J Supercomput (2013) 63:836–853
DOI 10.1007/s11227-012-0774-x

Multi-environmental cooperative parallel metaheuristics for solving dynamic optimization problems

Mostepha R. Khouadjia · El-Ghazali Talbi · Laetitia Jourdan · Briseida Sarasola · Enrique Alba

Published online: 28 June 2012

Abstract In dynamic optimization problems, changes occur over time. These changes could be related to the optimization objective, the problem instance, or involve problem constraints. In most cases, they are seen as an ordered sequence of sub-problems or environments that must be solved during a certain time interval. The usual approaches tend to solve each sub-problem when a change happens, dealing always with one single environment at each time instant. In this paper, we propose a multi-environmental cooperative model for parallel meta-heuristics to tackle dynamic optimization problems. It consists in dealing with different environments at the same time, using different algorithms that exchange information coming from these environments. A parallel multi-swarm approach is presented for solving the Dynamic Vehicle Routing Problem. The effectiveness of the proposed approach is tested on a well-known set of benchmarks, and compared with other meta-heuristics

M.R. Khouadjia · E.-G. Talbi · L. Jourdan
INRIA Lille Nord-Europe, Lille, France

M.R. Khouadjia
e-mail: mostepha-redouane.khouadjia@inria.fr

L. Jourdan
e-mail: laetitia.jourdan@inria.fr

E.-G. Talbi (✉)
University of Lille 1, CNRS UMR, CNRS 8022, Lille, France
e-mail: talbi@lifl.fr

B. Sarasola · E. Alba
E.T.S.I. Informática, Universidad de Málaga, Málaga, Spain

B. Sarasola
e-mail: briseida@lcc.uma.es

E. Alba
e-mail: eat@lcc.uma.es

from the literature. Experimental results show that our multi-environmental approach outperforms conventional meta-heuristics on this problem.

Keywords Dynamic optimization problems · Multi-environmental cooperative model · Integration policies · Multi-swarm · Dynamic vehicle routing problem

1 Introduction

While most of the optimization problems discussed in the scientific literature are static, many real-world problems change over time, i.e. they are dynamic. In those cases, the optimization algorithm has to track a moving optimum as closely as possible. These problems are known as Dynamic Optimization Problems (DOPs). A DOP can be described as a sequence of changes that occur over time. Each change is seen as the arrival of a new static optimization problem that must be solved in a certain time interval. To tackle this kind of optimization problem, several approaches have been proposed in the literature, e.g. random immigrants [8], multi-populations [17], and memory schemes [16]. These mechanisms try to enhance their adaptability to the changing environment by reusing some information gathered previously during the search. Unless the change in the problem is strong, the new environment is usually correlated to the old one. This correlation between changes or sub-problems can be used to enhance the algorithm adaptation during the optimization process. A "sub-problem" is the problem state in one single period, where a period is defined as the interval between two consecutive changes in the environment.

In this work, we use this correlation characteristic to deal with the changing environment. The idea is that the algorithm must be capable of continuously adapting the solutions. A DOP can be viewed as a sequence of time-dependent sub-problems. Therefore, sequential algorithms usually deal with one sub-problem at a time. However, different algorithms could benefit from exploring different sub-problems at the same time instant. A cooperative parallel meta-heuristics seem to be a suitable candidate. Cooperative parallel models have been largely discussed in the literature [1]; they are models where two or more algorithms are executed in parallel and exchange information to guide the search. However, the conventional parallel model must be slightly redesigned to address the parallel solving of DOPs. Since several independent meta-heuristics are launched in parallel, they may need to deal with different problem environments. Therefore, different algorithms exchange information related to the search on different sub-problems. The difficulty is the integration of the received information that come from different environments into the host population with respect to the environment treated by the local algorithm.

We propose a multi-environmental cooperative algorithmic-level parallel model of meta-heuristics for solving DOPs. This model is characterized by its adaptive integration strategies, which are designed to take into account a parallel solving of different sub-problems. These sub-problems correspond to the ordered steps of the entire problem progress.

In DOPs, it has been argued that multi-population algorithms are potentially well-suited to track the shifting optimum [3, 4, 13]. One of the most popular multi-population approaches is Particle Swarm Optimization (PSO) [2, 3, 13]. Cooperative multi-swarm can tackle DOPs, as well as improve the quality of the obtained solutions and their robustness.

In this paper, a Multi-Environmental Multi-Swarm Optimizer (MEMSO) based on our multi-environmental parallel model is proposed. To validate our approach, we have chosen a well-known real-world problem that is the Dynamic Vehicle Routing Problem (DVRP).

The remainder of this article is organized as follows: Sect. 3 presents our multi-environmental cooperative algorithmic-level parallel model, whose integration policies are described in detail in Sect. 4. Section 5 gives a formal definition of the DVRP, while Sect. 6 describes the proposed Multi-Environmental Parallel Meta-heuristic based on Multi-Swarm approach. Computational results are given and analyzed in Sect. 7. Finally, Sect. 8 presents conclusions and opens some lines for further research.

