Applied Soft Computing 12 (2012) 1426–1439

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

A comparative study between dynamic adapted PSO and VNS for the vehicle routing problem with dynamic requests

Mostepha R. Khouadjia [a,**,1], Briseida Sarasola [b,*], Enrique Alba [b], Laetitia Jourdan [a], El-Ghazali Talbi [a]

[a] INRIA Lille Nord-Europe, Parc scientifique de la Haute-Borne, Bâtiment A, 40 Avenue Halley, Park Plaza, 59650 Villeneuve d'Ascq Cedex, France
[b] Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, E.T.S.I. Informática, Campus de Teatinos, 29071 Málaga, Spain

ARTICLE INFO

Article history:
Received 10 November 2010
Received in revised form 20 July 2011
Accepted 26 October 2011
Available online 18 November 2011

Keywords:
Vehicle routing problem (VRP)
Dynamic vehicle routing problem (DVRP)
Degree of dynamism (dod)
Particle swarm optimization (PSO)
Variable neighborhood search (VNS)
Adaptive memory
Performance measures for dynamic problems

ABSTRACT

Combinatorial optimization problems are usually modeled in a static fashion. In this kind of problems, all data are known in advance, i.e. before the optimization process has started. However, in practice, many problems are dynamic, and change while the optimization is in progress. For example, in the dynamic vehicle routing problem (DVRP), new orders arrive when the working day plan is in progress. In this case, routes must be reconfigured dynamically while executing the current simulation. The DVRP is an extension of a conventional routing problem, its main interest being the connection to many real word applications (repair services, courier mail services, dial-a-ride services, etc.). In this article, a DVRP is examined, and solving methods based on particle swarm optimization and variable neighborhood search paradigms are proposed. The performance of both approaches is evaluated using a new set of benchmarks that we introduce here as well as existing benchmarks in the literature. Finally, we measure the behavior of both methods in terms of dynamic adaptation.

1. Introduction

Thanks to recent advances in information and communication technologies, vehicle fleets can now be managed in real-time. When jointly used, devices like geographic information systems (GIS), global positioning systems (GPS), traffic flow sensors and cellular telephones are able to provide real-time data, such as current vehicle locations, new customer requests, and periodic estimates of road travel times. If suitably processed, this large amount of data can be used to reduce the cost and improve the service level of a modern company. To this end, revised routes have to be timely generated as soon as new events occur [1]. In this context, dynamic vehicle routing problems (DVRPs) are getting increasingly important [2–5].

The VRP [6] is a well-known combinatorial problem which consists in designing routes for a fleet of capacitated vehicles that are to service a set of geographically dispersed points (customers, stores, schools, cities, warehouses, etc.) at the least cost (distance, time, or any other desired factor). It is possible to define several dynamic features which introduce dynamism in the classical VRP: roads between two customers could be blocked off, customers could modify their orders, the travel time for some routes could be increased due to bad weather conditions, etc. This implies that Dynamic VRPs constitute in fact a set of different problems, which are of crucial importance in today's industry, accounting for a significant portion of many distribution and transportation systems.

The main goal of this work is to present a comparative study between particle swarm optimization (PSO) and variable neighborhood search (VNS) for the resolution of the DVRP with dynamic requests; in this case, the terms "dynamic requests" refer to the fact that some customers are unknown when the optimization process begin, i.e. their orders and positions will be known only after the vehicles are already in route. These two algorithms have been chosen as representative of the two main classes of metaheuristics: trajectory- and population-based algorithms. PSO has already been pointed out as an appropriate method to tackle dynamic continuous problems [7–9] as well as adapted to solve combinatorial optimization problems like VRP [10], while VNS has proved to be one of the most successful methods for solving different types of static VRP [11,12] and other similar trajectory-based mechanisms have been successfully explored to tackle dynamic VRP [13]. In addition, we also propose a new DVRP benchmark, which is aimed at upgrading

** Corresponding author. Tel.: +33 (0) 359577921.
* Corresponding author.
E-mail addresses: mostepha-redouane.khouadjia@inria.fr, khouadjia@gmail.com (M.R. Khouadjia), briseida@lcc.uma.es (B. Sarasola), eat@lcc.uma.es (E. Alba), laetitia.jourdan@inria.fr (L. Jourdan), talbi@lifl.fr (E.-G. Talbi).
[1] Principal corresponding author.

1568-4946/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2011.10.023

the existing instances in terms of enhancing problem complexity and scalability.

The remainder of this article is organized as follows. Section 2 describes the static and the dynamic VRP and its specific features. The resolution scheme is presented in Section 3, which includes both the event scheduler as well as the description of our algorithms. Computational results are analyzed in Section 4, and finally Section 5 presents conclusions and opens some lines for further research.

2. The dynamic vehicle routing problem

The conventional VRP can be mathematically modeled by using an undirected graph $G=(C, E)$, where C is a vertex set, and E is an edge set. They are expressed as $C=\{c_0, c_1, \ldots, c_n\}$, and $E=\{(c_i, c_j)|c_i, c_j \in C, i<j\}$. Furthermore, a set V of m homogeneous vehicles each with capacity Q originate from a single depot, represented by the vertex c_0 and must service all the customers represented by the set C. The quantity of goods q_i requested by each customer i ($i>1$) is associated with the corresponding vertex. We define on E a non-negative distance, travel time or cost matrix $D=(d_{ij})$ between customers c_i and c_j. The goal is to find a feasible set of tours with the minimum total traveled distance. The VRP thus consists in determining a set of m vehicle routes of minimal total cost, starting and ending at a depot, such that every vertex in C is visited exactly once by one vehicle. The sum of the items associated with the vertexes contained in it never exceeds the corresponding vehicle capacity Q. The capacity means the quantity of items (goods) that the vehicle could carry during its travel. Let be a solution $S=R_1, \ldots, R_m$ a partition of C representing the routes of the vehicles to service all the customers. The cost of a given route $R_j=\{c_0, c_1, \ldots, c_{k+1}\}$, where $c_i \in C$ and $c_0=c_{k+1}$ (denote the depot), is given by:

$$\text{Cost}(R_j)=\sum_{i=0}^{k} d_{i,i+1} \tag{1}$$

and the cost of the problem solution (S) is:

$$F_{\text{VRP}}(S)=\sum_{j=1}^{m}\text{Cost}(R_j) \tag{2}$$

Where the total demand of any route cannot exceed the vehicle capacity:

$$\text{Demand}(R_j)=\sum_{i=1}^{k} q_i \times y_i^j \leqslant Q^j \tag{3}$$

where q_i is the associated quantity of the customer c_i (items to be delivered/picked up), Q^j is the capacity of the vehicle j, and

$$y_i^j=\begin{cases}1, & \text{if } c_i \text{ is served by the vehicle } j\\ 0, & \text{otherwise}\end{cases} \tag{4}$$

We will consider a service time δ_i (time needed to unload/load all goods), required by a vehicle to load the quantity q_i at c_i. It is required that the total duration of any vehicle route (travel plus service times) may not surpass a given bound T, so, a route $R_j=\{c_0, c_1, \ldots, c_{k+1}\}$ is feasible if the vehicle stops exactly once in each customer and the travel time of the route does not exceed a prespecified bound T corresponding to the end of the working day

$$\text{Time}(R_j)=\sum_{i=0}^{k} d_{i,i+1}+\sum_{i=1}^{k}\delta_i \leq T \tag{5}$$

There may exist some restrictions such as the capacity of each vehicle, total traveling distance allowed for each vehicle, time windows to visit the specific customers, and so forth. The basic VRP deals with customers which are known in advance; all other information such as the driving time between the customers and the service times at the customers are also usually known prior to the planning.

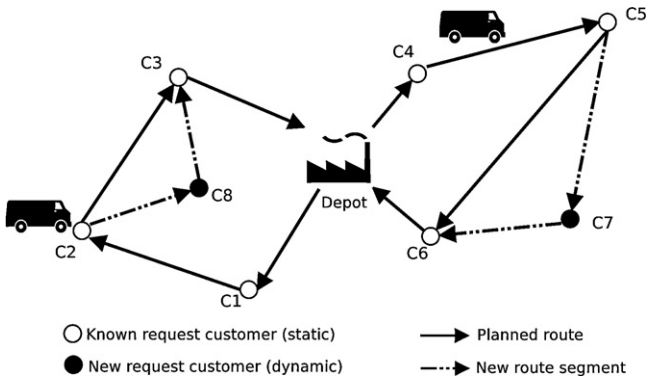


Fig. 1. A dynamic vehicle routing case.

The dynamic vehicle routing problem (DVRP) [5] is strongly related to the static VRP, as it can be described as a routing problem in which information about the problem can change during the optimization process. As conventional static VRPs are NP-hard, DVRP also belongs to the class of NP-hard problems. It is a discrete-time dynamic problem, and can be viewed as a series of P instances; each instance is a static problem, which starts at time t and must be solved within a specific deadline Δ_t. We summarize that as follows:

$$P=\{(P_i, t_i, \Delta_t)/i=0, 1, \ldots, i_{max}\} \tag{6}$$

With this information the duration of the instance i is $t_{i+1}-t_i$. The maximum number of instances i_{max} can be infinite if the problem is open-ended. A new instance P_{i+1} is generated by the action of the environment change ρ_i on the instance i. This is expressed by $P_{i+1}=\rho_i \oplus P_i$. This change in the environment can be due to several factors; for example, travel times can be time [14] or traffic-dependent [16], orders may be withdrawn or changed [17], some clients may be unknown when the execution begins [18], etc. In this work we study this last problem class, which is commonly referred to in the literature as DVRP with dynamic requests, and we follow the model proposed in [19]. In this model, a partial static VRP should be solved each time a new request is received. A simple example of a dynamic vehicle routing situation is shown in Fig. 1. In the example, two un-capacitated vehicles must service both known and new request customers.

