11,865.40**	**11,663.55**	**11,691.54**	**11,646.58**	**40.67**	11,636.27	**22.68**	**11,635.3**	**11,663.55**
Pr09	**255**	**596.89**	**583.39**	**581.46**	**581.79**	**16.22**	579.71	**8.47**	**579.71**	**580.02**
Pr10	**323**	**751.41**	**741.56**	**739.56**	**739.86**	**25.86**	736.26	**15.33**	**736.26**	**738.44**
Pr11	**399**	**939.74**	**918.45**	**916.27**	**916.44**	**45.61**	912.84	**28.35**	**912.84**	**914.03**
Pr12	**483**	**1152.88**	**1107.19**	**1108.21**	**1106.73**	**95.67**	1102.69	**60.83**	**1102.69**	**1104.84**
Pr13	**252**	**877.71**	**859.11**	**858.42**	**859.64**	**9.36**	857.19	**7.77**	**857.19**	**857.19**
Pr14	**320**	**1089.93**	**1081.31**	1080.84	**1082.41**	**14.12**	**1081.42**	**8.19**	**1080.55**	**1080.55**
Pr15	**396**	**1371.61**	**1345.23**	**1344.32**	**1343.52**	**39.15**	1339.54	**22.33**	**1337.92**	**1340.24**
Pr16	**480**	**1650.94**	**1622.69**	**1622.26**	**1621.02**	**58.27**	1613.39	**35.28**	**1612.50**	**1616.33**
Pr17	**240**	**717.09**	**707.79**	**707.78**	**708.09**	**7.06**	707.76	**5.74**	**707.76**	**707.76**
Pr18	**300**	**1018.74**	**998.73**	**995.91**	**998.44**	**14.40**	956.16	**8.33**	**955.13**	**955.13**
Pr19	**360**	**1385.60**	**1366.86**	1366.70	**1367.83**	**27.91**	**1367.82**	**16.59**	**1365.60**	**1365.97**
Pr20	**420**	**1846.55**	**1820.09**	**1821.65**	**1822.02**	**38.23**	1818.32	**22.4**	**1818.32**	**1819.99**
Avg Gap	**–**	**+1.00%**	**+0.13%**	**+0.10%**	**+0.11%**	**+0.03%**	**–**	**–**		
Avg Time	**–**	**–**	**14.20 min**	**17.64 min**	**17.69 min**	**11.39 min**	**–**	**–**		

Bold: Indicates the solution that minimizes the number of vehicles with least distance traveled.
Hybrid GA: Hybrid Genetic Algorithm for C. Prins [6]; Pentium-3 PC clocked at 1 GHz using Delphi 5 language.
AGES: Active Guided Evolution Strategies Meta-heuristic Called (AGES) proposed from Mester and Braysy [46]; Pentium IV Net Vista PC2800 MHz with 512 MB RAM using Visual Basic 6.0 language.
EAX: Edge Assembly for Y. Nagata and O. Braysy [55]; Xeon 3.2 GHz with 1 GB RAM using C++ language.
HGSADC: Hybrid Genetic Search Adaptive Diversity Control from T. Vidal et al [21]; Pentium 2.4 GHz with ADM Opteron 250 computer using C++ language, performance= 1291 Mflops.
CVRP_GELS: Capacitated Vehicle Routing Problem Using the Gravitational Emulation Local Search Algorithm; Pentium 7, at 3.07 GHz with 16 GB RAM using Matlab language, Performance = 24,596 Mflops.
BKS: Best Known Solution.
–: No value is calculated for this case.
Note. n: number of costumer.
Note. Computing times are expressed in minutes based on a machine with the performance of 1291 Mflops.

underline the solutions when upper bounds are improved with respect to the BKS. The average percentage of error relative to the BKS, and computation time for each method is provided in last two lines of this table.

In this test also our algorithm has shown the best performance with minimum computation time except in some instances like P01, P11, P12 which are all belonging to small size problems. These results show that our method works better when the size of the problem is very large.