2 Dynamic optimization problems

A DOP can be described as a problem in which available information can change during the optimization process. Any dynamic problem can be expressed as $P(t) = (X(t), f(t))$, which is a time dependent formulation of the static definition of the problem itself. Thus, any component of the problem can change with time, whether the objective function, decision variables, or the constraints. A discrete-time dynamic problem can be viewed as a series of P instances; each instance is a static sub-problem, which starts at time t and must be solved within a specific deadline Δ_t.

$$P = \{(p_i, t_i, \Delta_t)/i = 0, 1, \dots, i_{\max}\} \tag{1}$$

with this information the duration of the instance i is $\Delta_i = t_{i+1} - t_i$. 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 change δ_{i+1} on the instance p_i. This is expressed by $p_{i+1} = p_i \oplus \delta_{i+1}$. So, the solutions obtained at time t_i could not be the feasible solutions for time t_{i+1}. Each environment corresponds to a given sub-problem of the entire problem. Since the problem can be considered a series of instances, the goal of the optimization process is no longer finding a single optimal solution, but rather, tracking the shifting optima over the time since the optimal solution for one instance could be a poor solution or possibly even infeasible in the next environment.

3 Multi-environmental cooperative algorithmic-level parallel model

In the cooperative model for parallel meta-heuristics, the different algorithms exchange information related to the search with the purpose of computing better solutions. We describe here four questions that arise while designing our multi-environmental cooperative parallel model for meta-heuristics [21, 22]. The main issue is how to integrate information coming from different environments.

1. *The exchange decision criterion*: The exchange of information between the metaheuristics can be decided either in a blind (periodic or probabilistic) way or according to an adaptive criterion. Periodic exchange occurs in each algorithm after a fixed number of iterations. Probabilistic exchange consists in performing a communication operation after each iteration with a given probability. Conversely, adaptive exchanges are guided by some run time characteristics of the search (e.g. evolution of the solution quality). Our model could use any of these exchange decision criteria. In this work, we exchange solutions periodically for the sake of simplicity.
2. *The exchange topology*: The communication exchange topology indicates for each meta-heuristic its neighbor(s) regarding the exchange of information that is, the source/destination algorithm(s) of the information. Several works have been dedicated to the study of the impact of the topology on the quality of the provided results, and they show that cyclic graphs are better than complete graphs [1]. The ring, mesh, and hypercube regular topologies are often used. Since we deal with DOPs, we have decided to use a ring topology in order to enhance diversity (each swarm has only two neighbors, so we expect to obtain more diverse populations than using higher topology neighborhood sizes).
3. *The exchanged information*: This parameter specifies the information to be exchanged between the algorithms. It can be composed of:

   – *Solutions*: This information deals with a selection of the generated and stored solutions during the search. In general, it contains elite solutions that have been found, such as the best solution at the current iteration, local best solutions, global best solution, neighborhood best solution, best diversified solutions, and randomly selected solutions. The quality of the solutions must also be sent so that the evaluation of the solutions is not recomputed in the destination metaheuristics. For trajectory-based metaheuristics (i.e., single-solution methods, such as Simulated Annealing and Tabu Search), the information exchanged is generally the best found solution. For population-based meta-heuristics (i.e., those managing a set of solutions at each iteration, such as Evolutionary Algorithms and Particle Swarm Optimization), the number of solutions to exchange may be an absolute value or a given percentage of the population. Any selection mechanism can be used to select the solutions. The most used selection strategy consists in selecting the best solutions for a given criteria (e.g., objective function of the problem, diversity, age) or random ones.
   – *Search memory*: This information deals with any element of the search memory that is associated with the involved metaheuristic. Depending on this latter, it could be short-term or long-term memories (tabu search) or pheromone trails (ant colonies).

   In this work, our algorithm exchanges particles.
4. *The integration policy*: The integration policy deals with the usage of the received information. In general, the local variables are updated using the received ones. For population-based meta-heuristics, any replacement strategy may be applied to the local population by the set of received solutions. For example, an elitist replacement will integrate the received k solutions by replacing the k worst solutions of the local population. The multi-environmental approach can lead to a

situation in which the received solutions do not belong to the local search space, or they are not feasible. In this case, a mapping function is necessary to obtain feasible solutions from those received. The integration policy must be adapted to take into account this mapping to assure the inter-operability between the different environments during the optimization process. In Sect. 4, we present in details the operating of these policies that are dedicated to DOPs.

4 Integration policies for dynamic multi-environmental parallel models

In a dynamic multi-environmental parallel model, the distributed meta-heuristics can attempt to deal with the same sub-problem as well as different sub-problems at a given moment. So, we can envisage different policies to allow the inter-operability of the exchanged solutions. In Fig. 1, a dynamic problem is split into 4 sub-problems. The population is decomposed into several subpopulations distributed among different nodes. Each node represents a metaheuristic and is responsible for the evolution of one subpopulation. The nodes deal with different sub-problems, which progress and evolve from a state to another over time.

We assume that the dynamic problem P is split into n sub-problems. Each one corresponds to an environment. The local meta-heuristic is dealing with the sub-problem