Designing a real-time routing algorithm depends to a large extent on how much the problem is dynamic. To quantify this concept, [20,21] have defined the degree of dynamism of a problem (dod). Without loss of generality, we assume that the planning horizon is a given interval $[0, T]$, possibly divided into a finite number of smaller intervals. Let n_s and n_d be the number of static and dynamic requests, respectively. Moreover, let $\tau_i \in [0, T]$ be the occurrence time of service request i. Static requests are such that $\tau_i=0$ while dynamic ones have $\tau_i \in]0, T]$. The degree of dynamism is defined as:

$$dod=\frac{n_d}{n_s+n_d} \tag{7}$$

which may vary between 0 and 1. If it is equal to 0, all requests are known in advance (static problem), while if it is equal to 1, all requests are dynamic (completely dynamic problem).

Background. We can consider different variants of DVRP that could inherit from the classical ones. However, some of them have been really adapted to the dynamic context. As stated before, our

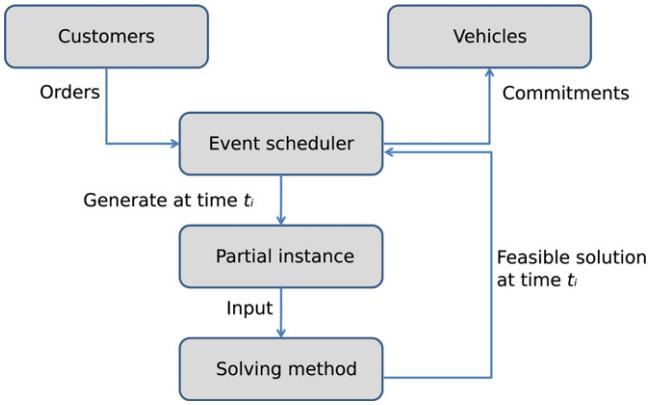


Fig. 2. Resolution framework for the DVRP.

model is based on the capacitated VRP with dynamic requests (CVRPDR) introduced in [19], which was further refined by Montemanni et al. [4] and Hanshar et al. [2]. Other variants add other properties related to traveling time and/or traffic constraints [14,16,22]. They can also try to anticipate the arrival of future demands through statistical information on their availability time in order to improve decision making [23]. One of the most popular variant of the considered problem is the dynamic vehicle routing problem with time windows DVRPTW [24,25,3], inspired by the vehicle routing problem with time window (VRPTW). The DVRP with pick-up and deliveries (DVRPPD) has also been proposed in [26–28]. Other variants introduce additional objective functions, such as the dynamic traveling repair problem (DTRP), where the goal is to minimize the expected time that the demand spends in the system (i.e. the average time a customer must wait before its request is completed), as opposed to the expected distance that the vehicle travels.

3. Resolution framework

The resolution strategy consists of two main components: the event scheduler (which receives new orders, commits vehicles, and constructs partial instances) and the solving algorithm (which finds a feasible solution at each time step). Fig. 2 represents this flowchart.

3.1. Event scheduler

The event scheduler manages the working day simulation and creates partial instances. It serves as an interface between the arrival of new orders and the optimization procedure. Based on the division of the working day into time slices and on the degree of dynamism, the event scheduler creates partial static problems and runs in sequence the solving algorithm on these problems. From the solutions provided by the algorithm, the event manager finally decides about commitments.

The idea of dividing the working day into several discrete time slices was proposed by Montemanni et al. [4]. The goal is to bind the time given to each partial static problem, hence providing an orderly way to service new requests. A different strategy is to include each new customer in the solution as soon as its request is received [24], which may be necessary when urgent requests must be served, as well as when customers have time windows constraints. However, an early inclusion of customers in the solutions does not guarantee that better solutions will be obtained. Concerning this point, Montemanni et al. [4] already showed that splitting the working day in more time slices does not necessarily lead to better results in this problem.

The first partial static problem created for the first time slice (i.e. at the beginning of the working day) consists of all orders remaining from the previous working day. These customers can be considered as being left unattended the day before and are known as static customers. The total number of them is determined using the dod parameter. The next partial problems will consider all orders received during the previous time slice as well as those which have not been committed to drivers yet. In our simulation, each vehicle starts from the location of the last customer committed to it, with a starting time corresponding to the end of the servicing time for this customer, and with a remaining capacity which equals the capacity left after serving all customers previously committed to it.

At the end of each time slice, the solutions found for the corresponding partial problem are examined, and customer orders c_i with a processing time t_i within the next time slice must be committed to their respective vehicles. Note that the t_i is the moment in which c_i should be served according to its position in the route; it is calculated as the departure time from the previous customer c_{i-1} plus the distance between c_{i-1} and c_i. An exception to this commitment strategy is represented by return journeys to the depot, which happens only in two circumstances: when all the customers have been served, or when the vehicle has used all its capacity. In practice, a vehicle will wait at its last committed customer if neither of the two conditions described above are satisfied. In this DVRP model, the commitment cannot be withdrawn, i.e. once an order has been assigned to a driver, the commitment cannot be changed. On the other hand, uncommitted customers can be replanned to another route and/or in another position in the same route. As proposed by Montemanni et al. [4], the pseudo-code of the event scheduler module is presented in Algorithm 1. The working day T is split into n_{ts} time slices, each one with $T_{ts} = T/n_{ts}$ duration. The set *UnServedOrds* initially contains the orders known from the previous day. The variable T_{step} is initialized to 0, while at the beginning of the working day the location of all the vehicles is set at the depot. A partial problem (*PartialProblem*) is created in each loop step and solved

Algorithm 1. Event Scheduler Procedure

T_{step}:=0
// Set each vehicle starting position at the depot and initialize variables
for $v_j \in V$ **do**
  (x_j, y_j):=$(depot_x, depot_y)$
  cap_j:=C
  $dist_j$:=0
  $commit_j$:=*false*
end for
// Add static customers
UnServedOrds:=$\{c_i \mid \tau_i = 0\}$
// Main loop
while *UnServedOrds* $\neq \emptyset$ **do**
  PartialProblem:= buildProblem(*UnServedOrds*, V)
  Run the solving method on *PartialProblem*
  // Commit orders
  CommOrds:=$\{c_i \mid t_i \in [T_{step}, T_{step} + T_{ts}[\}$
  commit(*CommOrds*)
  // Update simulation time
  T_{step}:=$T_{step} + T_{ts}$
  UnServedOrds:=*UnServedOrds* \ *CommOrds*
  // Add orders that appeared in the last time slice
  UnServedOrds:=*UnServedOrds* $\bigcup \{c_i \mid \tau_i \in [(T_{step} - T_{ts}), T_{step}[\}$
  // Update vehicle positions, capacities, and travel times
  for $v_j \in V \wedge v_j$ gives service to R_k **do**
    $commit_j$:=*true*
    updatePosition(x_j, y_j)
    updateCapacity(cap_j)
    updateTravelTime($dist_j$)
  end for
end while
// Send all vehicles back to the depot
for $v_j \in V$ **do**
  (x_j, y_j):=$(depot_x, depot_y)$
end for

with the procedures that will be described in Sections 3.3 and 3.4. The appropriate commitments (*CommOrds*) are done accordingly to the solution of *PartialProblem*. *UnServedOrds* is updated together with starting positions, capacities and travel times of the vehicles. These operations are repeated until $UnServedOrds \neq \emptyset$. When all customers are serviced, the vehicles return to the depot.

In our work, we avoid the problem of detecting environment changes by the partitioning of the working day into time slices, and assuming that the scheduler checks the arrival of new requests at the end of each time slice.

3.2. Solution representation

The representation used in our work is dedicated to dynamic routing problems and both algorithms, PSO and VNS, use the same representation. We propose a simple discrete representation which expresses the route of m vehicles over the n customers to serve. The representation allows the insertion of dynamic customers in the already planned routes. Since the problem is dynamic and customer requests arrive along time, it is necessary to have some kind of knowledge on the state of each customer (served/not served) and its time of service. On the other side, we keep some information about each vehicle. This information is related to its current position in the service region, its remaining capacity, its traveled distance, and its status (committed/not committed). The representation of each route R_k is a permutation of n customers as follows:

$$R_k : (c_0, c_1, c_2, \ldots, c_i, \ldots, c_n, c_{n+1}) \quad (8)$$

For each customer c_i, we assign the following information:

- (x_i, y_i): coordinates of the customer c_i.
- s_i: boolean variable which indicates if the customer c_i has been already served or not.
- t_i: processing time of the customer c_i (time in which the customer is served).

Furthermore, for each route R_k served by the vehicle v_j we keep these information:

- (x_j, y_j): coordinates of the vehicle v_j.
- cap_j: remaining capacity of the vehicle v_j.
- $dist_j$: distance traveled by the vehicle v_j.
- $commit_j$: boolean variable which indicates if the vehicle v_j has been committed or not.

3.3. Dynamic adapted PSO for DVRP

The particle swarm optimization (PSO) algorithm was proposed by J. Kennedy et al. [29,30], follows a collaborative population-based search, which models over the social behavior of bird flocking and fish schooling. A particle is defined by its position $\vec{x}_i$, the position of its personal best solution found so far $\vec{p}_i$, and its velocity $\vec{v}_i$. Furthermore, each particle knows the best solution found so far by any of its neighbors $\vec{p}_g$. Different particle topologies are explored in [30], but the standard neighborhood is the global neighborhood. The algorithm proceeds iteratively, updating first all velocities, and then all particle positions as follows:

$$\vec{v}_i = \omega\vec{v}_i + \varphi_1 \times r_1(\vec{p}_i - \vec{x}_i) + \varphi_2 \times r_2(\vec{p}_g - \vec{x}_i) \quad (9)$$

$$\vec{x}_i = \vec{x}_i + \vec{v}_i \quad (10)$$

The first equation is used to calculate the ith particle's new velocity by taking into consideration three terms: the particle's previous velocity, the distance between the particle's best previous and current position, and finally, the distance between swarm's best experience and ith particle's current position. Then, following the second equation, the ith particle moves towards a new position. The role of the inertia weight ω is to regulate the trade-off between the global and local exploration abilities of the swarm. The parameters φ_1, φ_2 control respectively, the relative attraction to the personal best and global best found solutions. Finally, r_1, r_2 are a random variables drawn with uniform probability from [0, 1].

The algorithm is slightly modified to take into account the dynamic aspect of the problem (arrival of new customers). It has been argued that PSO may be a particularly suitable candidate for this type of problems and, over the past decade, a large number of PSO variants for continuous dynamic optimization problems have been proposed [7–9,31]. In order to have better response to environmental changes, Khouadjia et al. have designed in [32] an adaptive particle swarm optimization (APSO) for solving DVRP. The adaptation of this algorithm consists in the usage of swarm's memory.

