J Comb Optim (2012) 24:437–458
DOI 10.1007/s10878-011-9400-8

Self-organizing maps in population based metaheuristic to the dynamic vehicle routing problem

Jean-Charles Créput · Amir Hajjam · Abderrafiaa Koukam · Olivier Kuhn

Published online: 18 May 2011

Abstract We consider the dynamic vehicle routing problem (dynamic VRP). In this problem, new customer demands are received along the day. Hence, they must be serviced at their locations by a set of vehicles in real time. The approach to address the problem is a hybrid method which manipulates the self-organizing map (SOM) neural network into a population based evolutionary algorithm. The method, called memetic SOM, illustrates how the concept of intermediate structure, also called elastic net or adaptive mesh concept, provided by the original SOM can naturally be applied into a dynamic setting. The experiments show that the heuristic outperforms the approaches that were applied to the Kilby et al. 22 problems with up to 385 customers. It performs better with respect to solution quality than the ant colony algorithm MACS-VRPTW, a genetic algorithm, and a multi-agent oriented approach, with a computation time used roughly 100 times lesser.

Keywords Dynamic vehicle routing problem · Neural network · Self-organizing map · Elastic net · Evolutionary algorithm · Memetic algorithm

1 Introduction

The vehicle routing problem (VRP) is one of the most widely studied problems in combinatorial optimization (Christofides et al. 1979; Gendreau et al. 2002; Cordeau et al. 2005). In the standard VRP, a fleet of vehicles must be routed to visit a set of customers at minimum cost, subject to vehicle capacity constraint and route

J.-C. Créput (✉) · A. Hajjam · A. Koukam
Laboratoire Systèmes et Transports, U.T.B.M., 90010 Belfort cedex, France
e-mail: jean-charles.creput@utbm.fr

O. Kuhn
LIRIS, Bât. Nautibus, 43, Boulevard du 11 novembre 1918, 69622 Villeurbanne cedex, France

duration constraint. In the static version of the problem, it is assumed that all customers are known in advance to the planning process. However, it may be the case that customers, routing costs or service times become available in real-time once the service has begun. Due to the recent advances in communication technologies and positioning systems, it is now possible to address such dynamic problems. Many versions of real-time routing and dispatching problems are presented and studied in the literature (Ghiani et al. 2003; Larsen et al. 2008). As observed in these surveys, most versions incorporate time-windows and/or pick-up and delivery requests (Gendreau et al. 1999).

The problem addressed in this paper is the dynamic VRP with capacity and time duration constraints only (Bertsimas and Simchi-Levi 1996; Kilby et al. 1998; Larsen 2000). In this problem, the customers are the only elements that have a dependence on time. Customers or demands are not known in advance but arrive as the day progresses, and the system has to incorporate them into the already designed routes in real time. As explained by Bertsimas and Simchi-Levi (1996) who studied the problem analytically, problems fitting this model appear frequently in industry. Important examples are found in finished goods distribution and freight consolidation or long-distance courier mail services (Bent and Van Hentenryck 2003). Another example would be the medical interventions of doctors that visit patients along a day. Orders are dispatched to vehicles with the objective of minimizing some tradeoff between the routing cost and the average wait for delivery. The approaches that address dynamic routing can be classified into three categories: simple policies, classical insertion procedures and metaheuristics. As mentioned by Ghiani et al. (2003) and Larsen et al. (2008), simple policies and insertion procedures do not always provide good solutions on realistic applications. Then, using metaheuristic methods, such as tabu search or evolutionary algorithms looks more and more customary in this context.

The approach presented in this paper is based on the concept of the self-organizing map (SOM) (Kohonen 2001). The SOM can be seen as a (on-line) non parametric regression that reflects topological information of some data distribution. It has been used on many applications in data analysis, pattern recognition and classification (Oja et al. 2003). This motivates its application to a dynamic and stochastic setting where data are unknown beforehand but reflect some underlying distribution. In the context of vehicle routing, a topological grid of cluster centers, called the network, represents transport routes that continuously distort and modify their shapes in the plane according to the demand distribution. The network can be seen as an "elastic net" (Durbin and Willshaw 1987) or "adaptive mesh" (Creput et al. 2007). Starting from the SOM as a basic operator, we derive customer insertion and grid deformation operators that are applied to the network. The operators are combined inside a population based metaheuristic similarly as in a memetic algorithm (Moscato and Cotta 2003). A memetic algorithm is an evolutionary algorithm embedding a local search process. Here, the SOM plays the role of a local search by adjusting the network shape to the demand. It is combined with a mapping operator, responsible for the assignment of customers to the network, a fitness evaluation and a selection operator. A last operator is dedicated to residual insertions according to the maximum duration constraint. We already applied the approach called memetic SOM to the static VRP and to the Traveling Salesman Problem (Creput and Koukam 2008, 2009) on instances with up to 85900 cities.

A question is how the classical heuristics to the static VRP (Gendreau et al. 2002; Cordeau et al. 2005) should be reused in a dynamic setting. Here, we exploit the "elastic net" paradigm in this context. Most of the operations consist of network distortions and elementary moves performed in the plane. The optimization process randomly and continuously extracts a demand or customer. Then, it finds its nearest vertex in the network and distorts its neighborhood to the direction of the customer location. Nearest point findings in the plane are the basic operations that are massively and repeatedly performed. They are implemented by a spiral search algorithm (Bentley et al. 1980) which is known to operate in constant time for bounded distributions. The spiral search operates on a cell decomposition of the plane. Since the network does not necessarily modify its structure but only its shape, new arrivals may naturally be incorporated in the network with a weak impact on the internal data structures. The data structure sizes can be adjusted at a lower rate. As they arrive, new demands are inserted in a buffer in constant time and hence participate to the repeated process of demand extraction, nearest vertex search and network distortion.

In this paper, we present empirical evaluations of the method by real-time simulation. A real-time simulator monitors the vehicle successive locations, simulates the service at each demand location, and manages the new arrivals of customers. It is coupled with an asynchronous optimization process continuously running in batch mode. The optimization process solves an evolving VRP, with evolving vehicle capacities and locations, and with an evolving set of available demands. We use the standard benchmarks of Kilby et al. (1998) with 22 problems of sizes from 50 with up to 385 customers to study the method. We present a comparative evaluation against three approaches from the literature. These approaches are an adaptation of the ant colony algorithm MACS-VRPTW (Gambardella et al. 1999) by Montemanni et al. (2005), a genetic algorithm proposed by Goncalves et al. (2007), and a multi-agent oriented approach by Zeddini et al. (2008).

The paper is organized as follows. Section 2 states the dynamic VRP considered in this paper with its constraints and objectives. The problem is presented as a straightforward extension of the static VRP. Section 3 presents the real-time simulator and communication protocol. Section 4 illustrates the "philosophy" of the proposed approach. Section 5 details the memetic SOM algorithm. Section 6 reports the experiments and comparative evaluations carried out. The last section is devoted to the conclusion and further research.

2 The dynamic vehicle routing problem

The static VRP is defined on a set $V = \{v_0, v_1, \dots, v_N\}$ of vertices. Vertex v_0 is a depot at which are based m identical vehicles of capacity Q. The remaining N vertices represent customers, also called requests, orders or demands. A non-negative cost, or travel time, is defined for each edge $(v_i, v_j) \in V \times V$. Each customer has a non-negative load $q(v_i)$ and a non-negative service time $s(v_i)$. A vehicle route is a circuit on vertices each starting and ending at the depot. The VRP consists of designing a set of m vehicle routes of least total cost, such that each customer is visited exactly once by a vehicle, the total demand of any route does not exceed Q, and the total duration of any route does not exceed a preset bound T.