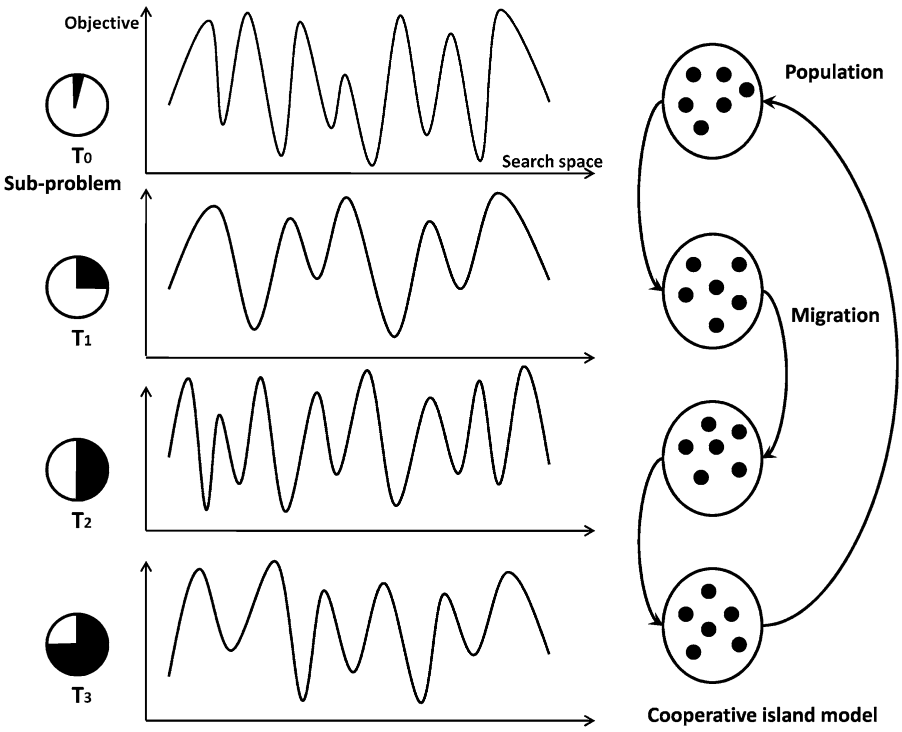


Fig. 1 Multi-environmental cooperative parallel metaheuristics

p_i, and the received immigrant solution s_j is coming from the environment that deals with the sub-problem p_j. Solution s_i is a local feasible solution. Algorithms 1 and 2 summarize the adopted integration policies when an immigrant solution s_j is received.

The *adaptive policy* (see Algorithm 1) consists in adapting the received solution to the local environment before its integration. This adaptation goes through a surjective mapping function Ψ_{ij} between the environment j and the environment i. This function allows to provide from an immigrant solution s_j, a feasible solution s_i for the local sub-problem i. Depending on the sub-problem i, the adaptation function should take into account the inner specificities of this latter in term of solution's representation, constraints of the problems and objective function.

The *keeping policy* (see Algorithm 2) keeps the immigrant solution in a temporary memory Q when it does not belong to the local search space. When the local environment reaches the immigrant's environment, a copy of the solution s_j is created before its removing from the memory. In both policies, if a solution coming from an environment $j = i$ is received (where i is the host environment), a local copy of the received solution is created. To improve the solution quality, a local search could be done on the resulting solutions. Finally, the solution s_i is integrated into the population by replacing one of the local solutions according to a replacement strategy (random, worst, etc.).

Algorithm 1 Adaptive integration policy

```
for each received solution $s_j$ do
    if $(i \neq j)$ then
        //Adapt the solution $s_j$ to the environment $i$.
        $s_i = \Psi_{ij}(s_j)$;
    else
        $s_i = s_j$;
    end if
    Integrate the solution $s_i$ into the population according to replacement strategies.
end for
```

5 A case study on the dynamic vehicle routing problem

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. 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 [7]. In this context, Dynamic Vehicle Routing Problems (DVRPs) are getting increasingly important [10, 15, 19]. The VRP [6] is a well-known combinatorial problem which consists in designing routes for a fleet of vehicles that are to

Algorithm 2 Keeping integration policy

```
$Q$: a temporary memory of received solutions that do not belong to the local search space,
if $Q \neq \emptyset$ then
    for each solution $s_k$ from $Q$ do
        if $(i = k)$ then
            $s_i = s_k$;
            $Q = remove(Q, s_k)$;
        end if
    end for
end if
for each received solution $s_j$ do
    if $(i \neq j)$ then
        //Keep the solution $s_j$ in the memory queue $Q$;
        $Q = pushback(Q, s_j)$;
    else
        $s_i = s_j$;
    end if
end for
Integrate the solution $s_i$ into the population according to replacement strategies.
```

service a set of geographically dispersed locations at the least cost. Several dynamic features can 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. A general description of the VRP is given in Sect. 5.1. Section 5.2 presents the dynamic version, and particularly that with dynamic requests.

5.1 The static vehicle routing problem

The static VRP can be mathematically modeled 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 $E = \{(c_i, c_j) | c_i, c_j \in C, i < j\}$. A set of m homogeneous vehicles with capacity Q originate from a single depot, represented by the vertex c_0 and must service all the customers in 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 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

$$Cost(R_j) = \sum_{i=0}^{k} d_{i,i+1}, \tag{2}$$

and the cost of the problem solution (S) is

$$F_{\mathrm{VRP}}(S) = \sum_{j=1}^{m} Cost(R_j), \tag{3}$$

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

$$Demand(R_j) = \sum_{i=1}^{k} q_i \times y_i^j \leqslant Q^j, \tag{4}$$

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

$$y_i^j = \begin{cases} 1, & \text{if } c_i \text{ is served by the vehicle } j \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

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.

$$Time(R_j) = \sum_{i=0}^{k} d_{i,i+1} + \sum_{i=1}^{k} \delta_i \leq T \tag{6}$$

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.

5.2 The dynamic vehicle routing problem

The Dynamic VRP [19] is strongly related to the static VRP, as it can be described as a discrete-time 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.

This change in the environment can be due to several factors, for example, travel times can be time [9] or traffic-dependent [23], orders may be withdrawn or changed [20], some clients may be unknown when the execution begins, 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 [12]. In this model, a partial static VRP should be solved each time a new request is received.