3.3.1. Adaptive memory

Some studies produced in the area of dynamic optimization deal with the non-stationary feature by regarding each change as the arrival of a new optimization problem that has to be solved from scratch like [33]. However this simple approach is impractical in some problems like DVRP, because solving a problem from scratch without reusing information from the past is meaningless, due to the fact that vehicles are already on routes and are servicing customers. Better solutions could be achieved by using an optimization algorithm that is capable of continuously adapting the solution to a changing environment, reusing the information gained in the past. A number of authors have addressed the issue of transferring information from the old environment to the new environment by enhancing the algorithms [34,35]. Usually, the approaches are enhanced with some sort of memory that might allow to store good solutions and reuse them later as necessary. This memory may be provided implicitly by using redundant representations, or explicitly by introducing an extra memory and formulating strategies to store in and retrieve solutions from that memory. The latter alternative seems to be more promising.

For the dynamic vehicle routing problem, Rochat and Taillard [15] propose a Tabu Search enhanced with adaptive memory. This latter stores the routes of the best solutions visited during the search. New solutions are then created by combining routes taken from different solutions taken from this memory. New solution produced by the tabu search is included in the memory if the memory is not filled yet, or the new solution is better than the worst solution stored in memory, in which case the latter is removed.

Our approach is closely related to the just mentioned ones. In PSO this explicit memory is intrinsic to the algorithm, since each particle is defined by its current position and its best position so far. In order to have a better response to environment changes, we propose a dynamic adapted particle swarm optimization (DAPSO). The adaptability of the algorithm consists in using the information gathered previously on the search space. When the environment changes, this mechanism allows the algorithm to restart the search from the best solutions found during the previous searches.

For the adjustment to the new environment, our algorithm selects the best positions found so far in the population and repositioning the particles in the search space according to these positions. The result of this mechanism is that the stored candidate solutions will produce outposts at different locations. If the optima return to the same proximity in the search space, the memory points can self-adjust to the translocated optima. After a change in the environment, the current position and the best particle position are re-evaluated and the best solution becomes the active current position. This mechanism is expected to give better response to the dynamic environment.

3.3.2. Velocity vector

The velocity vector $\vec{v}_i$ of the particle is initialized for each new dimension by a random number between [1, m], where m is the number of the planned routes. The normalization of particle velocity in index sequence allows to have feasible solutions. For each dimension of position vector $\vec{x}_i$, if the customer is served, we cannot change neither the tour nor the position of this latter in the already established vehicle routes. The updating process is very similar to the *ejection chain* method that has been applied successfully to vehicle routing [36]. It consists in generating a compound sequence of interrelated simple moves between routes, leading from one solution to another. The customers are moved from their route and inserted into another one according to the cheapest cost insertion. Each customer is placed in the position which minimizes the cost of its insertion into the route.

3.3.3. DAPSO algorithm

Pseudo-code of the DAPSO procedure for solving DVRP is presented in Algorithm 2. The initial population of DAPSO is obtained by generating a random permutation of customers which were left over from the previous working day as defined by the dod parameter. At each time step t_{step}, the current position of each particle is initialized from the solution(s) found in the adaptive memory.

Algorithm 2. Pseudo-code of DAPSO for the DVRP

INPUT: VRP instance that corresponds to the set of customers arrived during the last time slice t_s.
// Reuse the best solutions found by the swarm (Adaptive memory)
for Each particle i **do**
  // Initialize the current position of the particle with its best position found so far if it is better
  if $f(\vec{p}_i) \leq f(\vec{x}_i)$ **then**
    $\vec{x}_i := \vec{p}_i$
  end if
end for
// PSO Main Algorithm
for Each particle i **do**
  // Insert the new customer requests in the current position
  $\vec{x}_i := Insert(\vec{x}_i, New\ customer\ orders)$
  // Initialize the best position
  $\vec{p}_i := \vec{x}_i$
  Evaluate $f(\vec{p}_i)$
end for
repeat
  for Each particle i **do**
    Update velocities:
    $\vec{v}_i := \omega\vec{v}_i + \varphi_1 \times r_1(\vec{p}_i - \vec{x}_i) + \varphi_2 \times r_2(\vec{p}_g - \vec{x}_i)$
    Move to the new position:
    $\vec{x}_i := \vec{x}_i + \vec{v}_i$
    Evaluate $f(\vec{x}_i)$
    // Update personal best
    if $f(\vec{x}_i) \leq f(\vec{p}_i)$ **then**
      $\vec{p}_i := \vec{x}_i$
    end if
    // Update global best
    if $f(\vec{x}_i) \leq f(\vec{p_g})$ **then**
      $\vec{p}_g := (\vec{x}_i)$
    end if
  end for
until termination criterion reached

3.4. VNS for DVRP

Variable neighborhood search (VNS) is a well-known trajectory-based metaheuristic algorithm proposed by Hansen and Mladenović [37]. Trajectory methods are characterized by the usage of one single solution at each time, as opposed to population-based metaheuristics, which manage a set of solutions at each time step. VNS is based on the principle of systematically changing neighborhoods in order to escape from local optima.

VNS consists in defining a succession of neighborhood structures $\mathcal{N}_1(s), \mathcal{N}_2(s), \ldots, \mathcal{N}_n(s)$, which define neighborhoods around a solution s in the search space. At each iteration, another solution s' is picked up from the current neighborhood $\mathcal{N}_k(s)$ and it is improved via the local search routine; this provides a new local minimum s''. If a better local optimum has been found, i.e. $f(s'') \leq f(s)$, s'' is chosen as the new current solution and the search continues in $\mathcal{N}_1(s'')$; otherwise, the search moves in the next iteration to the next neighborhood $\mathcal{N}_{k+1}$.

In order to adapt VNS for a particular problem, it is necessary to define the set of neighborhood structures and to establish the local search procedure which is applied to the solutions. Both our neighborhoods and the local search are related to move operators specific to the VRP. We propose four different neighborhoods $\mathcal{N}_k(s)$ for our canonical VNS algorithm. The neighborhoods are defined as follows:

1. $\mathcal{N}_1(s)$ is the set of solutions which results of swapping any 2 customers in the solution s.
2. $\mathcal{N}_2(s)$ is the set of solutions which results of λ-exchange [38] operator with (1, 0) and (1, 1) moves. It results in customers either being shifted from one route to another for the (1, 0) move, or being exchanged between routes for the (1, 1) move. The insertion of a customer is done using the cheapest cost insertion (i.e. the position that minimizes the cost of the insertion).
3. $\mathcal{N}_3(s)$ is the set of solutions which results of applying 2-Opt [39] to any subroute of the solution s.
4. $\mathcal{N}_4(s)$ is the set of solutions which results of using 2-Opt* [40] in any two subroutes of the solution s.

It is important to notice that our neighborhoods allow the algorithm to escape from local minima, as constraints are not enforced at this stage; this means that a solution s' picked up from the neighborhood does not need to comply with the capacity and depot working day restrictions. Section 3.5 explains the local search method for the VNS. A repair procedure makes this new solution feasible before its evaluation. This repair procedure is necessary since the neighborhood operators can generate unfeasible solutions.

Initial solutions are generated using the Savings algorithm [41]. In order to avoid determinism in the construction of initial solutions, we use a parameter γ to calculate the savings as $s(i, j) = d(0, i) + d(0, j) - \gamma\, d(i, j)$, where $\gamma \sim [0, 1]$ [42]. The same strategy is followed to insert dynamic customers in the solution: a partial solution including only new customers is built using the Savings algorithm and these routes are added to the current solution.

The VNS algorithm as applied to VRP is given in Algorithm 3.

Algorithm 3. VNS for the DVRP

INPUT: VRP instance which corresponds to the set of customers who are known at T_s
if $T_s < 0$ **then** {no dynamic orders appeared yet}
  s:=buildSavingsInitialSolution()
else {potential dynamic orders waiting to be scheduled }
  s:= getLastTimeSliceSolution()
  s':= buildSavingsPartialSolution(*New customer orders*)
  s:= merge(s, s')
end if
while termination conditions not met **do**
  k:=1
  while $k < k_{max}$ **do**
    // Select one solution from the current neighborhood
    s':= pickAtRandom($N_k(s)$)
    // Apply local search procedures
    for all local search heuristic ls_j **do** {See Section 3.5 for further details}
      s'':= apply(ls_j, s'')
    end for

Algorithm 3. (Continued.)

```
    s'':= repair(s'')
    // Update current solution and/or neighborhood
    if f(s'') ≤ f(s) then
      s:=s''
      k:=1
    else {n is the number of neighborhoods}
      k:=(k + 1)mod n
    end if
  end while
end while
```

3.5. Hybridizing DAPSO and VNS with 2-Opt and 2-Opt*

Over the last years, interest in hybrid metaheuristics has risen considerably among the combinatorial optimization research community, as best results for academic and industrial problems are usually obtained by hybrid algorithms. We use a low-level relay hybridization schema (LRH) for both algorithms. In [43], Talbi describes this class of hybrids as algorithms in which a given metaheuristic is embedded into another one.

We have used the 2-Opt heuristic as a local search for DAPSO. The heuristic is applied after the move of particles. The 2-Opt operator reverses a subroute of a given route R_k by selecting two arcs $a = [c_1, c_2]$ and $b = [c'_1, c'_2]$ and substituting them by $a' = [c_1, c'_2]$ and $b' = [c'_1, c_2]$ (see example in Fig. 3). The already traveled segment of tour is left untouched. The heuristic is operated on each route and only for the segments with unserved customers.

Our VNS model uses a local search strategy which is a result of consecutively combining four local search operators: λ-exchange with (1, 1) moves, λ-exchange with (1, 0) moves, 2-Opt and 2-Opt*. The 2-Opt* heuristic selects two arcs $a = [c_1, c_2] \in R_\alpha$ and $b = [c'_1, c'_2] \in R_\beta$ and constructs two new arcs so that $a = [c_1, c'_2]$ and $b = [c'_1, c_2]$ (see an example in Fig. 4). The motivation here is giving VNS a method which explicitly exchanges subroutes instead of merely single customers; besides, 2-Opt* is a suitable operator for DVRP, since it allows exchanging the remainder of two routes while not affecting already committed customers. For each heuristic, all possible moves are checked and the best one is performed, i.e. the one which reduces the solution cost the most. Each local search is applied with probability P_{LS} equal to 1.