As it is the mostly done in practice (Cordeau et al. 2005), we address the Euclidean VRP where each vertex v_i has a location in the plane, and where the travel cost is given by the Euclidean distance $d(v_i, v_j)$ for each edge $(v_i, v_j) \in V \times V$. Then, the objective for the static problem is the total route length defined by

$$Length = \sum_{i=1,\dots,m} \left(\sum_{j=1,\dots,k_i-1} d(v^i_j, v^i_{j+1}) + d(v_0, v^i_1) + d(v^i_{k_i}, v_0) \right), \tag{1}$$

where $v^i_j \in V$, $0 \le j \le k_i$, $0 \le k_i \le N$, are the ordered set of demands served by the vehicle/route i, with $1 \le i \le m$. The capacity constraint is defined by

$$\sum_{j=1,\dots,k_i} q(v^i_j) \le Q, \quad i \in \{1, \dots, m\}. \tag{2}$$

Assuming without loss of generality that the vehicle speed has value 1 the time duration constraint is given by

$$\sum_{j=1,\dots,k_i} s(v^i_j) + \sum_{j=1,\dots,k_i-1} d(v^i_j, v^i_{j+1}) + d(v_0, v^i_1) + d(v^i_{k_i}, v_0) \le T, \quad i \in \{1, \dots, m\}. \tag{3}$$

The VRP is a NP-hard problem (Gendreau et al. 2002). Then, for large instances, using heuristics is encouraged. In the static VRP, vehicles must be routed to visit a set of customers at minimum cost, assuming that all orders are known in advance. In the dynamic VRP however, new demands arrive randomly in time and must be incorporated into the vehicle schedules and served as the day progresses. The optimization task is a continuous process of collecting demands, forming and optimizing tours, and dispatching demands to the vehicles that serve the demands. The tasks are performed during a period of time that is called the working day or planning horizon of length D.

In a real life situation, the objective function often consists of a trade-off between travel costs and customer waiting time. The customer waiting time is the delay between the occurrence time of a demand and the instant the service of the demand begins. It is often called "response time" or "system time" (Bertsimas and Simchi-Levi 1996). Hence, in addition to the classical objective and constraints defined above, we add a supplementary criterion to be considered when evaluating solutions. This criterion is the average customer waiting time (WT):

$$WT = \sum_{i \in \{1,\dots,N\}} W_i / N, \tag{4}$$

where W_i is the waiting time of demand i, defined by $W_i = st_i - t_i$ where $t_i \in [0, D]$ is the demand occurrence time, and st_i is the time when the service starts for that demand.

The criteria (1)–(3) reflect the structure of the schedules independently of the request arrival dates. They are stated in accordance to the static VRP and to the dynamic VRP's applications considered in this paper. Only, the customer waiting time (4) reflects the dynamicity and reactivity of the system and depends on a real-time realization. We consider the customer waiting time as an important real-time criterion to gauge effectiveness of algorithms on this problem.

3 Real time simulator and optimizer

This section presents the simulator developed in Java which allows simulating the real-time optimization process. To make the things concrete, we assume that a transport company centralizes the optimization procedure. The company receives the orders from the environment, monitors the vehicle locations, and dispatches the plans to the vehicles. Hence, we assume the existence of a communication system between the company, the customers and the vehicles. Also, we assume that the communication times are negligible to the rest of the real-time activities. The simulator consists of two main threads that communicate through an asynchronous protocol. The first thread plays the role of a real-time scheduler which decomposes the working day into many short time-slices based on a timer clock. The goal is to simulate the real-time vehicle activities, which are modeled as synchronous simple state machines. The second thread plays the role of a background task that encapsulates the optimization process. This optimization procedure runs in a continuous way as long as possible. It continuously optimizes the solution according to the current requests. Hence, the computation time includes the synchronous state-machines as well as the optimization process.

The working day is decomposed into many time-slices. The optimization process solves a continuously evolving static VRP. Two parameters denoted *ToR* and *To* respectively define the time-slice length and computation time allowed. The *ToR* value represents the real time fraction of the working day which corresponds to a given time-slice. It is expressed in real-time units. The *To* value represents the computation time allowed to simulate a given time-slice. In the experiments presented in this paper, the *ToR* value is set to the smallest integer greater than $0.1 \times D/N$, with D the working day duration expressed in minutes, and N the total number of demands. Such time-slice decomposition is chosen for practical purposes. It allows adjusting the computation time with the demand rate. In the experiments, the *To* parameter is set to $To = 30$ ms, or $To = 200$ ms, depending on the computation time, fast or long, allowed. As an example, a working day of length $D = 351$ minutes with a request set of size $N = 50$ yield a basic time-slice of length $ToR = 1$ minute, simulated within a computation time of *To* ms. Hence, if we take $To = 30$ ms, a working day of 10 hours is compressed into a computation time period of 18 seconds.

On the one hand, the company receives new orders from the environment and communicates with the vehicles. On the other hand, the company communicates with the optimizer, using mailboxes to exchange information. The company sends to the vehicles the updated route plans. At the same time, it sends to the optimizer the currently available requests removing the ones already served. The optimizer sends back to the company the built vehicle routes. The communication protocol is illustrated in Fig. 1. The optimization time step *Topt* defines the time elapsed between two consecutive route updates ("optimize()" procedure). Each time a route update is received by the optimizer at time t, this one anticipates the vehicle locations at time $t + Topt$. This is

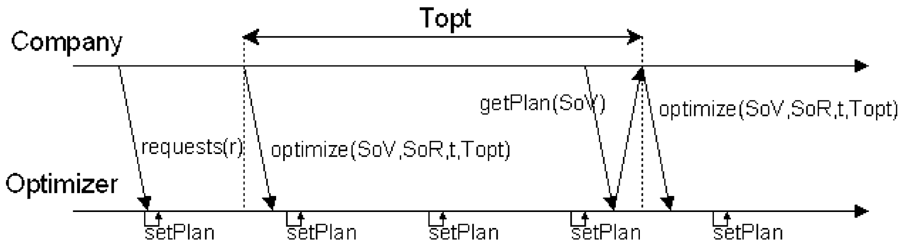


Fig. 1 Asynchronous communication protocol

the future moment when the company will get back the new plan ("getPlan()" procedure). The demands that have to be serviced before this future time are definitely assigned to vehicles and will be serviced during this time interval. All other demands participate to the optimization process and can be reallocated to other vehicles continuously. The optimizer regularly returns the new plan generated ("setPlan()" procedure). The mailbox encapsulates a set of vehicle routes (*SoV* parameter) and a set of requests (*SoR* parameter). New demands that arrive at the company are retransmitted as new orders to the optimizer ("request()" procedure) whenever they arrive on the basis of the basic time slice of *ToR* time units. In the experiments, the optimization time step *Topt* is set to $10 \times ToR$, allowing roughly a single request occurrence on average at each optimization step.