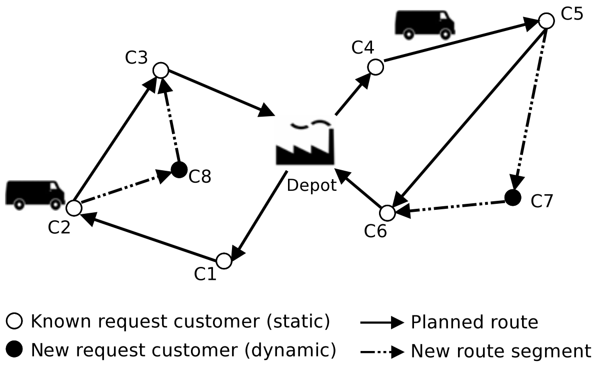

C1
C2
C3
C4
C5
C6
C7
C8
Depot
○ Known request customer (static)
● New request customer (dynamic)
Planned route
New route segment

Fig. 2 A dynamic vehicle routing problem case

A simple example of a dynamic vehicle routing situation is shown in Fig. 2. In the example, two incapacitated vehicles must service both static and dynamic request customers. Advance requests are represented with white nodes, while those that are dynamic are depicted by black nodes. The solid lines represent the two routes that the dispatcher has planned before the vehicles leave the depot, while the new route segments are indicated by dotted lines.

6 Multi-environment parallel multi-swarm for solving the dynamic vehicle routing problem

Multi-population approaches have been used to enhance meta-heuristics for diversity [2, 3]. An interesting algorithm in this field is Particle Swarm Optimization (PSO). A multi-swarm based meta-heuristic can be beneficial on the DVRP [11]. In this section, we present our Multi-Environment Multi-Swarm Optimizer (MEMSO), and its adaptation to the DVRP.

6.1 Algorithm

PSO is a population-based technique inspired by models of social swarm and flock behavior. A particle is defined by its position x_i, the position of its personal best solution (personal attractor) found so far p_i, and its velocity v_i. Furthermore, each particle knows the best position found so far by the global swarm p_g. The algorithm proceeds iteratively, updating first all velocities, and then all particle positions as follows:

$$v_i = \omega v_i + \varphi_1 \times r_1(p_i - x_i) + \varphi_2 \times r_2(p_g - x_i) \tag{7}$$

$$x_i = x_i + v_i \tag{8}$$

Equation (7) is used to calculate the ith particle's new velocity by taking into account three terms: the particle's previous velocity, the distance between the particle's best, and current position, and finally, the distance between swarm's best position (swarm attractor) and the ith particle's current position. Then the ith particle moves towards a new position according to Eq. (8). The role of the inertia weight ω is to regulate the moving velocity of the particles. The parameters φ_1, φ_2 control the relative attraction of the personal best and global best found solutions. Finally, r_1, r_2 are random variables drawn with uniform probability from [0, 1].

The multi-swarm algorithm is presented in Algorithm 3. It iterates a main loop with two stages: test for function change and initialization if necessary, and the updating of the personal and swarm attractors. At each change in the environment, all the particles are re-initialized and evaluated. Consequently, the personal attractor is updated. Then the particles move through the search space according to Eqs. (7) and (8), and update their attractors. When the migration criterion is met, the exchange of particles is carried out between the meta-heuristic and its neighbor(s) according to the exchange topology.

Algorithm 3 Parallel MEMSO for the DVRP

```
repeat
  for Each swarm $s_i$ do
    Evaluate $f(p_g^i)$
    if new value is different from last iteration then
      for Each particle $j$ of swarm $s_i$ do
        Initialize $v_j^i$, $x_j^i$, $p_j^i = x_j^i$
        Evaluate $f(p_j^i)$
      end for
      $p_g^i := \arg\min\{f(p_j^i)\}$
    end if
  end for
  repeat
    repeat
      for Each particle $j$ of swarm $s_i$ do
        Apply (7) and (8).
        Evaluate $f(x_j^i)$.
        if $f(x_j^i) < f(p_j^i)$ then
          $p_j^i := x_j^i$
        end if
        if $f(x_j^i) < f(p_g^i)$ then
          $p_g^i := x_j^i$
        end if
      end for
    until migration criterion reached
    Migration(); ParticleIntegration();
  until termination criterion reached
until no change occurs in the environment
```

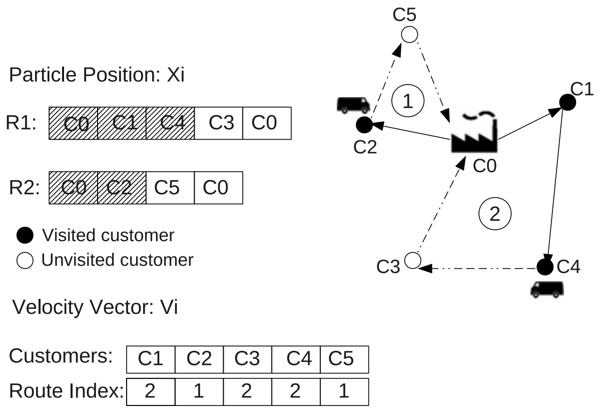


Fig. 3 Particle position and its velocity vector for DVRP

6.2 Particle representation

We propose a simple discrete representation which expresses the route of vehicles over the n customers to serve. We set up a dedicated solution encoding for the DVRP problem. The representation allows the insertion of dynamic customers in the already planned routes. Since customer requests arrive along time, it is necessary to have some information about the state of each customer (served/unserved) and its processing time (i.e., the time in which he is served). Furthermore, the representation specifies for each vehicles: its current position in the service area, its remaining capacity, its traveled distance and its status (committed/uncommitted). We adopted a representation consisting in a permutation of customers. Each permutation contains both customers and route splitters delimiting different routes, so we will use a permutation with length $m = n + k$ for representing a solution for the DVRP with n customers and a maximum of $k - 1$ possible routes. Each route R starts and ends at the depot ($c_0 = c_{n+1}$). The representation is summarized in Fig. 3.