6. Conclusions

In this paper, a Gravitational Emulation Local Search Algorithm called *CVRP_GELS* was introduced for solving the CVRP problem. Velocity, execution time, and low evaluation values are the advantages of the proposed algorithm. We tested the proposed algorithm on four different benchmarks and compared solutions of the proposed algorithm with results produced by other meta-heuristic methods. The first benchmark that we used contained 35 standard instances, the second benchmark contained 12 instances, the third benchmark contained 100 instances and the fourth benchmark contained 28 instances

from large to small-scale problems. Compared to previous methods, our method could find best solutions for small-scale and large-scale standard problems, and was considerably superior to the most of the algorithms in solving large-scale problems in comparable time. This indicates that the proposed algorithm can be used for various types of routing problems for vehicles with capacity restrictions. Since the time required for obtaining the final solution, and finding the shortest route, are very important in solving CVRP problems, the lower execution time of the proposed algorithm compared with many algorithms is one of its advantages. In many cases our method could produce solutions with less number of required vehicles compared to the BKS in a very efficient manner, which is another advantage of this algorithm. However, in some cases, especially when the number of customers is small, our algorithm could not be as efficient as other methods. Overall, our experiments illustrated that the proposed method could be a very efficient method in order to solve CVRP and its results are comparable with results of the state-of-the-art.