4 The self-organizing map approach to vehicle routing

The self-organizing map (SOM) (Kohonen 2001) which belongs to the class of "intermediate structure" or "elastic net" algorithms has been applied to the traveling salesman problem (TSP) from a long time (Angeniol et al. 1988). The SOM can be seen as a center-based clustering algorithm that preserves density and topology of the data distribution. The data is mapped to a topological grid of cluster centers, called the network. To illustrate the "philosophy" of the SOM behavior when applied to the TSP, an example of a tour construction is illustrated in Fig. 2. The topological grid is a ring network. It is shown in the figure at different steps of a long simulation run. The procedure is applied to the bier127 instance from the TSPLIB (Reinelt 1991). The ring network is a graph that deploys its vertices in the plane to match the cities and to constitute a tour. At the beginning, local moves are large in order to let the ring deploy from scratch (a). Then, the amplitude of the moves slightly decreases in order to progressively freeze the vertices near cities (b–c). At a final step, cities have just to be mapped to their nearest vertex in the ring in order to generate a final tour ordering.

To extend the SOM to the VRP, the SOM can be used as an operator into an evolutionary algorithm. The SOM is now a long run process applied to a population of solutions. This process is interrupted at each cycle, called a generation, by the application of several evolutionary operators. The optimization process alternates network distortion and network projection at each generation. The network distortion refers to the application of a SOM operator. The projection refers to the mapping or matching of customers to network vertices. Here, no recombination, nor crossover, operator is

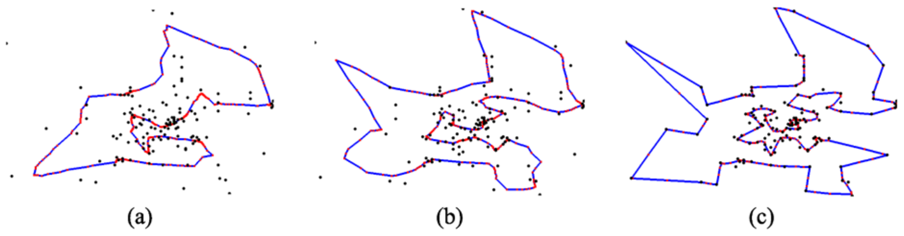


Fig. 2 Tour construction by SOM using the bier127 instance

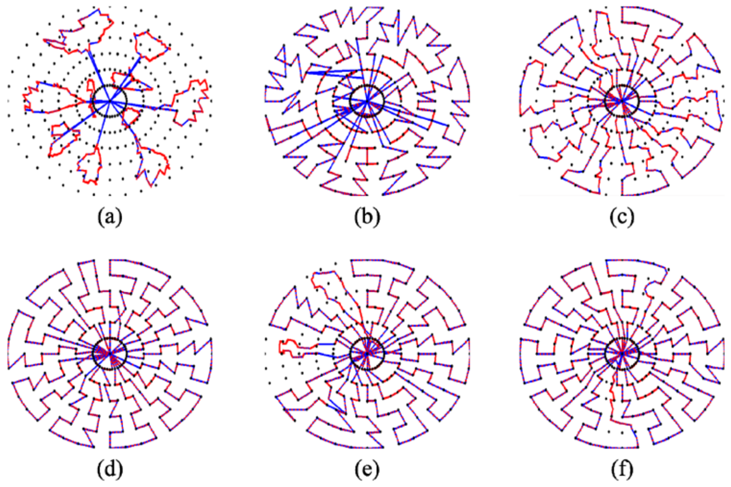
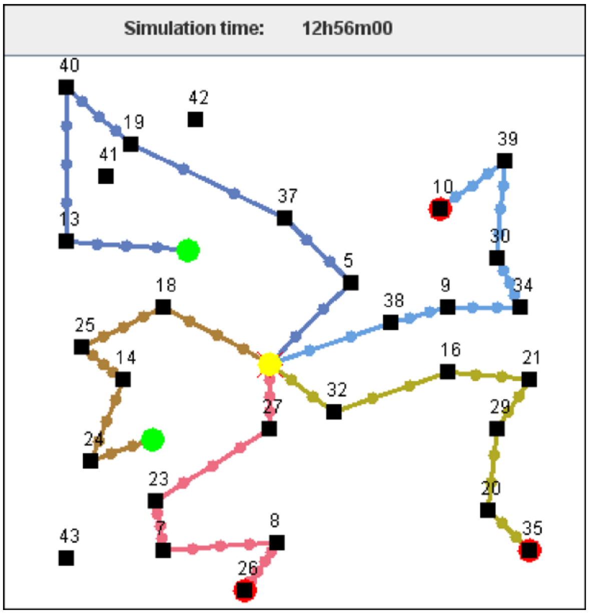


Fig. 3 Static VRP with a SOM based algorithm. (**a**–**d**) Deployment phase. (**e**–**f**) Improvement phase

considered. Once operators have been individually applied to each solution, a selection procedure modifies the overall population according to a fitness evaluation. The optimization process is divided within two phases, that are, a deployment phase followed by an improvement phase. Figure 3 illustrates the optimization process on a static VRP test case with 240 customers of the Golden et al. (1999) benchmarks. The network represents routes starting and ending at the depot. The deployment phase is illustrated in (a–d). Two consecutive pictures show the network as distorted by the SOM operator as in (a), followed by the projection which generates an admissible solution as in (b), at a given generation. During the deployment phase, the moves are large to let the routes deploy from scratch. Figures (a–b) present the network at the beginning of deployment and (c–d) several generations later, the amplitude of moves vanishing. In (e–f), the network is shown at different steps of the improvement phase, illustrating how local perturbations randomly affect some parts of the network at each generation.

The application to a dynamic setting mainly consists of considering an evolving static VRP. The Fig. 4 shows an example of a network shape at a given step of a dynamic simulation. The test case is the instance c50 of the Kilby et al. (1998) benchmarks. Routes are modeled as paths starting from the current vehicle location and

Fig. 4 Dynamic VRP



ending at a common depot. The figure shows five vehicle routes represented by lines that pass by their assigned customers (small black squares) and end at the common depot. The locations of the vehicles, represented by the five filled circles in the figure, evolve step by step as the vehicles travel and perform their services along the working day. Once a vehicle arrives to a customer location, it performs the service (black circle) and then continues to follow its route (gray circle). New customers may arrive in real-time as represented by isolated small squares in the figure. Then, such new customers participate to the optimization process and the vehicle routes will distort themselves to incorporate them in their schedules, exactly the same way as in a static VRP. There is no need to introduce a new insertion procedure to deal with the arrival of new demands. The approach is already based on repeated insertions to the network representing the routes. Once a vehicle has no more customers to be serviced, it returns to the depot following its route. Routes may continuously modify their shapes according to the arrival of new demands. A vehicle can change its direction at any moment to deal with new demands or to follow a better schedule returned by the optimizer.

5 The evolutionary algorithm embedding self-organizing map

5.1 The Kohonen's self-organizing map

The self-organizing map algorithm operates on a non directed graph $G = (A, E)$, called the network, where each vertex $n \in A$ is a neuron having a location $w_n = (x, y)$ in the plane. The set of neurons A is provided with the d_G induced canonical metric $d_G(n, n') = 1$ if and only if $(n, n') \in E$, and with the usual Euclidean distance $d(n, n')$.