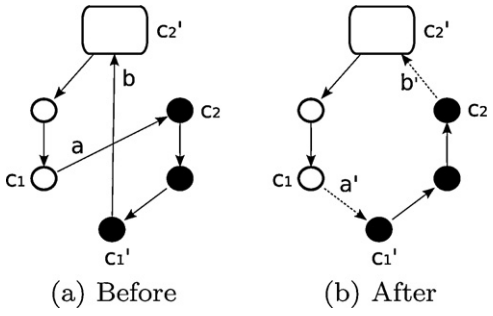


Fig. 3. Example of the 2-Opt heuristic applied to arcs a and b in one single route.

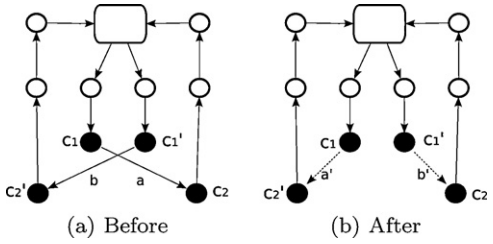


Fig. 4. Example of the 2-Opt* heuristic applied to arcs a and b belonging to two different routes.

The local searches are designed in a manner in which we can avoid the whole route evaluation. An efficient way to evaluate the set of candidates is the evaluation $\Delta(s, m)$ of the objective function, where s is the current solution and m is the applied move. This incremental evaluation consists in evaluating only the transformation (s, m) applied to a solution s rather than the complete evaluation of the neighbor solution $f(s') = f(s \oplus m)$. This is an important issue in terms of efficiency and must be taken into account in the design of high-achieving metaheuristics especially in dynamic optimization context [43].

3.6. Metaheuristics in static and dynamic VRP

Both DAPSO and VNS are designed using state of the art techniques which have been successfully tested in the static VRP. However, achieving good performance in a dynamically changing environment poses new challenges to our algorithms, as new considerations should be taken into account. First, convergence times are much more significant in the dynamic case; if two algorithms reach solutions of similar quality in a static environment, the algorithm which converges fast is more likely to achieve good solutions if the available optimization time is limited. By fast convergence times we refer to the algorithm being able of finding a good solution in a quick manner, and by no means to premature population convergence in an evolutionary algorithm. Second, execution time is another important issue here, since vehicles wait and depend on the algorithm results in order to take decisions on changing their routes. Last but not least, final solutions are highly influenced by decisions which were taken early in the optimization process (commitment of vehicles to routes and customers); this means early choices are essential to latter solution quality. Therefore, the performance of algorithms in the dynamic VRP cannot be directly inferred from the static version: a new focus and further studies are needed for a proper evaluation of algorithms and solutions.

4. Experimental results and discussion

This section is devoted to the experimental evaluation of DAPSO and VNS algorithms. The adopted benchmarks will be described in Section 4.1. Section 4.2 details some algorithms parameters, while, in Section 4.3, the results achieved by the algorithms will be presented.

4.1. Benchmark description

In order to compare our algorithms with other approaches on DVRP [2], a number of parameters need to be set in the classical benchmarks. To standardize the benchmarks,[2] Montemanni et al. [4] fixed some parameters that can affect the final travel distances. The first is n_{ts}, the number of time slices in the optimization process. This parameter subdivided the day into discrete time periods, in which optimization is carried out on each one in succession. Montemanni et al. [4] have tested several values (10, 25, 50) for this parameter on different instances, and found that setting $n_{ts} = 25$ leads to the best trade off between the objective value and computational cost. Montemanni et al. have defined the dynamic and static demands by their available time. They introduce the cut-off time T_{co}, the demands which arrive after $T_{co} \times T$ are postponed to the following day, while those that arrive before this time are considered as dynamics. The T_{co} is set to 0.5. Besides, they consider an

[2] http://www.fernuni-hagen.de/WINF/inhalte/benchmark_data.htm.

advanced commitment time T_{ac} to give the driver an appropriate reaction time after having been committed in the new orders. The T_{ac} is set to 0.

We propose here, for the first time, a large set of new instances for dynamic vehicle routing problems (*k-series*). The goal is to standardize DVRP instances and also to propose new ones which are harder to solve. So, the benchmark data set contains large size instances for DVRP, larger than the standard Kilby's problem set [19] (21 instances, where the number of customers ranges between 50 and 385, although in the literature only instances up to 199 are solved). Another important difference with Montemanni's proposal [4] is the usage of a fixed number of evaluations (further details in Section 4.2), while he used CPU time as termination condition. CPU time is highly dependent on hardware, and not a suitable standard to compare with existent algorithms. Finally, we use a different way to measure the degree of dynamism on the system: Montemanni proposes splitting the working day in two halves of equal length, whereas we define dod depending on the proportion of unknown customers. The *k-series* instances are generated via a VRP-generator.[3] These instances are *k100*, *k250*, and *k500*, where the number of customers can be inferred from its name. For example, f71 corresponds to Fisher's instance with 70 customers to serve and one single depot. As *k-series* has different variants, we choose to indicate if the instance is with a single or multi-depots. In this work, we deal with single depot instances. Thus, the instance *k100* means that there are 100 customers to serve and one single depot. Each instance also contains the following data:

- Length of the working day: we will refer to this parameter as *T*.
- Appearance time of each request: it contains, for each request, the moment of the working day when the order becomes known by the dispatcher.
- Duration of each request: it represents, for each request, the time required to serve the corresponding customer.
- Number of vehicles: it contains the dimension, in number of vehicles, of the available fleet to serve customers. The number of vehicles is set to ensure that it is possible to serve all the orders for the problems considered. The customers arrive with a uniform distribution during the working day.

In this work, we deal with pickup problems. The driver of the vehicle is not concerned with what he is transporting, but only the quantity that he will pick from the customer. we fix the above mentioned parameters that can affect the final travel distances as follows:

- As tested by Montammani et al. [4], we adopt to set n_{ts} to 25. This parameter subdivides the day *T* into 25 discrete time slices.
- For Kilby's instances we set the cutoff time T_{co} to 0.5 in order to determine the known customers, while in *k-series* instances, the degree of dynamism dod is fixed to 0.5; this means that a half of the customers is considered as static, while the other half is dynamic. The optimization begins to plan routes with the known static customers at time $t = 0$.

4.2. Algorithm parameters

We have used the following parameters in our algorithms.

4.2.1. Stopping criterion

The stopping criterion is a fixed number of evaluations for the both *k-series* and Kilby's instances. One evaluation of a solution is either an evaluation of a single particle, or an evaluation of a VNS solution. For *k-series*, we have taken into account the number of necessary evaluations that the algorithms spend in order to reach the optimum and converge in the static case with the entire instance. We think that if the algorithm can solve the entire instance in a fixed number of evaluations, so with the half number of these evaluations it could solve a partial part of it in dynamic context. In Table 1, we give the number of evaluations assigned to each instance in the static and dynamic cases.

Table 1
Number of evaluations assigned to each instance as stopping criterion with $n_{ts} = 25$.

Instances	Static	Dynamic (one time slice)	Dynamic (complete day)
k100	1200	600	$25 \times 600 = 15000$
k250	1200	600	$25 \times 600 = 15000$
k500	2400	1200	$25 \times 1200 = 30000$

For Kilby's instances we have fixed the number of evaluations at 500 evaluations per time slice ($25 \times 500 = 12500$), while Refs. [4,2] use the CPU time as termination condition.

4.2.2. DAPSO parameters

The DAPSO population size is 100. The values of the parameter ω is fixed to 1.0, whereas φ_1, φ_2 take value in the range [0.5, 1].

4.3. Performance assessment

This section presents the results obtained both in the new *k-series* instances and the Kilby benchmark. The DAPSO algorithm was implemented on ParadisEO[4] [43]. The experiments were run on Intel Xeon 3 GHz with 2 GB memory. The VNS algorithm was coded in Java 1.5.0, and runs on Intel Core 2 Quad 2.6 GHz machines with 4 GB memory.

First, we present a study on the *k-series* instances in Section 4.3.1. Next, we compare with state of the art algorithms in Section 4.3.2. We continue with a study on several degrees of dynamism in Section 4.3.3 and finish with an analysis of our results regarding specific measures for dynamic optimization problems in Section 4.3.4.

4.3.1. K-series instances

We have performed 30 independent runs of each experiment. The results are shown in Table 2, which includes the best achieved fitness, the average, the standard deviation, as well as the running time for each instance and each algorithm measured in minutes. Bold entries indicate where the best solution has been obtained.

In order to be able to compare our results accurately, we have also performed statistical significance tests. We use a Kolmogorov–Smirnov test to check whether distributions are normal or not and a Levene test to check the data homocedasticity (homogeneity of variances); if both tests are positive, ANOVA is used, otherwise we perform a Kruskal–Wallis test to compare the medians of the algorithms [44]. As a result, all our experiments have a confidence level of 95% (p-value ≤ 0.05). Table 3 is marked with "+" sign if there are statistical differences between a certain pair of algorithms, and with a "−" sign otherwise.

4.3.1.1. Results on the static instances. In the first place, we study the behavior of algorithms on the static problem, i.e. considering all the customers in the instance as static (dod = 0). VNS behaves significantly better than DAPSO for the three instances, as it finds the best known solution in the three instances. Our statistical study

[3] The VRP generator is available on the following links: http://dolphin.lille.inria.fr, http://neo.lcc.uma.es/dynamic.

[4] http://paradiseo.gforge.inria.fr.

Table 2
Solutions obtained by DAPSO and VNS on static and dynamic instances.