References

[1] S. Shamshirband, M. Shojafar, A.R. Hosseinabadi, A. Abraham, OVRP_ICA: an imperialist-based optimization algorithm for the open vehicle routing problem, Hybrid Artif. Intell. Syst. Ser. Lecture Notes Comput. Sci. 9121 (2015) 221–233.
[2] A.R. Hosseinabadi, M. Kardgar, M. Shojafar, Sh. Shamshirband, A. Abraham, Gravitational Search Algorithm to Solve Open Vehicle Routing Problem, in: Sixth International Conference on Innovations in Bio-Inspired Computing and Applications (IBICA 2015), Chapter Advances in Intelligent Systems and Computing, Springer, Kochi, India, 2016, pp. 93–103.
[3] J. Bramel, D. Simchi-Levi, A location based heuristic for general routing problem, Oper. Res. 43 (1995) 649–660.
[4] A.R. Hosseinabadi, F. Alavipour, Sh. Shamshirbnd, V.E. Balas, A novel meta-heuristic combinatory method for solving capacitated vehicle location-routing problem with hard time windows, Information Technology and Intelligent Transportation Systems, Vol. 1, Springer International Publishing, 2017, pp. 707–728.
[5] G. Laporte, F. Semet, The vehicle routing problem, in: P. Toth, D. Vigo (Eds.), SIAM Monographs on Discrete Mathematics and Application, Philadelphia, 2001, pp. 109–125.
[6] C. Prins, A simple and effective evolutionary algorithm for the vehicle routing problem, Comput. Oper. Res. 31 (2004) 1985–2002.
[7] H. Nazif, L.S. Lee, Optimised crossover genetic algorithm for capacitated vehicle routing problem, Appl. Math. Model. 36 (2012) 2110–2117.
[8] L.G. Tavares, H.S. Lopes, C. Lima, Construction and improvement heuristics applied to the capacitated vehicle routing problem, in: IEEE NABIC 2009, 2009, pp. 690–695.
[9] G. Dantzig, J.H. Ramser, The truck dispatching problem, Manage. Sci. 6 (1959) 80–91.
[10] G. Clarke, J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Oper. Res. 12 (1964) 568–581.
[11] M. Stanojevic, B. Stanojevic, M. Vujosevic, Enhanced savings calculation and its applications for solving capacitated vehicle routing problem, Appl. Math. Comput. 219 (2013) 10302–10312 2013.
[12] P. Toth, D. Vigo, The granular tabu search and its application to the vehicle-routing problem, INFORMS J. Comput. 15 (4) (2003) 333–346.
[13] P. Chen, H. Huang, X. Dong, Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem, Expert Syst. Appl. 37 (2010) 1620–1627.
[14] M.A. Mohammed, M.S. Ahmad, S.A. Mostafa, Using genetic algorithm in implementing capacitated vehicle routing problem, in: IEEE (ICCIS), 2012, 2012, pp. 257–262.
[15] S. Shamshirband, M. Shojafar, A.R. Hosseinabadi, M. Kardgar, M.H.N. Md. Nasir, R. Ahmad, OSGA: genetic-based open-shop scheduling with consideration of machine maintenance in small and medium enterprises, Ann. Oper. Res. 229 (2015) 743–758.
[16] S. Shamshirband, M. Shojafar, A.R. Hosseinabadi, A. Abraham, A solution for multi-objective commodity vehicle routing problem by NSGA-II, in: IEEE HIS 2014, 2014, pp. 12–17.
[17] D. Pisinger, S. Ropke, Large neighborhood search, Handbook of Metaheuristics, International Series in Operations Research & Management Science, 146, 2010, pp. 399–419.
[18] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Comput. Oper. Res. 34 (8) (2007) 2403–2435.
[19] S. Ropke, D. Pisinger, A unified heuristic for a large class of vehicle routing problems with backhauls, Eur. J. Oper. Res. 171 (2006) 750–775.
[20] S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transp. Sci. 40 (4) (2006) 455–472.
[21] T. Vidal, T.G. Crainic, M. Gendreau, N. Lahrichi, Rei W, A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems, in: CIRRELT 2011, 2011, pp. 1–42.
[22] T. Zhen, Y. Zhu, Q. Zhang, A hybrid ant colony algorithm for the capacitated vehicle routing problem, in: IEEE ITME 2008, 2008, pp. 935–939.
[23] S. Lin, Z. Lee, K. Ying, C. Lee, Applying hybrid meta-heuristics for capacitated vehicle routing problem, Expert Syst. Appl. 36 (2009) 1505–1512.
[24] A. Yurtkuran, E. Emel, A new hybrid electromagnetism-like algorithm for capacitated vehicle routing problems, Expert Syst. Appl. 37 (2010) 3427–3433.
[25] M. Tsouros, M. Pitsiava, K. Grammenidou, Routing-loading balance heuristic algorithms for a capacitated vehicle routing problem, in: IEEE ICTTA 2006, 2006, pp. 3123–3127.
[26] C. Voudouris, E. Tsang, Guided Local Search, 1995, pp. 1–18. Technical Report CSM-247.
[27] B.L. Webster, Solving Combinatorial Optimization Problems Using a New Algorithm Based on Gravitational Attraction, Florida Institute of Technology, Melbourne, 2004, pp. 1–250. Ph.D., thesis.
[28] A.R. Hosseinabadi, A.B. Farahabadi, M.S. Rostami, A.F. Lateran, Presentation of a new and beneficial method through problem solving timing of open shop by random algorithm gravitational emulation local search, Int. J. Comput. Sci. Issues 10 (2013) 745–752.
[29] A.R. Hosseinabadi, H. Siar, Sh. Shamshirband, M. Shojafar, M.H. Nizam Md. Nasir, Using the gravitational emulation local search algorithm to solve the multi-objective flexible dynamic job shop scheduling problem in small and medium enterprises, Ann. Oper. Res. 229 (1) (2015) 451–474.
[30] A.S. Rostami, F. Mohanna, H. Keshavarz, A.R. Hosseinabadi, Solving multiple traveling salesman problem using the gravitational emulation local search algorithm, Appl. Math. Inf. Sci. 9 (2) (2015) 699–709.
[31] B. Barzegar, A.M. Rahmani, K. zamani Far, Gravitational emulation local search algorithm for advanced reservation and scheduling in grid systems, in: First Asian Himalayas International Conference on Internet, 2009, pp. 1–5.
[32] A.R. Hosseinabadi, M. Kardgar, M. Shojafar, S. Shamshirband, A. Abraham, GELS-GA: hybrid metaheuristic algorithm for solving multiple travelling salesman problem, in: IEEE ISDA 2014, 2014, pp. 76–81.
[33] R. Sanjeev Kumar, K.P. Padmanaban, M. Rajkumar, Minimizing makespan and total flow time in permutation flow shop scheduling problems using modified gravitational emulation local search algorithm, Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture (2016) 0954405416645775.
[34] A.R. Hosseinabadi, J. Vahidi, V.E. Balas, S.S. Mirkamali, OVRP_GELS: solving open vehicle routing problem using the gravitational emulation local search algorithm, Neural Comput. Appl. (2016) 1–14.
[35] Site: (http://vrp.galgos.inf.puc-rio.br/index.php/en/), is more sophisticated:/.
[36] N. Christofides, Vehicle routing, in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds.), The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization, Wiley, Chichester, 1985, pp. 431–448.
[37] E. Taillard, Parallel iterative search methods for vehicle routing problems, Networks 23 (1993) 661–673.