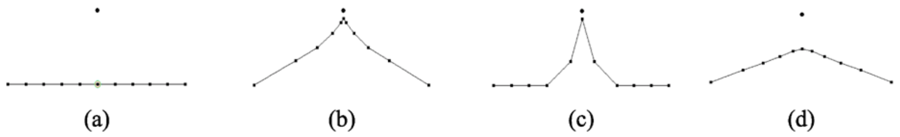


Fig. 5 A single SOM iteration with learning rate α and radius σ. (**a**) Initial configuration. (**b**) $\alpha = 0.9$, $\sigma = 4$. (**c**) $\alpha = 0.9$, $\sigma = 1$. (**d**) $\alpha = 0.5$, $\sigma = 4$

The training/optimization procedure applies a given number of iterations *niter* to a network. The vertex coordinates are randomly initialized into an area delimiting the data set. Here, the data set is the set of demands, or customers. The network is a set of routes, each one represented by a path of vertices starting and ending at some locations. Each iteration follows four basic steps. At each iteration t, a point $p(t) \in \mathcal{R}^2$ is randomly extracted from the data set (extraction step). Then, a competition between neurons against the input point $p(t)$ is performed to select the winner neuron n^* (competition step). Usually, it is the nearest neuron to $p(t)$. Then, the learning law

$$w_n(t+1) = w_n(t) + \alpha(t) \cdot h_t(n^*, n) \cdot (p(t) - w_n(t)), \tag{5}$$

is applied to n^* and to all neurons within a finite neighborhood of n^* of radius σ_t, in the sense of the topological distance d_G, using learning rate $\alpha(t)$ and function profile h_t (triggering step). The function profile is given by the Gaussian

$$h_t(n^*, n) = \exp(-d_G(n^*, n)^2/\sigma_t^2). \tag{6}$$

Finally, the learning rate $\alpha(t)$ and radius σ_t decrease as geometric functions of the time (decreasing step). To perform a decreasing run within t_{max} iterations, at each iteration t, coefficients $\alpha(t)$ and σ_t are multiplied by $\exp(\ln(x_{final}/x_{init})/t_{max})$, with respectively $x = \alpha$ and $x = \sigma$, x_{init} and x_{final} being respectively the values at the starting and final iterations. Examples of a basic iteration with different learning rates and neighborhood sizes are shown in Fig. 5. In the population based metaheuristic presented in this paper, a SOM simulation becomes an operator specified by its running parameters $(\alpha_{init}, \alpha_{final}, \sigma_{init}, \sigma_{final}, t_{max})$.

5.2 Memetic SOM

In order to address the VRP, the SOM has been extended to become an operator embedded into an evolutionary algorithm. The structure of the algorithm is similar to the memetic algorithm, which is an evolutionary algorithm incorporating a local search. The SOM is a long run process interrupted at each evolutionary iteration, called a generation, by the application of problem dependent operators. The SOM process is performed within *Gen* generations, *Gen* being proportional to the problem size N. A deployment (or construction) loop as well as an improvement loop are instantiated. The memetic loop applies a set of operators to a population of *Pop* individuals at each generation. One individual encapsulates exactly one solution. A solution is a network where each vehicle/route is represented by an independent path which starts at the vehicle current location and ends at the depot. A path has $5N_t/m_t$ vertices,

N_t being the number of demands and m_t the number of vehicles at the optimization time-slice t. The number of vertices defines the maximum number of customers a route can handle at a given optimization time-slice. It has been adjusted empirically. The memetic loop has the following structure:

```
Memetic loop:
  0. Initialize population with $Pop$ individuals with routes randomly generated.
  1. Initialize individuals and the SOM parameters for the deployment phase.
  2. $g = 0$
  While not (a stop order is received from the company)
  3. Look at the received messages from the company. When an "update" order is
     received, update the vehicles locations at future time $t + Topt$, the path lengths,
     and the demand set by removing the ones that are serviced before $t + Topt$.
     When a "request" order is received, add the new demands into the demand
     buffer.
  4. Update the best solution by applying MAPPING followed by FITNESS opera-
     tors
  5. To each individual in population do
       a. In deployment phase only, apply operator $SOM_1$.
       b. In improvement phase only, apply $SOM_2$ followed by $SOM_3$.
       c. Apply mapping operator MAPPING to the solution network to assign each
          demand to its nearest vertex in the network.
       d. Apply fitness evaluation operator FITNESS to the solution.
       e. Apply insertion operator SOMDVRP.
  6. Save the best solution, and send it back to the company.
  7. Apply selection operator SELECT to the population.
  8. Apply elitist selection operator SELECT_ELIT.
  9. $g = g + 1$
  10. If $g = Gen$, activate the improvement phase.
  11. If $g = 2 \times Gen$, re-initialize parameters and activate the deployment phase.
End while
```

The deployment phase starts its execution with solutions having vertex coordinates randomly generated into a rectangle area containing the demands. The improvement phase follows the deployment phase. Once the improvement phase has finished, the algorithm restarts from the beginning. Each phase uses one or more SOM operators to dispatch the vertices around customers while preserving the network topology. A particular implementation point is that all the nearest point findings of the SOM operators are based on spiral search (see below). The main difference between the deployment and improvement phases is that the former is responsible for creating an initial ordering from scratch. The SOM process embedded in the deployment loop has a large neighborhood proportional to N_t. The improvement loop simply performs local improvements. The SOM processes have smaller neighborhoods and perform less iterations.

Three operators are based on the SOM algorithm. They are the three operators SOM_1, SOM_2, SOM_3 with their respective parameters given in Table 1. The parameter values are set according to Creput and Koukam (2008). Parameter $t_{\max}$ is the

Table 1 SOM parameters

Operator	α_{init}	α_{final}	σ_{init}	σ_{final}	t_{max}	$niter$	Applied to	Phase
1 SOM_1	0.5	0.5	$2 \times N_t/m_t$	4	$Gen \times niter$	$N_t/4$	Network	Deployment
2 SOM_2	0.5	0.5	10	4	$Gen \times niter$	N_t/m_t	Single vehicle	Improvement
3 SOM_3	0.9	0.5	$2 \times N_t/m_t$	$(2 \times N_t/m_t)/2$	$Gen \times niter$	1	Network	Improvement

maximum number of basic SOM iterations performed during a long decreasing run of *Gen* generations. At each generation, *niter* basic SOM iterations are performed and applied to a given individual. The initial and final intensities and neighborhood sizes for the learning law are respectively given by parameters α_{init}, α_{final} and σ_{init}, σ_{final}. In a dynamic setting, *niter* and other parameters may depend on the available demands N_t and vehicles m_t at a given optimization time-slice. The SOM_1 operator is used to deploy the network from scratch in the deployment phase. The two other operators are applied during the improvement phase. The SOM_2 operator is applied to a single vehicle to help eliminate the remaining crossing edges from the tour. The SOM_3 operator introduces punctual moves in the network to help exit from local minima.

The long SOM decreasing run is interrupted and combined with the application of other operators. Such operators are the followings:

- Mapping/assignment operator. This operator, denoted *MAPPING*, generates a VRP solution by inserting customers into routes and modifying the shape of the network accordingly, at each generation. The operator greedily maps customers to their nearest vertex by spiral search. The corresponding vehicle capacity constraint must be satisfied. Then, the operator moves vertices to the location of their assigned customer (if exist). It also regularly dispatches (by translation) other vertices along edges formed by two consecutive customers in a route.
- Fitness operator, denoted *FITNESS*. Once the assignment of customers to routes has been performed, this operator evaluates a scalar fitness value for each solution. The value returned is $fitness = sat - 10^{-5} \times Length$, where *sat* is the number of customers successfully assigned to the routes, and *Length* is the length of the routes defined by the ordering of such customers. The value *sat* is considered as the first objective whereas *Length* is the secondary objective. Note that the customer waiting time is not introduced in the fitness function. Here, it is an external criterion in order to gauge dynamicity. It is indirectly addressed by the whole algorithm.
- Insertion operator, denoted *SOMDVRP*. It deals specifically with the time duration constraint. It performs few greedy insertions to the routes at each generation. Given a customer not yet inserted, the operator selects a nearest vertex which satisfies the time duration constraint and which produces minimum route time increase.
- Selection operators. At each generation, the operator *SELECT* replaces $Pop/5$ individuals that have the lowest fitness in the population by $Pop/5$ individuals with the highest fitness. An elitist version *SELECT_ELIT* replaces $Pop/10$ individuals which have the lowest fitness by the single best individual encountered during the run.