Instance	Algorithm	Solution	Static	Dynamic	Time
k100	DAPSO	Best	1497.70	**1819.01**	
		Average	1563.80	1871.25	6.15
		Std-Dev	30.13	29.55	
	VNS	Best	**1448.18**	1874.37	
		Average	1529.49	2084.47	0.74
		Std-Dev	36.71	102.14	
k250	DAPSO	Best	6038.08	7658.27	
		Average	6722.67	8194.08	41.89
		Std-Dev	277.08	99.82	
	VNS	Best	**5869.38**	**6845.82**	
		Average	6187.80	7251.54	13.23
		Std-Dev	270.88	249.44	
k500	DAPSO	Best	20396.5	26347.8	
		Average	21157.28	27592.34	223.29
		Std-Dev	312.16	383.07	
	VNS	Best	**18582.83**	**24082.73**	
		Average	20108.49	24939.88	130.83
		Std-Dev	1457.51	520.99	

(see Table 3) reflects that there is a statistical difference between DAPSO and VNS for the three instances.

4.3.1.2. Results on the dynamic instances. DAPSO obtains better results than VNS for *k100*, while it is outperformed by VNS in *k250* and *k500*. All these results are statistically significant, as shown in Table 3. An interesting issue is the fact that DAPSO achieves worse results in the dynamic case as well as in the static case for *k250* and *k500*. This is due to the number of evaluations not allowing a convergence of the algorithm (see Table 1 in Section 4.2). Furthermore, DAPSO has a slow evolution, and this affects its performance. However, we should take into account the number of allowed evaluations as a constraint of the changing environment.

When it comes to the instance size, the VNS algorithm outperforms DAPSO in the case of the two bigger instances (*k250* and *k500*). This can be observed in Fig. 5b and c, where VNS is able to converge much quicker and reaches better solutions. The case is the opposite for the instance *k100* (see Figure 5a), in which DAPSO has a good performance from the beginning until the end of the simulation. It must also be noticed that the bound fitness values represented in Figure 5 have been computed running our algorithms on the static instance which results of each time slice. These bounds are not attainable by the dynamic algorithms in any case, but we find them useful as reference values for the behavior of our algorithms.

Table 3
Statistical results of comparing our algorithms with a multiple comparison test.

Dynamism	Instance	Algorithm	Test result	
			DAPSO	VNS
Static	*k100*	DAPSO	–	+
		VNS	+	–
	k250	DAPSO	–	+
		VNS	+	–
	k500	DAPSO	–	+
		VNS	+	–
Dynamic	*k100*	DAPSO	–	+
		VNS	+	–
	k250	DAPSO	–	+
		VNS	+	–
	k500	DAPSO	–	+
		VNS	+	–

4.3.2. Kilby's instances

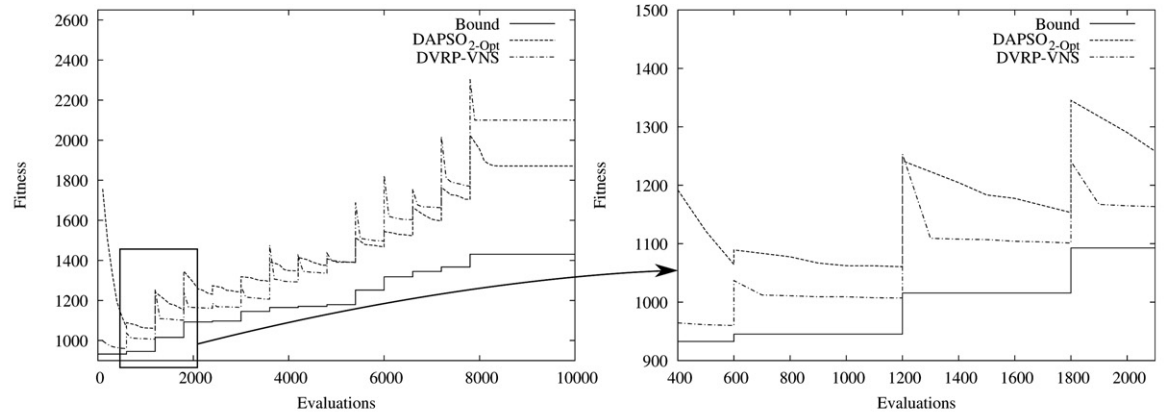
We are also interested in comparing our algorithms with those already proposed in the literature. A comparison of the solution quality in terms of minimizing travel distances is done between DAPSO, VNS, and other metaheuristics as Ant System (*AS*) [4], Genetic Algorithm (GA_{2-Opt}), and Tabu Search (*TS*), both of them proposed in [2]. The benchmark dataset consists of 21 instances, where the number of customers ranges in [50, 199] and the service area may consist of uniformly distributed customers, clustered customers, or a combination of the both (semiclustered instances). Further details about area topologies can be found in Hanshar's work.

Table 4 shows and compares the results obtained by the five algorithms. For each instance, 30 runs of our algorithms have been considered. The best, the average distances, and running time in minutes of the different algorithms are reported. Bolded entries indicate where the best solutions were obtained. We provide in total seven new best solutions on Kilby's instances: DAPSO finds five new best solutions, while VNS gives two new best ones. APSO is able to provide new best solutions in all types of instances regarding their topology: uniform (c75 and c199), cluster (c120) and semi-cluster (f71 and tai75b). The two new best solutions found by VNS can be classified as cluster (c100b) and semicluster (tai100c). While DAPSO obtains more new best solutions, VNS achieves a shorter total traveled distance over the 21 instances. The Genetic Algorithm implemented by Hanshar [2] outperforms the other metaheuristics over 9 instances. AS and TS provide respectively 1 and 4 of the best solutions on Kilby's instances.

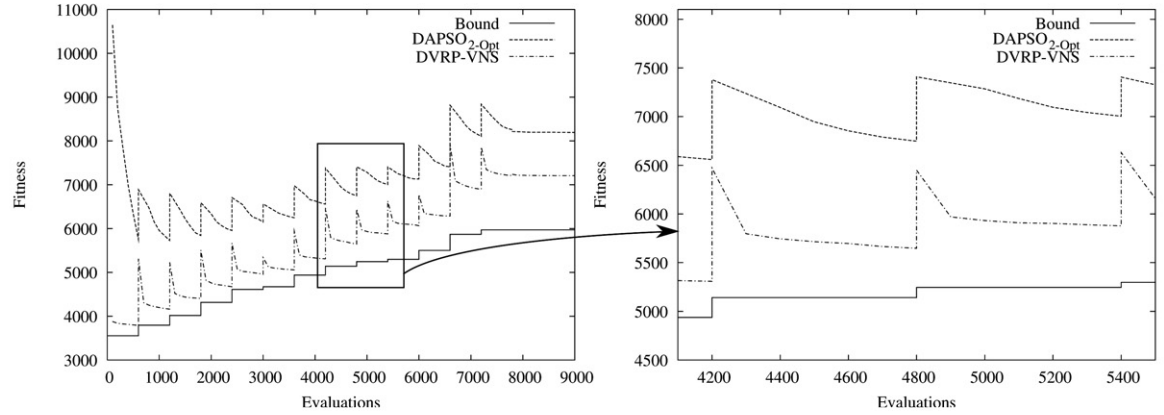
It is also significant to notice that each AS execution lasts 25 min in a Pentium 4 1.5 GHz and each GA and TS execution lasts 12.5 min in a Pentium 4 2.8 GHz, which results in a total execution time of 525 and 262.5 min respectively. These results are improved by both DAPSO (113.07 min) and VNS (9.28 min). These execution times can be normalized according to the processor used in each case. For that purpose, we used a set of benchmarks [45]. When comparing with PSO, AS normalized time is 115.63 min, while GA and TS normalized time is 151.73 min (both of them slower than PSO time 113.07 min). Comparing with VNS, the normalized times are 60.69 min in the case of AS and 73.5 min for GA and TS, which are also improved by our VNS (9.28 min).

4.3.3. Study on the changes of degree of dynamism (dod)

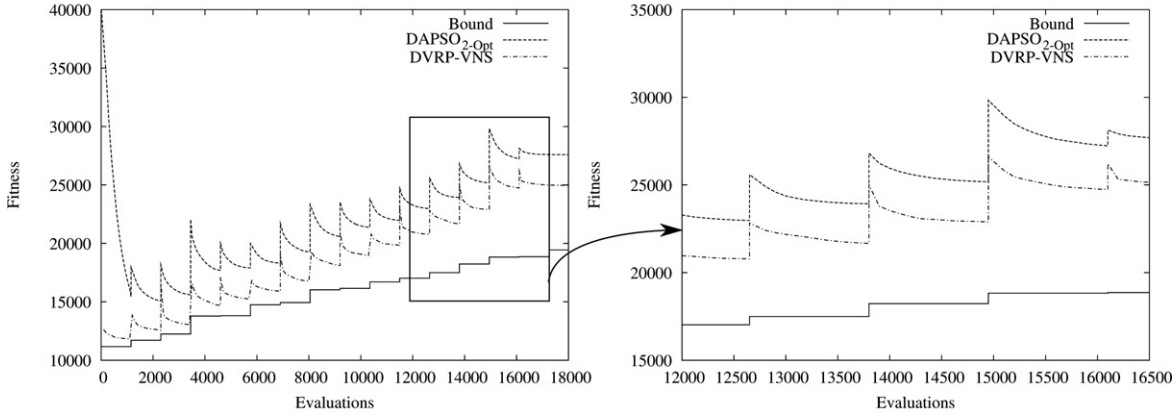
We have performed a study on the behavior of our algorithms in relation to different degrees of dynamism. The dods take their values in range [0.5, 1]. If the dod is 0.5, the



(a) *k100* instance.



(b) *k250* instance.



(c) *k500* instance.

Fig. 5. The evolution of each algorithm mean trace for each instance; each of them shows also the optimum value for each time slice as obtained by running our algorithms over the static subproblems. Each square on the left figure is enlarged in the right figure. (a) *k100* instance. (b) *k250* instance. (c) *k500* instance.

problem is semi-dynamic, while with a dod equal to 1, the problem is completely dynamic. We have done experiments only on the *k-series* instances. The aim is to present the dod effect on the quality of the obtained solutions in term of minimizing the fitness function, and the average of the served customers during the working day. For each instance, 30 runs of DAPSO and VNS are considered. We kept respectively the same benchmark and algorithm parameters as in Sections 4.1 and 4.2.