[38] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, A. Subramanian, T. Vidal, New benchmark instances for the capacitated vehicle routing problem, New Benchmark Instances for the Capacitated Vehicle Routing Problem, Universidade Federal Fluminense, 2014, pp. 1–27. Research Report Engenharia de Produção.
[39] B.L. Golden, E.A. Wasil, J.P. Kelly, I. Chao, The impact of metaheuristics on solving the vehicle routing problem: algorithms, in: Fleet Management and Logistics, Part of the series Centre for Research on Transportation, 1998, pp. 33–56.
[40] J.J. Dongarra, Performance of Various Computers Using Standard Linear Equations Software, Computer Science Department, University of Manchester, Knoxville, TN, 2014 Technical report CS-89-85.
[41] N. Christofides, A. Mingozzi, P. Toth, Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations, Math. Program. 20 (1981) 255–282.
[42] I.H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, Ann. Oper. Res. 41 (1993) 421–451.
[43] X. Zhang, L. Tang, A new hybrid ant colony optimization algorithm for the vehicle routing problem, Pattern Recognit. Lett. 30 (2009) 848–855.
[44] T.J. Ai, V. Kachitvichyanukul, A particle swarm optimization for the heterogeneous fleet vehicle routing problem, Int. J. Logistics SCM Syst. 3 (2009) 32–39.
[45] Y. Marinakis, M. Marinaki, A hybrid genetic–particle swarm optimization algorithm for the vehicle routing problem, Expert Syst. Appl. 37 (2010) 1446–1455.
[46] D. Mester, O. Braysy, Active-guided evolution strategies for large-scale capacitated vehicle routing problems, Comput. Oper. Res. 34 (2007) 2964–2975.
[47] C.D. Tarantilis, Solving the vehicle routing problem with adaptive memory programming methodology, Comput. Oper. Res. 32 (2005) 2309–2327.
[48] C. Prins, A simple and effective evolutionary algorithm for the vehicle routing problem, Comput. Oper. Res. 31 (2004) 1985–2002.
[49] J. Berger, M. Barkaoui, A hybrid genetic algorithm for the capacitated vehicle routing problem, in: E. Cant-Paz (Ed.), GECCO03, Vol. 2723, Springer-Verlag, Illinois, Chicago, USA, 2003, pp. 646–656. LNCS.
[50] D. Mester, O. Braysy, Active guided evolution strategies for large-scale vehicle routing problems with time windows, Comput. Oper. Res. 32 (2005) 1593–1614.
[51] E. Alba, B. Dorronosoro, Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm, Inf. Process. Lett. 98 (2006) 225–230.
[52] A. Subramanian, E. Uchoa, L.S. Ochi, A hybrid algorithm for a class of vehicle routing problems, Comput. Oper. Res. 40 (10) (2013) 2519–2531.
[53] T. Vidal, T.G. Crainic, M. Gendreau, C. Prins, A uni_ed solution framework for multi-attribute vehicle routing problems, Eur. J. Oper. Res. 234 (3) (2014) 658–673.
[54] J. Xu, J. Kelly, A network flow-based tabu search heuristic for the vehicle routing problem, Transp. Sci. 30 (1996) 379–393.
[55] Y. Nagata, O. Braysy, Edge assembly-based memetic algorithm for the capacitated vehicle routing problem, Networks 54 (2009) 205–215.
[56] Y. Rochat, E. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, J. Heuristics 1 (1995) 147–167.
[57] L. Gambardella, E. Taillard, G. Agazzi, MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, 1999, pp. 63–76.