5.3 Spiral search algorithm

By the evolutionary dynamics, the goal is to make the nearest point assignments coincide to the right assignment, which minimizes the objective and satisfies constraints. The algorithm can be seen as a massive and parallel insertion method to the nearest points. To perform closest point findings, we have implemented the spiral search algorithm of Bentley et al. (1980) based on a cell partitioning of the area. It performs an optimal nearest point search with expected $O(1)$ time complexity for uniform or bounded distributions, and $O(N)$ space complexity with N the number of points in the plane. A cell based decomposition of the area within $O(N^{1/2} \times N^{1/2})$ cells is performed during the initialization phase of the memetic algorithm. Each cell has a (non null) memory capacity large enough to contain the demands at that location. The memory is allocated once. The contents of the memory cells are updated each time a given operator (SOM or mapping) has to be applied. The network vertices are introduced into the cells and the subsequent at most $O(N)$ closest point findings are based on their content. Once a customer has been randomly selected, the process first consists in finding its cell location using its coordinates in the plane. Then, this cell and its surrounding cells contents are examined in a spiral way. If a network vertex is found in a cell, this vertex will be the closest one unless a nearest vertex is found in the immediate next stage of surrounding cells. In that case, the last vertex will be the closest one.

6 Experimental results

6.1 Experiments overview

We study the impact of the main algorithm parameters on length minimization and customer waiting time. Also, we report a comparative evaluation against other algorithms on a common benchmark set. The benchmarks are the 22 test problems originally proposed by Kilby et al. (1998). Few approaches addressed the dynamic VRP. The benchmarks were reused by Montemanni et al. (2005), Goncalves et al. (2007) and Zeddini et al. (2008) which report detailed numeric results. Kilby et al. (1998) do not report numeric results that could be reused for comparison. We should mention that the approach presented by Bent and Hentenryck (2003) addresses a slightly different version of the dynamic VRP. In this version, some stochastic information is available before the start of the system, and maximizing the number of serviced customers is the real-time criterion rather than minimizing the customer waiting time. In our version of the problem, the question is how to service all the customers as soon as possible, rather than how to service a maximum of customers before some real-time deadline. We will discuss this point in the next section. Here, a route time duration constraint is set relative to its schedule as stated by (3). No real-time bound is defined for the latest time a vehicle can return to the depot. Our version is intended to be very close to the static VRP. All demands necessarily have to be serviced. Furthermore, we assume no a priori knowledge about the stochastic distribution. The only way to anticipate future requests is by sampling the already received demands, rather than by sampling a known distribution of the stochastic variables.

The Kilby et al. (1998) benchmarks are derived from some very popular static VRP benchmark datasets, namely 13 problems are taken from Taillard (1994), 7 problems are from Christofides et al. (1979), and 2 problems are from Fisher et al. (1981). These problems range from 50 to 385 customers. The available times of the demands are generated with uniform random distribution and are spread throughout the working day. The number of customers can be inferred from the name of each instance. The maximum number of vehicles available is set to 50 for each problem. This setting guarantees that it is possible to serve all the demands for the problems considered. The original static problems are available at VRP Web http://neo.lcc.uma.es/radi-aeb/WebVRP/. In the site are reported the optimal solution values for the static case. The extended test cases to the dynamic VRP are available at the APES web page http://www.cs.st-andrews.ac.uk/~apes/apedata.html along with a description of the datafile format or at http://www.feu.de/WINF/inhalte/benchmark_data.htm.

The proposed memetic SOM was programmed in Java and has been run on a AMD Athlon 2 GHz computer. All the tests performed with the memetic SOM are done on a basis of 10 runs per instance. For each test case is evaluated the percentage deviation, denoted "*%Length*", to the best known route length, of the mean solution value obtained, i.e.

$$\%Length = (\text{mean } Length - Length^{*}) \times 100/\ Length^{*} \tag{7}$$

where $Length^{*}$ is the best known value taken from the VRP Web, and "mean *Length*" is the sample mean based on 10 runs. The average computation times are reported in seconds. The average customer waiting time (see (4)) is expressed as a fraction of the working day by

$$\%WT = \text{mean } WT \times 100/D \tag{8}$$

with "mean *WT*" being the average waiting time based on 10 runs.

6.2 Influence of the main parameters

In this section, we study the influence of three important parameters and their impact to the length objective and waiting time tradeoffs. We also study correlation with the maximal vehicles arrival time to the depot, also called makespan, which represents the latest time a vehicle returns to the depot once all demands have been serviced. The parameters studied are the population size *Pop* of the metaheuristic, the computation time allowed by the choice of the basic temporization *To*, or timer-clock, and the delay imposed to the vehicle starts in order to simulate different degrees of dynamism. To implement a high degree of dynamism, vehicles immediately start at time 0. They try to service customers as soon as possible. To implement a medium degree of dynamism, a very simple waiting strategy is adopted. Vehicles start their service at the half of the working day, i.e. at time $D/2$. In that way, half of the demands in average are received before the vehicles start. The computation times are fast or long depending on the choices $To = 30$ ms or $To = 200$ ms to simulate a given time slice of a working day. Three population sizes are considered: $Pop = 1$, $Pop = 10$, and $Pop = 50$.

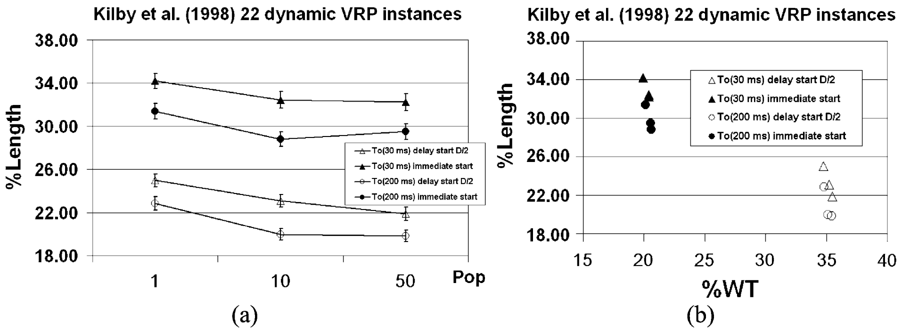


Fig. 6 Influence of population size, delay start-time, and computation time

The experiments were done with the 22 dynamic instances of Kilby et al. (1998) performing 10 runs by instance and reporting the average lengths and average customer waiting times. The results are presented in Fig. 6. Figure 6(a) presents the impact of the population size *Pop* parameter to the length objective. The results are presented with error bars representing 95% confidence intervals for the sample mean. Confidence intervals were computed on the basis of standard deviations over the 10 runs and the 22 instances. The way of computing confidence intervals is detailed in Creput and Koukam (2008). In Fig. 6(b) are shown the length and waiting time tradeoffs. They are represented by dots in the figure. They correspond to the twelve algorithm configurations executed on the benchmarks and presented in (a).