Table 5 reports the obtained results on the different degrees of dynamism for the DAPSO and VNS algorithms. It indicates the best found solution, the average, and the percentage as well as the range of served customers. When we increase the degree of dynamism, it is easy to see that the percentage of served customers decreases. This is due to the fact that as the problem is bounded by the length of the working day T, the vehicles have to go back to the depot before its closing. In general, customers that arrive at the end of the working day are unserved. For a dod equal to 0.5, results are analyzed in

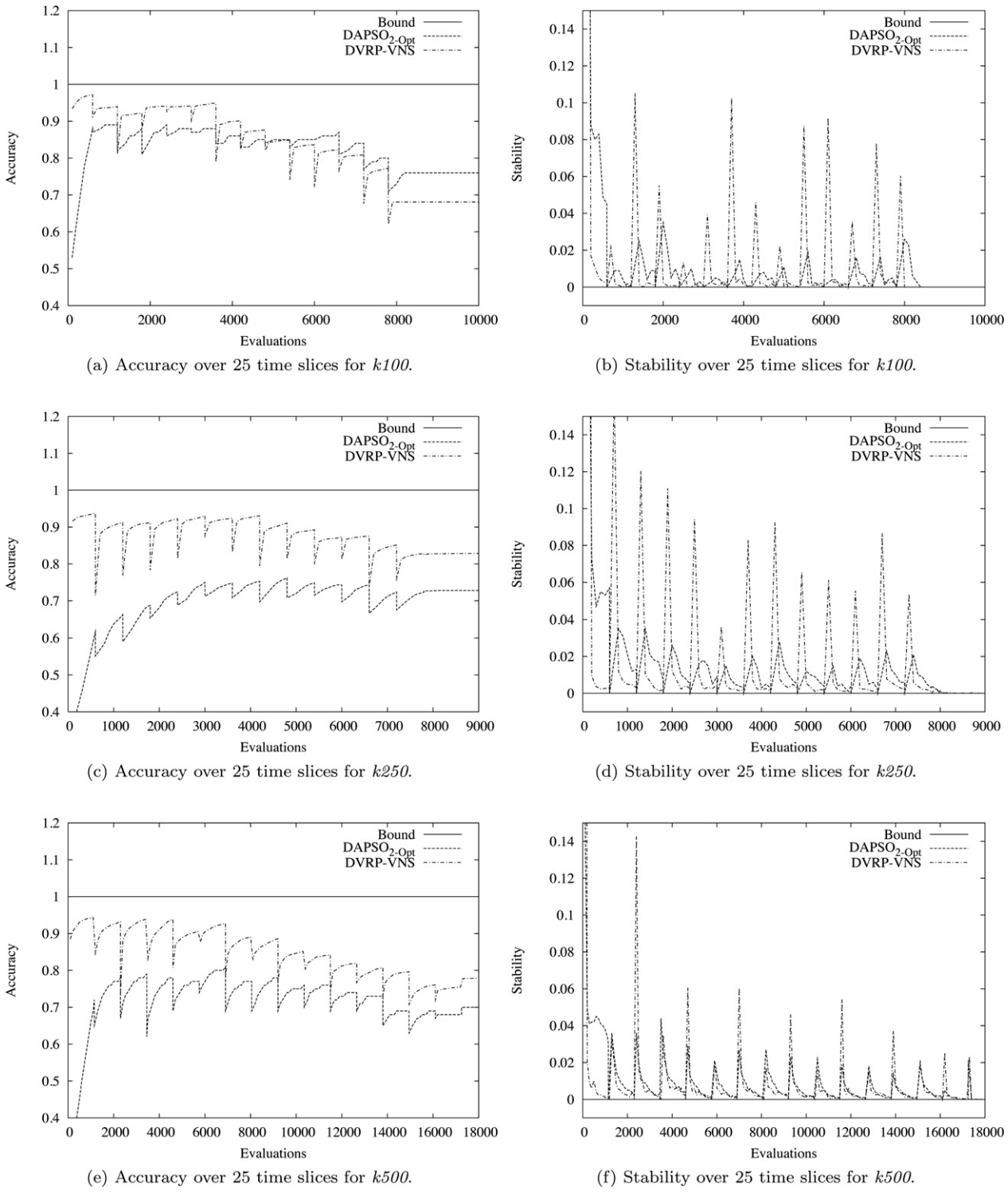
Table 4
Numerical results obtained by DAPSO and *VNS* compared to *AS*, GA_{2-Opt}, and *TS*.

Instances	Metaheuristics											
	DAPSO			*VNS*			*AS*		GA_{2-Opt}		*TS*	
	Best	Average	*Time*	Best	Average	*Time*	Best	Average	Best	Average	Best	Average
c50	575.89	632.38	1.65	599.53	653.84	0.12	631.30	681.86	**570.89**	593.42	603.57	627.90
c75	**970.45**	1031.76	2.49	981.64	1040.00	0.25	1009.36	1042.39	981.57	1013.45	981.51	1013.82
c100	988.27	1051.5	5.81	1022.92	1087.18	0.41	973.26	1066.16	**961.10**	987.59	997.15	1047.60
c100b	924.32	964.47	3.58	**866.71**	942.81	0.30	944.23	1023.60	881.92	900.94	891.42	932.14
c120	**1276.88**	1457.22	6.88	1285.21	1469.24	0.55	1416.45	1525.15	1303.59	1390.58	1331.22	1468.12
c150	1371.08	1470.95	11.74	1334.73	1441.37	0.84	1345.73	1455.50	1348.88	1386.93	**1318.22**	1401.06
c199	**1640.40**	1818.55	17.97	1679.65	1769.95	1.30	1771.04	1844.82	1654.51	1758.51	1750.09	1783.43
f71	**279.52**	312.35	3.45	304.32	325.18	0.27	311.18	358.69	301.79	309.94	280.23	306.33
f134[a]	15875.00	16645.89	3.38	15680.05	16522.18	0.29	**15135.51**	16083.56	15528.81	15986.84	15717.90	16582.04
tai75a	1816.07	1935.28	1.66	1806.81	1954.25	0.22	1843.08	1945.20	1782.91	1856.66	**1778.52**	1883.47
tai75b	**1447.39**	1484.73	1.62	1480.70	1560.71	0.18	1535.43	1704.06	1464.56	1527.77	1461.37	1587.72
tai75c	1481.35	1664.4	2.09	1621.03	1746.07	0.20	1574.98	1653.58	1440.54	1501.91	**1406.27**	1527.72
tai75d	1414.28	1493.47	2.12	1446.50	1541.98	0.19	1472.35	1529.00	**1399.83**	1422.27	1430.83	1453.56
tai100a	2249.84	2370.58	4.59	2250.50	2462.50	0.34	2375.92	2428.38	2232.71	2295.61	**2208.85**	2310.37
tai100b	2238.42	2385.54	4.43	2169.10	2319.72	0.34	2283.97	2347.90	**2147.70**	2215.93	2219.28	2330.52
tai100c	1532.56	1627.32	3.04	**1490.58**	1557.81	0.31	1562.30	1655.91	1541.28	1622.66	1515.10	1604.18
tai100d	1955.06	2123.9	4.42	1969.94	2100.38	0.37	2008.13	2060.72	**1834.60**	1912.43	1881.91	2026.76
tai150a	3400.33	3612.79	9.15	3479.44	3680.35	0.79	3644.78	3840.18	**3328.85**	3501.83	3488.02	3598.69
tai150b	3013.99	3232.11	9.1	2934.86	3089.57	0.73	3166.88	3327.47	**2933.40**	3115.39	3109.23	3215.32
tai150c	2714.34	2875.93	6.46	2674.29	2928.77	0.62	2811.48	3016.14	**2612.68**	2743.55	2666.28	2913.67
tai150d	3025.43	3347.6	6.95	2954.64	3147.38	0.66	3058.87	3203.75	**2950.61**	3045.16	2950.83	3111.43
Total	50190.87	53538.72	113.07	50033.15	53341.24	9.28	50876.23	53794.02	**49202.73**	**51089.37**	49987.8	52725.85

[a] For this instance, the plan of the service area is in scale 10 times larger than the Fisher's instance.

Table 5
Solutions obtained by DAPSO and VNS over different degrees of dynamism. Bold marks the best algorithm regarding the best solution cost, the average solution cost, the number of customers that could be served, and the range of served customers respectively.

Dod	Inst.	Algorithm	Best	Average	Custom.	Range
0.5	k100	DAPSO	**1819.01**	**1871.25**	**100%**	[**100–100**]
		VNS	1950.47	2129.68	**100%**	[**100–100**]
	k250	DAPSO	7658.27	8194.08	**100%**	[**250–250**]
		VNS	**6903.29**	**7221.88**	**100%**	[**250–250**]
	k500	DAPSO	26347.80	27592.34	**100%**	[**500–500**]
		VNS	**24082.73**	**24939.88**	**100%**	[**500–500**]
0.6	k100	DAPSO	**2167.89**	**2295.47**	**100%**	[**100–100**]
		VNS	2313.35	2571.78	99.9%	[99–100]
	k250	DAPSO	8145.35	8706.67	**100%**	[**250–250**]
		VNS	**7361.08**	**7781.98**	**100%**	[**250–250**]
	k500	DAPSO	27535.21	**28761.64**	**99%**	[**495–498**]
		VNS	**27354.50**	28861.12	99.0%	[493–498]
0.7	k100	DAPSO	**2267.38**	**2491.69**	**96.7%**	[**96–98**]
		VNS	2436.85	2680.72	95.5%	[94–96]
	k250	DAPSO	8856.33	**9165.44**	**99%**	[**249–250**]
		VNS	**8239.43**	9244.82	**99%**	[248–250]
	k500	DAPSO	27662.10	28477.72	**97%**	[**483–488**]
		VNS	**26101.17**	**28209.95**	96%	[478–482]
0.8	k100	DAPSO	**2141.93**	**2381.81**	**89.45%**	[89–90]
		VNS	2214.75	2647.02	88.5%	[**87–91**]
	k250	DAPSO	**8221.32**	**8914.49**	**94%**	[**235–237**]
		VNS	8666.36	9365.09	93.8%	[232–236]
	k500	DAPSO	25618.20	**27133.46**	**91%**	[**454–465**]
		VNS	**24327.26**	27185.13	90.14%	[449–451]
0.9	k100	DAPSO	**2070.50**	**2283.56**	79.32%	[79–80]
		VNS	2348.79	2647.33	**79%**	[**77–82**]
	k250	DAPSO	**7944.94**	**8459.35**	**86%**	[**215–218**]
		VNS	8184.28	9098.23	86.1%	[212–218]
	k500	DAPSO	24545.60	**25442.21**	**84%**	[**415–424**]
		VNS	**23242.42**	25640.75	82.41%	[410–413]
1	k100	DAPSO	**2028.51**	**2191.45**	69.29%	[68–71]
		VNS	2289.09	2581.81	**70%**	[**68–74**]
	k250	DAPSO	**7063.96**	**7727.19**	**76%**	[191–193]
		VNS	7567.24	9010.27	76.5%	[**188–194**]
	k500	DAPSO	21485.20	**22546.13**	**74%**	[**357–376**]
		VNS	**22147.62**	23764.54	73.09%	[362–366]



(a) Accuracy over 25 time slices for *k100*.
(b) Stability over 25 time slices for *k100*.
(c) Accuracy over 25 time slices for *k250*.
(d) Stability over 25 time slices for *k250*.
(e) Accuracy over 25 time slices for *k500*.
(f) Stability over 25 time slices for *k500*.