$$R : (c_0, c_1, c_2, \dots, c_i, \dots, c_n, c_{n+1}) \tag{9}$$

Since we deal with an on-line problem, we keep for each customer its time of service. At any moment, we know for each vehicle its remaining capacity, its location in the service area, and its status (waiting, committed) concerning new requests. The initial population is obtained by generating a permutation of customers according to the nearest neighborhood greedy heuristic. The algorithm first starts a route with a random client and repeatedly visits the nearest client until the vehicle constraint (capacity, depot time window) cannot be satisfied. New routes are built until all clients have been visited. The initial population is constituted by the set of vehicle routes which serve customers that were left over from the day before.

6.3 Particle's movement

The particle velocity is a vector of n dimensions, where n is the number of customers to serve. The velocity vector corresponds to the route index, each dimension takes a value in the range $[1, m]$, where m is the number of planned routes. Each vector

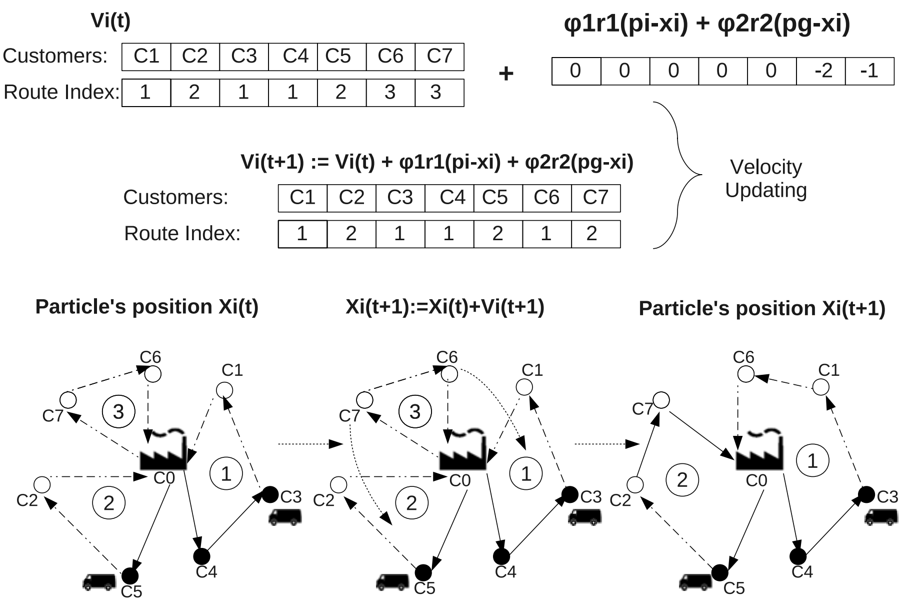

Vi(t)
Customers: C1 C2 C3 C4 C5 C6 C7
Route Index: 1 2 1 1 2 3 3
+
φ1r1(pi-xi) + φ2r2(pg-xi)
0 0 0 0 0 -2 -1
Vi(t+1) := Vi(t) + φ1r1(pi-xi) + φ2r2(pg-xi)
Customers: C1 C2 C3 C4 C5 C6 C7
Route Index: 1 2 1 1 2 1 2
Velocity Updating
Particle's position Xi(t)
Xi(t+1):=Xi(t)+Vi(t+1)
Particle's position Xi(t+1)
C0
C1
C2
C3
C4
C5
C6
C7
1
2
3

Fig. 4 Illustration of a particle's movement: the velocity is updated followed by the particle's current position

position corresponds to the route where the corresponding customer belongs. The particle's movement is summarized in shifting customers from one route to another one according to the new velocity vector (calculated using Eq. (7)), which involves the route index of the best position found so far by the particle and the global best position of the swarm. Each customer is inserted in a route using the cheapest insertion heuristic, i.e., at a location which minimizes the overall cost of the entire tour. This requires computing the cost of inserting the customer at each location in the route. A new route is created if it is cheaper to insert the customer in a new route.

Figure 4 shows an example of the particle position updating. The movement related to the particles is very similar to the λ-interchange local optimization method that is one of the most successful techniques in the past years [18].

We have used the 2-Opt [14] heuristic as a local search for each swarm. It is applied after the move of particles. For more efficiency, our local search is designed in a manner in which we can avoid the whole route evaluation. A way to evaluate the set of candidates $N(s)$ 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 $\Delta(s, m)$ applied to a solution s rather than the complete evaluation of the neighbor solution $f(s') = f(s \oplus m)$ [22].

6.4 Integration policies

The integration policies of our multi-environmental based approach follow the policies described in Sect. 4. For the adaptive integration policy, the mapping function

Ψ_{ij} must be defined. Recall that this function provides a feasible solution s_i from an immigrant solution s_j.

The whole dynamic problem is split into sub-problems. These sub-problems are ordered according to the customers request time. In each sub-problem P_i, the customers to serve are those with request time into the time slot $[0, T_i]$. Depending on the order of the sub-problems i and j, the mapping is summarized in two cases as follows:

- $i > j$: Insert customers that belong to the sub-problem i into the vehicle routes of the solution j. These customers have a request time in $]T_j, T_i]$.
- $i < j$: Remove customers that do not belong to the sub-problem i from the vehicle route of the solution j. These customers are those that appear in the time slot $]T_i, T_j]$.