The results presented in Fig. 6(a–b) show that each parameter may have a significant impact on the length objective. Results can be grouped within two classes depending on the vehicle delay starts. The customer waiting time clearly depends on the time the vehicles start. It is near 20% and 35% of a working day for respectively high and medium dynamism. As seen in (a), an augmentation of the computation time and of the population size lead to a significant improvement on length minimization. As seen in (b), the population size or the computation times allowed have no impact on the customer waiting time. On the one hand, these experiments illustrate the importance of a diversification mechanism implemented by a population search and the significant impact of the computation time to length minimization. On the other hand, the customer waiting time mainly depends on the delay start time strategy.

Considering a long run with $To = 200$ ms and a population size of 50 solutions, Fig. 7(a) illustrates the tradeoff between length and waiting time as a function of the vehicle delay start time. With a delay start time from 0 to $D/2$, and hence to D, the (linear) decrease of the route length corresponds to an augmentation of the waiting time. It varies from 21% to 35%, and hence to 74%, of a working day respectively. With a delay start of D, all the demands are available when the vehicles start. Augmenting the delay start time from D to $2D$ simulates an augmentation of the computation time allowed for optimization before the vehicles start. The large 7.51% deviation observed over the known optima for the $2D$ start case may be explained by the large number of vehicles specified in the Kilby et al. (1998) benchmarks. The number of vehicles was set to 50 vehicles by Kilby et al. for all the benchmarks in

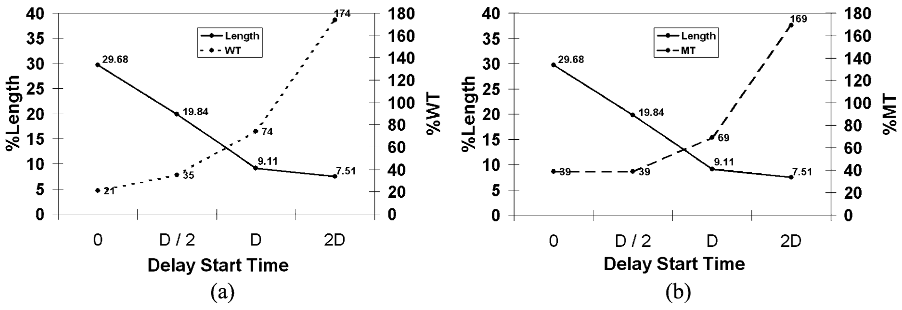


Fig. 7 Routes length, waiting times, and maximum arrival time as a function of the system dynamism

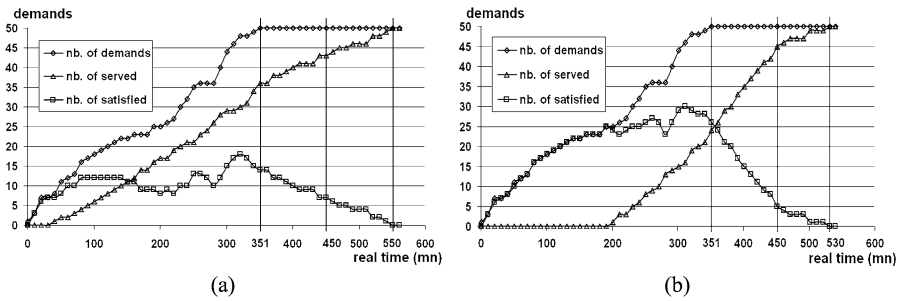


Fig. 8 Trace analysis on the c50 test case. (**a**) High dynamism. (**b**) Medium dynamism

order to ensure feasibility of solutions in a dynamic context. The original benchmarks from the VRP web that were adapted by Kilby et al. specify a smaller number of vehicles. A better performance is reported by Creput and Koukam (2008) on such original VRP Web static problems. In the Fig. 7(b), the maximum vehicles arrival time *%MT* is reported instead of the waiting time. It defines the moment when the last vehicle arrives back to the depot once all demands have been serviced. The maximum arrival time, or makespan, is expressed as an excess deviation to the working day by $\%MT = (\text{mean } MT - D) \times 100/D$, where MT is the latest time a vehicle arrives to the depot. We can observe on the Fig. 7(b) that with a delay start time of 0 or $D/2$, the makespan remains constant at 39% behind a working day. This illustrates the importance of reporting the customer waiting time in order to discriminate systems with different degrees of dynamism. A system can be more or less dynamic depending on the internal waiting strategy that is adopted. Compressing the route length by using a waiting strategy allows more customers to be serviced within a given deadline. Here, a time-duration deadline exists by the constraint in (3) but it is relative to the vehicle schedule and not to the absolute arrival time.

The execution traces presented in Fig. 8(a–b) illustrate how the service of demands behaves as the time goes on for the two cases of dynamism. Immediate vehicle start simulates a high dynamism in (a), and half of the day start simulates a medium dynamism in (b). The problem considered is the c50 test case with a working day of $D = 351$ time units. In (a–b) are shown, as a function of time the cumulative number of demands, the number of served demands, and the number of demands already scheduled in routes, called “nb of satisfied”. We can see that vehicles perform their services in a shorter period of time in (b). They finish at time 530 in (b) whereas at time 550 in (a). Drivers start their work later but finish their service earlier. At time 351, the immediate start strategy has serviced more customers than the delay start strategy. At time 450 it is the opposite. The delay start strategy has serviced more customers. But this should not hide the increase of the customer waiting time. Here, the customer waiting time is the important criterion in order to gauge the real-time effectiveness of algorithms. It defines the extension of the standard VRP to a dynamic setting.

6.3 Comparative evaluations

We report detailed results of the experiments performed on the Kilby et al. (1998) benchmarks in Table 2 and Table 3. The first column “Name” indicates the name and size of the instance. The second column “D” is the working day length, and the third column is the best known length value obtained for the static problem. Results are given within three columns. The columns “%Length” and “%WT” are respectively defined by (7) and (8). The column “Sec” reports the computation times in seconds. The working day is compressed into a small period of computation time specified by the *To* parameter. Two algorithm configurations are respectively considered with fast ($To = 30$ ms) and long ($To = 200$ ms) computation times. The metaheuristic population size was set to $Pop = 50$. Some authors (Bent and Van Hentenryck 2004; Kilby et al. 1998) discard computation time in a dynamic setting because they consider the working day as relatively long. They assume the availability of a large amount of computation resources. On the contrary, we consider the computation time as an essential resource to gauge effectiveness of algorithms, as usual in complexity analysis of algorithms. In a dynamic setting, the amount of computation time spent may illustrate the “anytime” nature of the algorithm. Furthermore, we could imagine future situations where computation resources become critical, may be with embedded systems where energy consumption must be minimized, for example in specific hostile environments where the central station would have to deal with many processes running simultaneously, or simply because of an increase of the problem size.