Fig. 6. Evolution of accuracy and stability across time slices for each instance. (a) Accuracy over 25 time slices for *k100*. (b) Stability over 25 time slices for *k100*. (c) Accuracy over 25 time slices for *k250*. (d) Stability over 25 time slices for *k250*. (e) Accuracy over 25 time slices for *k500*. (f) Stability over 25 time slices for *k500*.

Section 4.3.1. From a dod upper than 0.6, the percentage of served customers for DAPSO is better or equal to VNS percentage in all cases, except in the instance *k100* for a dod equal to 1. DAPSO algorithm is able to find solutions which cover more served customers. We can explain this by the diversity of the solutions brought by DAPSO as a population based metaheuristic. At the opposite VNS covers less customers leading to the fact that the traveled distance is lower than that of DAPSO.

4.3.4. Performance measures

The goal of optimization in dynamic environments is not only to find an optimum within a given number of generations, but rather a perpetual adjustment to changing environmental conditions. Besides the accuracy of an approximation at time t_{step}, the stability of the algorithm is also of interest as well as the recovery time to reach again a certain approximation quality. [46] proposes three features for describing the goodness

Table 6
Accuracy of the different metaheuristics on the Kilby's instances. Bold entries indicate the best algorithm regarding the accuracy.

Instance	Accuracy				
	DAPSO	VNS	*AS*	*GA*	*TS*
c50	0.90	0.87	0.83	**0.91**	0.86
c75	**0.86**	0.85	0.82	0.85	0.85
c100	0.83	0.80	0.84	**0.85**	0.82
c100b	0.89	**0.95**	0.87	0.93	0.92
c120	**0.82**	0.81	0.74	0.80	0.78
c150	0.75	0.77	0.76	0.76	**0.78**
c199	**0.79**	0.77	0.73	0.78	0.74
f71	**0.85**	0.78	0.76	0.79	**0.85**
f134	0.73	0.74	**0.77**	0.75	0.74
tai75a	0.89	0.90	0.88	**0.91**	**0.91**
tai75b	**0.93**	0.91	0.88	0.92	0.92
tai75c	0.87	0.80	0.82	0.90	**0.92**
tai75d	0.97	0.94	0.93	**0.98**	0.95
tai100a	0.91	0.91	0.86	0.91	**0.92**
tai100b	0.87	0.89	0.85	**0.90**	0.87
tai100c	0.92	**0.94**	0.90	0.91	0.93
tai100d	0.81	0.80	0.79	**0.86**	0.84
tai150a	0.90	0.88	0.84	**0.92**	0.88
tai150b	0.88	0.91	0.84	**0.91**	0.85
tai150c	0.86	0.88	0.83	**0.90**	0.88
tai150d	0.87	**0.90**	0.86	**0.90**	**0.90**
Average	0.86	0.86	0.83	**0.87**	0.86

of a dynamic adaptation process: accuracy, stability, and ε-reactivity.

The optimization *accuracy* at time t for a fitness function F and optimization algorithm A is defined as

$$\text{accuracy}_{F,A}^{t} = \frac{\text{Min}_F^t}{F(\text{best}_A^{(t)})} \tag{11}$$

where best_t^A is the best candidate solution in the population at time t and Min_F^t the best fitness value in the search space (best known solution). The optimization accuracy ranges between 0 and 1, where accuracy 1 is the best possible value.

As a second goal, stability is an important issue in optimization. In the context of dynamic optimization, an algorithm is called stable if changes in the environment do not affect the optimization accuracy severely. Even in the case of drastic changes an algorithm should be able to limit the respective fitness drop. The stability at time t is defined as

$$\text{stability}_{F,A}^{t} = \max\{0, \text{accuracy}(t) - \text{accuracy}(t-1)\} \tag{12}$$

and ranges between 0 and 1. A value close to 0 implies a high stability.

Finally, another aspect to be considered is the ability of the algorithm to react quickly to changes. This is measured by the

Table 7
Accuracy and stability of DAPSO and VNS on the dynamic *k-series* instances over different time slices. Best accuracy/stability values are marked in bold.

Instance	Time slice	Accuracy		Stability	
		DAPSO	VNS	DAPSO	VNS
k100	0	0.876	**0.972**	0.532	0.933
	5	0.885	**0.949**	**0.003**	0.039
	10	**0.865**	0.822	**0.004**	0.092
	15	**0.765**	0.681	**0.000**	**0.000**
	20	**0.765**	0.681	**0.000**	**0.000**
	25	**0.765**	0.681	**0.000**	**0.000**
	Average	**0.820**	0.797	**0.090**	0.177
k250	0	0.618	**0.937**	**0.334**	0.916
	5	0.748	**0.924**	**0.015**	0.036
	10	0.743	**0.875**	**0.019**	0.055
	15	0.728	**0.829**	**0.000**	**0.000**
	20	0.728	**0.829**	**0.000**	**0.000**
	25	0.728	**0.829**	**0.000**	**0.000**
	Average	0.716	**0.866**	**0.061**	0.168
k500	0	0.777	**0.944**	**0.277**	0.885
	5	0.805	**0.928**	**0.021**	0.068
	10	0.741	**0.819**	**0.019**	0.054
	15	0.704	**0.779**	0.021	**0.000**
	20	0.704	**0.779**	**0.000**	**0.000**
	25	0.704	**0.779**	**0.000**	**0.000**
	Average	0.739	**0.832**	**0.048**	0.168

ε-reactivity, which ranges in [1, *maxgen*] (a smaller value implies a higher reactivity):

$$\varepsilon - \text{reactivity}_i = \min\left\{i' - i | i < i' \leq maxgen, i \in \mathbb{N}, \frac{\text{accuracy}_{i'}}{\text{accuracy}_i} \geq (1-\varepsilon)\right\} \quad (13)$$

For the classical Kilby's instances, we have computed the accuracy at the end of the working day *T*. Table 6 shows the accuracy of our algorithms DAPSO and VNS, compared to other metaheuristics (Ant System (*AS*), Genetic Algorithm ($GA_{2-\text{Opt}}$), and Tabu Search (*TS*)). The accuracy has been computed using the best known solutions of the static instances[5] as the bound to compute accuracy (Min_F^T in Eq. 11). These best known solutions consider all customers to be static, and then are not feasible solutions for the DVRP. They work as a bound for our algorithms. From Table 6, we infer that our algorithms have on average the same average accuracy at the end of the simulation. This accuracy is equal to 0.86 (being 1.0 a perfect metric) which denotes that our algorithms are able to produce good solutions on the conventional dynamic benchmarks.

Table 7 shows the accuracy and stability over the three *k-series* instances on different time slices, and the average on the whole working day. These results are graphically represented in Fig. 6. We have excluded ε-reactivity from this analysis since it provides no significant results (it is always equal to one). It is interesting here to pay attention to the different behaviors of our algorithms on the three instances. The accuracy results confirm numerically what we already explained in Section 4.3, the size of the instance affects differently the performance of our algorithms. In instance *k100*, the highest accuracy levels correspond to DAPSO; although VNS is better in the first time slices (0–5), PSO has a better adaptation from the 10th time slice until the end. VNS achieves the best accuracy for all time slices on the instances *k250* and *k500*, whereas $\text{DAPSO}_{2-\text{Opt}}$ performances are poor due to its slow evolution comparatively to VNS, which adapts faster to the changes. Both the final fitness and the accuracy point to a better performance of DAPSO in *k100* and VNS in the larger *k250* and *k500*. DAPSO provides enough diversity to achieve good solutions in the smaller instance, while VNS profits from the fast convergence of trajectory based techniques. This is to be considered an essential issue in dynamic optimization due to the reduced available time in each time slice.

With respect to stability, DAPSO is more stable than VNS. The difference between algorithms is noticeable in the three instances: the average stability values for DAPSO are always less than 0.1, while for VNS it is ranged between 0.072 and 0.168 (quite stable for a metric which ranges in [0, 1]). This is caused by DAPSO being a population-based metaheuristic, which provides diversity and different types of solutions when a change occurs in the environment; this means DAPSO can choose from a wide range of solutions which one is more adequate in the next time slice. However, VNS provides a single solution at the end of each period; thus there is a steeper fitness variation between the end of a time slice and the beginning of the next one.

5. Conclusion

A vehicle routing problem with dynamic requests has been studied in this article. This problem is important both in research and industrial domains due to its many real world applications. A comparative study between two metaheuristics for this problem has been described, one based on particle swarm optimization (PSO) and another one on variable neighborhood search (VNS). These algorithms are representative of the two main classes of metaheuristics: population-based (PSO) and trajectory-based (VNS) metaheuristics.