7 Experimental analysis

The results achieved by our algorithm are presented in this section. The MEMSO algorithm has been implemented on ParadisEO framework[1] and the experiments were carried out using the Grid'5000[2] experimental testbed. A comparison of the solutions quality in term of minimizing travel distances is done between our multi-environmental multi-swarm, and several algorithms proposed previously in literature. We study the performance of both integration policies on a well-known set of benchmarks.

In order to compare our algorithms with the reported results of other approaches on DVRP [10, 15], a number of parameters need to be set. To standardize them, Montemanni et al. [15] fixed some parameters. The first is n_{ts}, the number of time slices in the optimization process. This parameter subdivides the working day T into discrete time periods T_s, in which the optimization is performed. They found that setting the parameter to $n_{ts} = 25$ yielded the best tradeoff between the objective value and computational cost. Second, the cutoff time T_{co} was set to 0.5. It corresponds to the degree of dynamism. This means that the orders which arrive after the half of the working day ($0.5 \times T$) are postponed to the following day. Finally, the advanced commitment time T_{ac} was set to 0. This parameter gives the drivers an appropriate reaction time after having been committed to the new orders. The algorithm parameters used for our approach are summarized in Table 1. They have been fixed in order to have the best trade-off between intensification and diversification, using the default values defined for the best algorithms in the literature [1]. For each instance, 30 independent runs were performed.

Table 2 reports the results obtained by our algorithm on the adaptive MEMSO$_A$ and keeping MEMSO$_K$ policies. They are compared with Ant System (*AS*) [15], Genetic Algorithm (GA), and Tabu Search (TS), both proposed in [10]. We highlight the best found solutions into dark shaded cells, and the average results are marked into light shaded cells.

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

[2] https://www.grid5000.fr.

Table 1 Algorithm parameters for multi-swarm meta-heuristic

Parameter type	Default value	Range
Number of swarms	8	–
Population size	100	–
Migration topology	Directional ring	–
Migration frequency	1000 evaluations	–
Migration size	5 % population size	–
Inertia weight (ω)	1	–
φ_1	0.75	0.5–1.0
φ_2	0.5	0.5–1.0
Stopping criterion	5000 evals per T_s	$25 \times 5000 = 125000$ evals per T

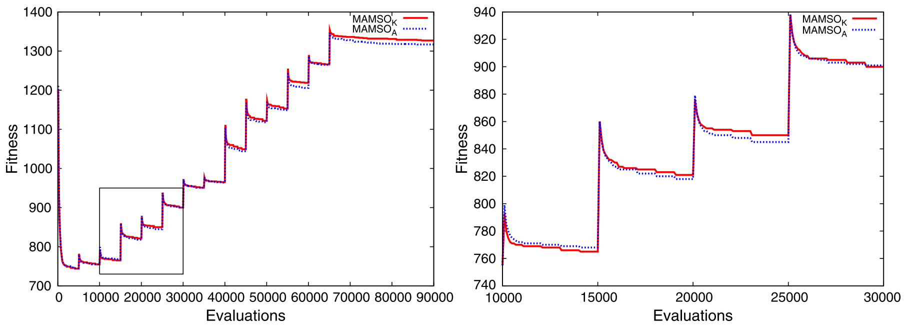
We can see that our algorithms are able to provide higher quality solutions. It outperforms the other meta-heuristics, and gives 17 new best solutions out of the 21 Kilby's instances. Our algorithm provides also the shortest average for the traveled distance on 19 instances. In terms of policies, the adaptive policy gives better results than the keeping policy. It outperforms the keeping policy on 14 instances. The mapping realized during the integration step allows to successfully adapt the immigrant solutions to the local environment, instead of keeping them along the environment's changes. This local adaptation seems to be efficient to deal with dynamic environments.

The improvement provided by our algorithm for the total traveled distance ranges between 3.54 % and 6.71 % compared to the tested metaheuristics. In terms of instances, the improvement reaches for some of them 17.78 %. Figure 5 shows the evolution of the running fitness over the entire time slices on three kinds of instances. Both proposed algorithms MEMSO$_K$ and MEMSO$_A$ are presented. Each peak corresponds to a new changing of the environment due to the arrival of new customer demands. We can see that MEMSO$_A$ has a faster evolution than MEMSO$_K$ over the three instances. The slower evolution of MEMSO$_K$ leads to affect its performance.