In Table 2, the memetic SOM is compared with the ant colony approach of Montemanni et al. (2005) and to the genetic algorithm of Goncalves et al. (2007). The authors have used the same benchmark set without the largest test case named tai385, and using a medium degree of dynamism. As explained by the authors, they implement a medium degree of dynamism by considering half of the demands as known in advance. It is worth noting that the authors do not report the customer waiting time. Hence, we have tried to closely follow their experimental setting in order to assume, without the possibility of proving it, a similar customer waiting time for the different approaches. Here, a medium degree of dynamism is achieved by delaying the vehicle starts to the half of the working day. Since the time distribution is uniform, half of the demands are then expected to be known beforehand. As shown in Table 2, the memetic SOM outperforms both the ant colony approach and the genetic algorithm. It improves the solution quality using lesser computation time. Computation time can be roughly a hundred times lesser. In Table 3 are given the results obtained using a

Table 2 Comparative evaluation on the 22 instances of Kilby et al. (1998) with medium dynamism

Name	D	Best	Memetic SOM (fast start time $D/2$, Pop = 50)			Memetic SOM (long, start time $D/2$, Pop = 50)			Montemanni et al. (2005)		Goncalves et al. (2007)	
			%Length	Sec[a]	%WT	%Length	Sec[a]	%WT	%Length	Sec[b]	%Length	Sec[c]
c50	351	524.61	18.89	17	47	18.29	111	47	30.00		13.99	
c75	346	835.26	21.24	17	40	19.99	106	40	24.75		15.89	
c100	399	826.14	14.24	18	43	13.13	118	43	29.03		24.07	
c100b	468	819.56	9.61	20	35	10.32	129	35	24.95		13.60	
c120	794	1042.11	7.00	30	30	7.73	196	30	46.34		37.22	
c150	399	1028.42	34.80	18	38	29.57	119	38	41.58		30.59	
c199	399	1291.29	30.16	18	37	29.40	112	36	42.88		30.18	
f71	211	237	31.78	10	51	27.99	64	52	47.26		19.41	
f134	11741	11620	47.21	49	21	43.55	316	19	38.42		35.29	
tai75a	769	1618.36	16.87	17	32	16.78	112	33	20.18		15.18	
tai75b	905	1344.62	25.53	19	31	22.51	129	32	26.73		13.86	
tai75c	782	1291.01	18.98	17	37	17.95	112	37	28.12		25.64	
tai75d	789	1365.42	16.77	18	34	13.83	115	35	11.98		10.22	
tai100a	897	2041.34	15.81	41	38	10.99	263	38	18.94		18.65	
tai100b	799	1940.61	15.36	37	40	13.89	247	40	20.99		15.48	
tai100c	905	1406.2	22.60	35	27	21.76	224	27	17.76		24.02	
tai100d	782	1581.25	24.19	34	33	21.85	228	34	30.34		20.66	
tai150a	1062	3055.23	19.02	46	36	15.62	298	36	25.69		20.51	
tai150b	988	2656.47	20.43	40	33	17.98	259	34	25.24		24.49	
tai150c	1081	2341.84	22.68	43	28	18.67	281	29	28.79		24.43	
tai150d	1025	2645.39	18.89	43	37	17.37	283	36	21.12		20.55	
ta385	4816	24431.44	29.45	142	31	27.37	629	30	–		–	
Average without tai385			21.53	28	36	19.84	182	35	28.62	1500	21.62	1500
Average all			21.89	33	35	19.48	202	35				

[a] Time per run in AMD Athlon (2 GHz) seconds, Java program
[b] Time per run in Pentium IV (1.5 GHz) seconds, C program
[c] Time per run in Pentium IV (2.4 GHz) seconds, Java program

Table 3 Evaluation on the 22 instances of Kilby et al. (1998) with high dynamism

Name	*D*	Best	Memetic SOM (fast, immediate start, Pop = 50)			Memetic SOM (long, immediate start, Pop = 50)			Zeddini et al. (2008)
			%Length	Sec[a]	%WT	%Length	Sec[a]	%WT	%Length
c50	351	524.61	47.41	18	30	46.91	121	31	49.06
c75	346	835.26	35.93	17	26	32.62	112	26	32.77
c100	399	826.14	29.35	18	31	24.44	120	32	60.02
c100b	468	819.56	22.61	20	19	17.39	128	20	–
c120	794	1042.11	13.11	30	16	12.54	195	16	57.76
c150	399	1028.42	49.67	19	28	45.99	122	28	–
c199	399	1291.29	47.92	18	26	43.71	113	25	–
f71	211	237	30.25	10	30	43.04	61	27	67.09
f134	11741	11620	67.79	48	5	62.09	316	5	49.63
tai75a	769	1618.36	24.45	17	21	20.91	111	22	24.94
tai75b	905	1344.62	26.49	20	13	23.06	129	14	33.27
tai75c	782	1291.01	29.15	17	20	27.33	107	21	–
tai75d	789	1365.42	16.86	18	20	20.27	111	19	–
tai100a	897	2041.34	24.39	41	21	20.70	262	21	23.50
tai100b	799	1940.61	16.78	36	23	16.77	235	23	24.91
tai100c	905	1406.2	37.44	36	12	27.88	234	13	–
tai100d	782	1581.25	40.78	35	20	35.73	219	21	–
tai150a	1062	3055.23	27.95	45	19	23.62	300	19	30.46
tai150b	988	2656.47	26.44	40	19	21.55	256	19	34.09
tai150c	1081	2341.84	35.45	43	13	30.46	283	14	–
tai150d	1025	2645.39	24.27	43	22	22.26	277	22	–
tai385	4816	24431.44	34.33	101	16	33.65	649	15	–
Avg. Zeddini et al. (2008)			30.86	28	21	29.02	185	21	40.63
Average all			32.22	31	20	29.68	203	21	

[a] Time per run in AMD Athlon (2 GHz) seconds, Java program

high degree of dynamism, which is implemented by vehicles immediate start. For this degree of dynamism, we only found the multi-agent approach of Zeddini et al. (2008) to which we can compare our method. In Zeddini et al., the benchmarks are used as they are stated with arrival dates along the day, with no hypothesis about the degree of dynamism or about requests known in advance. Vehicles are supposed to start as soon as possible as the day begins. This approach looks not competitive. Their average results on a subset of the benchmarks are reported at line named "Avg. Zeddini et al. (2008)" in the table and can be compared with our results. The authors do report neither the computation time nor the customer waiting time. This illustrates the need of a standard definition of the VRP when considered in a dynamic setting. Results on the last line of Table 3 are given to allow future comparisons with further approaches on the complete benchmark set.

6.4 Discussion

This section is intended to highlight some characteristics of our SOM based approach that must explain its better performances with regards to the other approaches considered in the paper. First, we need to consider the performances in relation to a static context, since finding good solutions quickly looks important in a dynamic setting. Then, we will discuss internal characteristics of the different approaches. It is generally admitted (Helsgaun 2000) that performances of heuristics often depend on many implementation tricks specifically designed to the problem in consideration. As stated in the survey of Cordeau et al. (2005), this is true for the static VRP. For example, the Active Guided Evolution Strategy (AGES) of Mester and Bräysy (2005) seems to be the overall winner heuristic considering both solution quality and computation time. But it is considered as very complicated. At the opposite, the Unified Tabu Search Algorithm (UTSA) of Cordeau et al. (2001) is considered as very simple and flexible but less performing. We have shown in Creput and Koukam (2008) that our memetic SOM looks competitive with UTSA on the static VRP. Furthermore, it was designed to naturally operate in a dynamic setting, with slight modifications. Then, a question is how the most powerful heuristics of operations research for the static VRP would behave in a dynamic setting and how they should be adapted. We have not the response yet on concrete realizations.