A computational study on conventional and newly defined set of large-scale benchmarks was performed. In addition, a study on varying the degree of dynamism has been done to evaluate the impact of this indicator on the performance of our algorithms in terms of servicing customer orders. Furthermore, in order to evaluate the dynamic performance of our approach, several indicators have been used. Weicker's measures allow to assess the accuracy, the stability, and the reactivity of an algorithm throughout the optimization process. Our approaches provide very competitive results comparatively to the other state-of-the-art metaheuristics, and Weicker's indicators demonstrated the high adaptivity and stability of our algorithm. From our study, the following conclusions can be drawn:

1. When comparing PSO and VNS in the classic instances, there are slight differences in terms of the solution quality. VNS computes shorter routes on average, while PSO is more effective at computing new best solutions in the literature.
2. Regarding our new benchmark instances, there are differences depending on the instance size. PSO behaves better in the smallest instance, whereas VNS outperforms PSO in the two biggest ones.
3. Additional measures can improve our knowledge about the algorithmic performance. The accuracy confirms the solutions are good when compared to a static bound. The stability expresses how stable are the algorithms when the environment changes. In our case study, VNS is more accurate than PSO, while PSO is more stable than VNS.
4. When striving to serve as much customers as possible for increasing degrees of dynamism, DAPSO is a better choice. A population-based algorithm such as PSO has a broader number of solutions where new customers should be inserted, therefore it is easier to find some solutions where most new customers fit. A trajectory-based algorithm such as VNS has more difficulties to fit all new customers since it manages a single solution.

Future work will examine problems with more severe fitness rescaling or additional problem characteristics. The aim is to reach a solution accuracy which is close to 0.9 on all the treated instances. We will try to enhance the hybridization scheme and propose multi-population metaheuristics for this problem. Also, we will apply the developed algorithms to other dynamic optimization problems.

Acknowledgements

Authors acknowledge funds from the Associated Teams Program of the French National Institute for Research in Computer Science and Control (INRIA http://www.inria.fr), the Spanish Ministry of Sciences and Innovation European FEDER under contract TIN2008-06491-C04-01 (M* project http://mstar.lcc.uma.es), and CICE, Junta de Andalucía under contract P07-TIC-03044 (DIRICOM project http://diricom.lcc.uma.es). Briseida Sarasola receives grant AP2009-1680 from the Spanish Government. Part of the experiments presented in this work were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see http://www.grid5000.fr).

[5] http://neo.lcc.uma.es/radi-aeb/WebVRP/.

References

[1] G. Ghiani, F. Guerriero, G. Laporte, R. Musmanno, Real-time vehicle routing: solution concepts, algorithms and parallel computing strategies, European Journal of Operational Research 151 (2003) 1–11.
[2] F. Hanshar, B. Ombuki-Berman, Dynamic vehicle routing using genetic algorithms, Applied Intelligence 27 (2007) 89–99.
[3] H. Housroum, T. Hsu, R. Dupas, G. Goncalves, A hybrid GA approach for solving the dynamic vehicle routing problem with time windows, in: 2nd International Conference on Information & Communication Technologies: Workshop ICT in Intelligent Transportation Systems, ICTTA'06, Damascus, Syria, 2006.
[4] R. Montemanni, L. Gambardella, A. Rizzoli, A. Donati, A new algorithm for a dynamic vehicle routing problem based on ant colony system, Journal of Combinatorial Optimization 10 (2005) 327–343.
[5] H. Psaraftis, Dynamic vehicle routing: status and prospects, Annals of Opertations Reasearch 61 (1995) 143–164.
[6] G. Dantzig, J. Ramser, The truck dispatching problem, Operations Research, Management Sciences 6 (1) (1959) 80–91.
[7] T. Blackwell, Particle swarm optimization in dynamic environments, in: Evolutionary Computation in Dynamic and Uncertain Environments, vol. 51, Springer, 2007, pp. 29–49.
[8] X. Li, J. Branke, T. Blackwell, Particle swarm with speciation and adaptation in a dynamic environment, in: GECCO'06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, 2006, pp. 51–58.
[9] J. Liang, P. Suganthan, Dynamic multi-swarm particle swarm optimizer, in: Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE, 2005, pp. 124–129.
[10] T.J. Ai, V. Kachitvichyanukul, A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery, Computers & Operations Research 36 (2009) 1693–1702.
[11] K. Fleszar, I. Osman, K. Hindi, A variable neighbourhood search algorithm for the open vehicle routing problem, European Journal of Operational Research 195 (3) (2009) 803–809.
[12] A. Goel, V. Gruhn, A general vehicle routing problem, European Journal of Operational Research 191 (3) (2008) 650–660.
[13] R. Moretti Branchini, V. Amaral Armentano, A. Løkketangen, Adaptive granular local search heuristic for a dynamic vehicle routing problem, Computers & Operations Research 36 (11) (2009) 2955–2968.
[14] A. Haghani, S. Jung, A dynamic vehicle routing problem with time-dependent travel times, Computers & Operations Research 32 (11) (2005) 2959–2986.
[15] Y. Rochat, É.D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, Journal of Heuristics 1 (1) (1995) 147–167.
[16] J.-Q. Wang, X.-N. Tong, Z.-M. Li, An improved evolutionary algorithm for dynamic vehicle routing problem with time windows, in: ICCS'07: Proceedings of the 7th International Conference on Computational Science, Part IV, Springer, Berlin, Heidelberg, 2007, pp. 1147–1154.
[17] L. Sun, X. Hu, Z. Wang, M. Huang, A knowledge-based model representation and on-line solution method for dynamic vehicle routing problem, in: ICCS '07: Proceedings of the 7th International Conference on Computational Science, Part IV, Springer, Berlin, Heidelberg, 2007, pp. 218–226.
[18] J.D. Magalhães, J.P.D. Sousa, Dynamic VRP in pharmaceutical distribution – a case study, Central European Journal of Operations Research 14 (2) (2006) 177–192.
[19] P. Kilby, P. Prosser, P. Shaw, Dynamic VRPs. A study of scenarios, APES-06-1998, University of Strathclyde, UK, 1998.
[20] K. Lund, O. Madsen, J. Rygaard, Vehicle Routing Problems with Varying Degrees of Dynamism, Tech. Rep., IMM, The Department of Mathematical Modelling, Technical University of Denmark, 1996.
[21] A. Larsen, The Dynamic Vehicle Routing Problem, Ph.D. Thesis, Technical University of Denmark, 2000.
[22] T. Van Woensel, L. Kerbache, H. Peremans, N. Vandaele, Vehicle routing with dynamic travel times: a queueing approach, European Journal of Operational Research 186 (3) (2008) 990–1007.
[23] R. Bent, P.V. Hentenryck, Dynamic vehicle routing with stochastic requests, in: International Joint Conference on Artificial Intelligence, IJCAI, 2003.
[24] M. Gendreau, F. Guertin, J. Potvin, E. Taillard, Parallel tabu search for real-time vehicle routing and dispatching, Transportation Science 33 (4) (1999) 381–390.
[25] G. Alvarenga, R. Silva, G. Mateus, A hybrid approach for the dynamic vehicle routing problem with time windows, in: Proceedings of the Fifth International Conference on Hybrid Intelligent Systems, IEEE Computer Society, Washington, DC, USA, 2005, pp. 61–67.
[26] A. Fabri, P. Recht, On dynamic pickup and delivery vehicle routing with several time windows and waiting times, Transportation Research Part B: Methodological 40 (4) (2006) 335–350.
[27] M. Gendreau, F. Guertin, J. Potvin, R. Séguin, Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries, Transportation Research Part C 14 (3) (2006) 157–174.
[28] D. Sáez, C. Cortés, A. Núñez, Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering, Computers & Operations Research 35 (11) (2008) 3412–3438.
[29] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE International Conference on Neural Networks, 1995, vol. 4, 1995, pp. 1942–1948.
[30] J. Kennedy, R. Eberhart, Y. Shi, Swarm Intelligence, Morgan Kaufmann Publishers, 2001.
[31] S. Janson, M. Middendorf, A Hierarchical Particle Swarm Optimizer for Dynamic Optimization Problems, in: Applications of Evolutionary Computing, vol. 3005, Springer, Coimbra, Portugal, 2004, pp. 513–524.
[32] M. Khouadjia, L. Jourdan, E.-G. Talbi, A particle swarm for the resolution of the dynamic vehicle routing, in: International conference on Metaheuristics and Nature Inspired Computing (META'08), 2008.
[33] R. Bent, P. Hentenryck, Scenario-based planning for partially dynamic vehicle routing with stochastic customers, Operations Research 52 (6) (2004) 977–987.
[34] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, in: Proceedings of the 1999 Congress on Evolutionary Computation, 1999 (CEC 99), vol. 3, 1999, p. 1882.
[35] S. Yang, Explicit memory schemes for evolutionary algorithms in dynamic environments, Evolutionary Computation in Dynamic and Uncertain Environments 51.
[36] C. Rego, Node-ejection chains for the vehicle routing problem: sequential and parallel algorithms, Parallel Computing 27 (3) (2001) 201–222.
[37] P. Hansen, N. Mladenović, An introduction to variable neighborhood search, in: S. Vo, S. Martello, I. Osman, C. Roucairol (Eds.), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, 1999, pp. 433–458 (Chapter 30).
[38] I. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, Annals of Opertations Reasearch 41 (1–4) (1993) 421–451.
[39] S. Lin, Computer solutions of the traveling salesman problem, Bell System Computer Journal 44 (1965) 2245–2269.
[40] J.Y. Potvin, J.M. Rousseau, An exchange heuristic for routing problems with time windows, Journal of the Operational Research Society 46 (1995) 1433–1446.
[41] G. Clarke, J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Operations Research 12 (4) (1964) 568–581.
[42] P.C. Yellow, A computational modification to the savings method of vehicle scheduling, Operational Research Quarterly (1970–1977) 21 (2) (1970) 281–283.
[43] E. Talbi, Metaheuristics: From Design To Implementation, Wiley-Blackwell, 2009.
[44] P. Cohen, R. Kohavi, Empirical Methods for Artificial Intelligence, MIT press Cambridge, MA, 1995.
[45] Geekbench benchmark, http://www.primatelabs.ca/geekbench, 2010.
[46] K. Weicker, Performance measures for dynamic environments, in: Parallel Problem Solving from Nature PPSN VII, Springer, 2002, pp. 64–76.