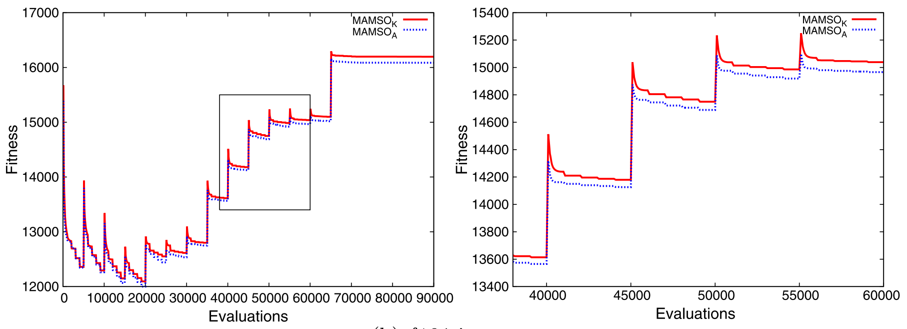
In order to be able to compare our results accurately, we have also performed statistical significance tests for a pairwise comparison of methods [5]. To this end, we have used the Wilcoxon signed rank test. Such a non-parametric statistical test is motivated by the fact that the samples collected here can be considered as matched samples. Indeed, for a given run, both the initial population and the random seed are identical for all algorithms, so that the final indicator values can be taken as pairs. Hence, for a given test instance, and according to a confidence level of 95 % (p-value = 0.05), this statistical test reveals if the sample of approximation sets obtained by a given search method is signifificantly better than the ones of another search method, or if there is no significant difference between both. Table 3 is marked with a $\triangledown$ ($\blacktriangle$) when $MEMSO_A$ obtains significantly shorter (longer) routes than $MEMSO_K$; if it is not possible to assert statistically significant differences, the corresponding cell is empty. According to the Table 3, both approaches are significantly different for 15 instances, while there is no significant difference between their results on 6 instances (c100, c120, c199, tai75a, tai75c, tai150b).

Table 2 Numerical results obtained by MEMSO compared to AS, GA and TS

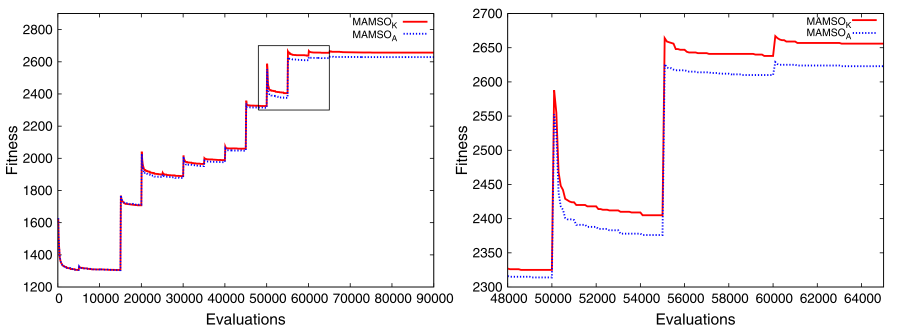
Instances	Metaheuristics									
	MEMSO$_K$		MEMSO$_A$		AS [16]		GA [11]		TS [11]	
	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
c50	571.34	610.67	577.595	**592.95**	631.30	681.86	**570.89**	593.42	603.57	627.90
c75	931.59	965.53	**928.53**	**962.54**	1009.36	1042.39	981.57	1013.45	981.51	1013.82
c100	953.79	973.01	**949.83**	**968.92**	973.26	1066.16	961.10	987.59	997.15	1047.60
c100b	866.42	882.39	**864.19**	**878.81**	944.23	1023.60	881.92	900.94	891.42	932.14
c120	1223.49	1295.79	**1164.63**	**1284.62**	1416.45	1525.15	1303.59	1390.58	1331.22	1468.12
c150	1300.43	1357.71	**1274.33**	**1327.24**	1345.73	1455.50	1348.88	1386.93	1318.22	1401.06
c199	**1595.97**	**1646.37**	1600.57	1649.17	1771.04	1844.82	1654.51	1758.51	1750.09	1783.43
f71	287.51	296.76	283.43	**294.85**	311.18	358.69	301.79	309.94	**280.23**	306.33
f134	15150.5	16193	**14814.10**	16083.82	15135.51	16083.56	15528.81	**15986.84**	15717.90	16582.04
tai75a	1794.38	1849.37	1785.11	**1837.00**	1843.08	1945.20	1782.91	1856.66	**1778.52**	1883.47
tai75b	**1396.42**	1426.67	1398.68	**1425.80**	1535.43	1704.06	1464.56	1527.77	1461.37	1587.72
tai75c	1483.1	1518.65	1490.32	1532.45	1574.98	1653.58	1440.54	**1501.91**	**1406.27**	1527.72
tai75d	1391.99	**1413.83**	**1342.26**	1448.19	1472.35	1529.00	1399.83	1422.27	1430.83	1453.56
tai100a	2178.86	2214.61	**2170.54**	**2213.75**	2375.92	2428.38	2232.71	2295.61	2208.85	2310.37
tai100b	2140.57	2218.58	**2093.54**	**2190.01**	2283.97	2347.90	2147.70	2215.93	2219.28	2330.52
tai100c	**1490.4**	**1550.63**	1491.13	1553.55	1562.30	1655.91	1541.28	1622.66	1515.10	1604.18
tai100d	1838.75	1928.69	**1732.38**	**1895.42**	2008.13	2060.72	1834.60	1912.43	1881.91	2026.76
tai150a	3273.24	3389.97	**3253.77**	**3369.48**	3644.78	3840.18	3328.85	3501.83	3488.02	3598.69
tai150b	**2861.91**	**2956.84**	2865.17	2959.15	3166.88	3327.47	2933.40	3115.39	3109.23	3215.32
tai150c	2512.01	2671.35	**2510.13**	**2644.69**	2811.48	3016.14	2612.68	2743.55	2666.28	2913.67
tai150d	**2861.46**	**2989.24**	2872.80	3006.88	3058.87	3203.75	2950.61	3045.16	2950.83	3111.43
Total	48104.13	50349.66	**47463.03**	**50119.29**	50876.23	53794.02	49202.73	51089.37	49987.8	52725.85