We think that the few algorithms already applied to the dynamic VRP and considered in this paper are not state-of-the-art approaches to the static problem. Only the ant colony algorithm by Montemanni et al. (2005) adapted from the MACS-VRPTW (Gambardella et al. 1999) was previously applied to the more complex vehicle routing problem with time-widows (VRPTW). It is often presented as one of the most powerful heuristics to the static VRPTW but it was not previously applied to the simplest and classical static VRP. When we look at its logical structure, the algorithm consists in a construction phase performed by ants, followed by a local search procedure. The working day is divided into many time-slices. Each corresponds to a static VRP. We should note that the algorithm operates on a complete graphs of demands with a pheromone matrix of size $O(N^2)$, with N the number of demands. Arrival of a new demand, as well as removal of a serviced customer, imply to update these data structures with a computation time in $O(N)$. An ant also needs $O(N^2)$ time steps to build a solution sequentially and hence to insert a new demand in a route. The local search procedure is very simple. It iteratively selects a customer and tries to move it into another position within its route or within another route. Similar remarks hold for the genetic algorithm of Goncalves et al. (2007) and the multi-agent approach of Zeddini et al. (2008). We think that these approaches are implemented in a naïve way and not customized to be efficient methods on the VRP in a Euclidean setting.

By contrast, our SOM based method operates in the plane. Data structures are of size $O(N)$, as it is the cellular partition of the plane. When a new demand arrives, it is introduced in the system in constant time. The build of a tour may only takes $O(N)$ computation time on average, since based on spiral searches in the plane. In many cases, a single local deformation of a route would be sufficient to insert a new demand in a route. Hence, the memetic SOM generates candidate solutions very quickly. We think that these characteristics are the main reasons that explain its good performances within short computation time.

7 Conclusion

We have presented the dynamic VRP as a straightforward extension of the classical static VRP and a heuristic to address the problem. Based at the origin on the standard self-organizing map algorithm, the memetic SOM applies the concept of an independent graph representing routes, called the network, that adapts its shape to the demand. Most of the operations are based on nearest point findings performed by a spiral search algorithm, which works in constant time for bounded distributions. The network modifies its shape rather than its structure. The algorithm adapts the network size from time to time. Since the demands actually scheduled may reflect the underlying data distribution, new demands can be incorporated into the network very quickly. New insertions have a weak impact on the data structures. We think that the concept is simple to understand and implement. Also, we think that the algorithm may be a good candidate for parallel implementations. This should be the case, at the level of the population based metaheuristic and at the level of the cellular partition of the plane.

We have proposed a formulation of the dynamic VRP where length, customer waiting time, and computation time are the main criteria to gauge effectiveness of policies on this problem. This paper has reported evaluations to allow further comparisons on a standard test set. The results look encouraging in that the approach outperforms the heuristics of the literature applied on the benchmarks. Actually, only the available demands already in the system and not yet serviced participate to the optimization and forecast process. Further research should focus on a better exploitation of the past information received in order to sample the demand distribution and predict future demands. Since there are many different versions of the dynamic VRP, it would be of interest to better normalize the problem definitions and also the benchmarks in order to favor empirical evaluations and comparisons of algorithms. We hope that the reported results will serve as a basis for future evaluations and comparisons of methods on this standard dynamic vehicle routing problem.

References

Angeniol B, de La Croix Vaubois G, Le Texier JY (1988) Self-organizing feature maps and the travelling salesman problem. Neural Netw 1:289–293

Bent R, Van Hentenryck P (2003) Dynamic vehicle routing with stochastic requests. In: Proceedings of the 18th international joint conference on artificial intelligence, Acapulco, Mexico, pp 1362–1363

Bent R, Van Hentenryck P (2004) Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Oper Res 52:977–987

Bentley JL, Weide BW, Yao AC (1980) Optimal expected time algorithms for closest point problems. ACM Trans Math Softw 6:563–580

Bertsimas D, Simchi-Levi D (1996) A new generation of vehicle routing research: robust algorithms, addressing uncertainty. Oper Res 4:286–304

Christofides N, Mingozzi A, Toth P (1979) The vehicle routing problem. In: Christofides N et al (eds) Combinatorial optimization. Wiley, New York, pp 315–338

Cordeau JF, Gendreau M, Hertz A, Laporte G, Sormany JS (2005) New heuristics for the vehicle routing problem. In: Langevin A, Riopel D (eds) Logistics systems: design and optimization. Springer, New York, pp 279–297

Cordeau JF, Laporte G, Mercier A (2001) A unified tabu search heuristic for vehicle routing problems with time windows. J Oper Res Soc 52:928–936

Creput JC, Koukam A (2009) A memetic neural network for the Euclidean traveling salesman problem. Neurocomputing 72:1250–1264

Creput JC, Koukam A (2008) The memetic self-organizing map approach to the vehicle routing problem. Soft Comput 12:1125–1141

Creput JC, Koukam A, Hajjam A (2007) Self-organizing maps in evolutionary approach meant for dimensioning routes to the demand. In: Proceedings of the 21th international conference on computer, electrical, and systems science, and engineering, Vienna, Austria, May 25–27, pp 444–551

Durbin R, Willshaw DJ (1987) An analogue approach to the traveling salesman problem using an elastic net method. Nature 326:689–691

Fisher M, Jakumar R, van Wassenhove L (1981) A generalized assignment heuristic for vehicle routing. Networks 11:109–124

Gambardella LM, Taillard E, Agazzi G (1999) MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw-Hill, New York, pp 63–76

Gendreau M, Guertin F, Potvin JY, Taillard E (1999) Parallel tabu search for real-time vehicle routing and dispatching. Transp Sci 33:381–390

Gendreau M, Laporte G, Potvin J-Y (2002) Metaheuristics for the capacitated VRP. In: Toth P, Vigo D (eds) The vehicle routing problem. SIAM, Philadelphia, pp 129–154

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

Golden BL, Wasil EA, Kelly JP, Chao IM (1999) Metaheuristics in vehicle routing. In: Crainic TG, Laporte G (eds) Fleet management and logistics. Kluwer, Boston, pp 33–56

Goncalves G, Hsu T, Dupas R, Housroum H (2007) Plateforme de simulation pour la gestion dynamique de tournées des véhicules. J Eur Des Syst Autom 41:515–539

Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur J Oper Res 126:106–130

Kilby P, Prosser P, Shaw P (1998) Dynamic VRPs: a study of scenarios. Technical Report APES-06-1998, University of Strathclyde, UK

Kohonen T (2001) Self-organization maps and associative memory, 3rd edn. Springer, Berlin

Larsen A (2000) The dynamic vehicle routing problem. PhD thesis, Technical University of Denmark, Lyngby, Denmark

Larsen A, Madsen OBG, Solomon M (2008) Recent developments in dynamic vehicle routing systems. In: The vehicle routing problem: latest advances and new challenges. Springer, Berlin, pp 199–218

Mester D, Bräysy O (2005) Active guided evolution strategies for large scale vehicle routing problems with time windows. Comput Oper Res 32:1593–1614

Montemanni R, Gambardella LM, Rizzoli AE, Donati AV (2005) Ant colony system for a dynamic vehicle routing problem. J Comb Optim 10:327–343

Moscato P, Cotta C (2003) A gentle introduction to memetic algorithms. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer Academic, Boston, pp 105–144

Oja M, Kaski S, Kohonen T (2003) Bibliography of self-organizing map (SOM) papers: 1998–2001 addendum. Neural Comput Surv 3:1–156

Reinelt G (1991) TSPLIB-A traveling salesman problem library. ORSA J Comput 3:376–384

Taillard E (1994) Parallel iterative search methods for vehicle-routing problems. Networks 23:661–673

Zeddini B, Temani M, Yassine A, Ghedira K (2008) An agent-oriented approach for the dynamic vehicle routing problem. In: International workshop on advanced information systems for enterprises. IEEE Comput Soc, Los Alamitos. doi:10.1109/IWAISE