(a) *c150* instance



(b) *f134* instance



(c) *tai150c* instance

Fig. 5 The evolution of each algorithm mean trace for three instances. Each square on the left figure is enlarged in the right figure

8 Conclusion and perspectives

We have proposed an original multi-environmental based approach to deal with DOPs. A multi-swarm meta-heuristic to solve the Dynamic VRP has been designed. Several policies are given for the integration of the immigrant solutions into the different environments. This meta-heuristic has been tested on several benchmark instances, and a study is done on the efficiency of the dedicated integration policies. The experimental results show that our multi-swarm algorithm is able to find high quality solutions compared to the state-of-the-art meta-heuristics, and introduces new best

Table 3 Statistical tests of comparing our algorithms with a Wilcoxon test

Instance	Test	Instance	Test	Instance	Test
c50	▽	c75	▽	c100	▽
c100b	▽	c120	▽	c150	▽
c199		f71	▽	f134	▽
tai75a		tai75b	▽	tai75c	
tai75d	▽	tai100a	▽	tai100b	▽
tai100c	▲	tai100d	▽	tai150a	▽
tai150b		tai150c	▽	tai150d	▲

A ▽ (▲) means that $MEMSO_A$ obtains significantly shorter (longer) routes than $MEMSO_K$; if it is not possible to assert statistically significant differences, the corresponding cell is empty.

solutions. As future work, we would apply the multi-environmental model on parallel cooperative single-solution meta-heuristics, and solve other dynamic problems.

Acknowledgements Authors acknowledge funds from the Associated Teams Program MOMDI of the French National Institute for Research in Computer Science and Control (INRIA), the Spanish Ministry of Sciences and Innovation European FEDER under contracts TIN2008-06491-C04-01 (M* project) and TIN2011-28194 (roadME project), and CICE, Junta de Andalucía under contract P07-TIC-03044 (DIRICOM project). The experiments 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. Briseida Sarasola acknowledges grant AP2009-1680 from the Spanish government.

References

1. Alba E (2005) Parallel metaheuristics: a new class of algorithms. Wiley-Interscience, New York
2. Blackwell T, Branke J (2004) Multi-swarm optimization in dynamic environments. In: Applications of evolutionary computing, Lecture notes in computer science. Springer, Berlin, pp 489–500
3. Blackwell T (2007) Particle swarm optimization in dynamic environments. In: Evolutionary computation in dynamic and uncertain environments. Springer, Berlin, pp 29–49
4. Branke J (2002) Evolutionary optimization in dynamic environments. Kluwer Academic, Dordrecht
5. Cohen P, Kohavi R (1995) Empirical methods for artificial intelligence. MIT Press, Cambridge
6. Dantzig G, Ramser J (1959) The truck dispatching problem. Manage Sci 6(1):80–91
7. Ghiani G, Guerriero F, Laporte G, Musmanno R (2003) Real-time vehicle routing: solution concepts, algorithms and parallel computing strategies. Eur J Oper Res 151:1–11
8. Grefenstette JJ (1992) Genetic algorithms for changing environments. In: Parallel problem solving from nature, pp 139–146
9. Haghani A, Jung S (2005) A dynamic vehicle routing problem with time-dependent travel times. Comput Oper Res 32(11):2959–2986
10. Hanshar F, Ombuki-Berman B (2007) Dynamic vehicle routing using genetic algorithms. Appl Intell 27:89–99
11. Khouadjia M, Alba E, Jourdan L, Talbi EG (2010) Multi-swarm optimization for dynamic combinatorial problems: a case study on dynamic vehicle routing problem. In: Swarm intelligence, Lecture notes in computer science, vol 6234. Springer, Berlin, pp 227–238
12. Kilby P, Prosser P, Shaw P (1998) Dynamic VRPs: a study of scenarios. APES-06-1998, University of Strathclyde, U.K.
13. Li C, Yang S (2008) Fast multi-swarm optimization for dynamic optimization problems. In: Proceedings of the 2008 fourth international conference on natural computation, vol 7. IEEE Comp Soc, Los Alamitos, pp 624–628
14. Lin S (1965) Computer solutions of the traveling salesman problem. Bell Syst Comp J 44:2245–2269
15. Montemanni R, Gambardella L, Rizzoli A, Donati A (2005) A new algorithm for a dynamic vehicle routing problem based on ant colony system. J Comb Optim 10:327–343

16. Mori N, Imanishi S, Kita H, Nishikawa Y (1997) Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In: 7th international conference on genetic algorithms, pp 299–306
17. Oppacher F, Wineberg M (1999) The shifting balance genetic algorithm: improving the GA in a dynamic environment. In: Proceedings of the genetic and evolutionary computation conference, vol 1, pp 504–510
18. Osman I (1993) Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Ann Oper Res 41(4):421–451
19. Psaraftis H (1995) Dynamic vehicle routing: status and prospects. Ann Oper Res 61:143–164
20. Sun L, Hu X, Wang Z, Huang M (2007) A knowledge-based model representation and on-line solution method for dynamic vehicle routing problem. In: Proceedings of the 7th international conference on computational science (ICCS'07), Part IV. Springer, Berlin, pp 218–226
21. Talbi E (2006) Parallel combinatorial optimization. Wiley, New York
22. Talbi E (2009) Metaheuristics: from design to implementation. Wiley, New York
23. Wang JQ, Tong XN, Li ZM (2007) An improved evolutionary algorithm for dynamic vehicle routing problem with time windows. In: Proceedings of the 7th international conference on computational science (ICCS'07:), Part IV. Springer, Berlin, pp 1147–1154