Chapter 14
From the TSP to the Dynamic VRP: An Application of Neural Networks in Population Based Metaheuristic

Amir Hajjam, Jean-Charles Créput, and Abderrafiãa Koukam

Abstract. In this paper, we consider the standard dynamic and stochastic vehicle routing problem (dynamic VRP) where new requests are received over time and must be incorporated into an evolving schedule in real time. We identify the key features which make the dynamic problem different from the static problem. The approach presented to address the problem is a hybrid method which manipulates the self-organizing map (SOM) neural network similarly as a local search into a population based memetic algorithm, it is called memetic SOM. The approach illustrates how the concept of intermediate structure provided by the original SOM algorithm can naturally operate in a dynamic and real-time setting of vehicle routing. A set of operators derived from the SOM algorithm structure are customized in order to perform massive and distributed insertions of transport demands located in the plane. The goal is to simultaneously minimize the route lengths and the customer waiting time. The experiments show that the approach outperforms the operations research heuristics that were already applied to the Kilby et al. benchmark of 22 problems with up to 385 customers, which is one of the very few benchmark sets commonly shared on this dynamic problem. Our approach appears to be roughly 100 times faster than the ant colony algorithm MACS-VRPTW, and at least 10 times faster than a genetic algorithm also applied to the dynamic VRP, for a better solution quality.

14.1 Introduction

The vehicle routing problem (VRP) is one of the most widely studied problems in combinatorial optimization. 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 duration constraint. In the static version of the problem, it is assumed that

Amir Hajjam · Jean-Charles Créput · Abderrafiãa Koukam
Laboratoire Systèmes et Transports, U.T.B.M., 90010 Belfort Cedex, France
e-mail: amir.hajjam@utbm.fr

E. Alba et al. (Eds.): Metaheuristics for Dynamic Optimization, SCI 433, pp. 309–339.
springerlink.com © Springer-Verlag Berlin Heidelberg 2013

all customers are known in advance to the planning process. However, many real world routing problems include some dynamic elements. The information data often tends to be uncertain or even unknown at the time of the planning. It may be the case that customers, driving times or service times, are unknown before the day of operation has begun, but become available in real-time. Due to the recent advances in information and communication technologies, such as geographic information systems (GIS), global positioning systems (GPS) and mobile phones, companies are now able to manage vehicle routes in real-time. Hence, with the increased access to these services, the need for robust real-time optimization procedures will be of critical importance, for small to big distribution companies, whose logistics are based on a high reactivity to the customer demand.

Depending on the application area, some authors propose to classify problems by their degree of dynamism and the objectives and constraints of the problem. As introduced by Larsen [26], the degree of dynamism may vary between 0 and 1, and has to be computed considering a given period of observation called a working day or planning horizon. The simplest measure is the ratio between the number of dynamic requests, which arrive during the working day, and the total number of requests including the static requests which are known in advance. For instance, if the degree of dynamism is 0.4, then 4 customers out of 10 arrive while the working day has begun. Other measures of dynamism are also introduced by Larsen in order to take into account both dynamic request occurrence times and time windows. In the standard dynamic VRP case, the author argues that a problem is more dynamic if immediate requests occur at the end of the working day and less dynamic as soon as immediate requests are received relatively early during the planning horizon. For example, two problem instances respectively with all arrivals at the beginning of the day or all arrivals at the end of the day have different degrees of dynamism, the latter being highly dynamic whereas the former being static. But we could argue that the crucial point is that customers would have to be served as quickly as possible regardless of the moment of the day they sent their demands. For example, the problem of fire fighting is highly dynamic whenever the disaster appears at the beginning or at the end of the day. Hence, it is necessary to look at the importance given to the real-time objectives and constraints.

Here, an ideal application field would be medical services. An application example would be medical interventions of doctors that require a high to moderate level of dynamicity, which would be constrained by the limited amount of doctors and resources available to perform the service. Hence, the degree of dynamism considered in this paper will be measured by a waiting time of roughly 15% to 30% of the working day. We assume that no information is available about requests locations prior to optimization. Only the overall capacity of the system and hence the total number of requests that the system could serve within a day are supposed to be known. We argue that this is a reasonable assumption since real-time performance mainly depends on the vehicle resources available.

In this paper, we propose a heuristic approach performing empirical evaluations by discrete time simulation on the standard benchmarks of Kilby et al. [24]. We shall focus on a hybrid method which follows metaphors of biologic systems, and hence naturally exhibits intrinsic parallelism and which may be considered intuitive and easy to implement. The approach presented in this paper is based on the concept of the "intermediate structure" pointed out by Glover [20] about applications of neural networks to optimization. The intermediate structure, called the network, represents transport lines that continuously distort and modify their shapes in the plane according to the demand distribution. Such paradigm that we called "adaptive meshing" in earlier papers [9, 13] partially has its origin from the Kohonen selforganizing map (SOM) [25]. Starting from the SOM, we define insertion and deformation operators that are applied to the network, and managed inside a population based metaheuristic similarly as in a memetic algorithm [29], which is an evolutionary algorithm embedding a local search process and mutation operators. The standard SOM is used similarly as a local search process combined with a mapping operator, responsible for the massive insertions of customers to the network, a fitness evaluation, a selection operator, and a specific operator dedicated to customer insertions according to the maximum duration constraint. Such operators perform elementary moves in the plane and a particular point is that almost all operators are based on the nearest point findings implemented on the top a cellular decomposition of the plane by a spiral search algorithm [1].

Successive generations of construction heuristics, improvement heuristics, and metaheuristics were developed by the operations research (OR) community to solve the static traveling salesman problem (TSP) [1] and the VRP [5, 18]. Metaheuristics often encapsulate a construction method followed by the application of one or more improvement heuristics performing local search. From stage to stage, such heuristics were enriched reusing the past enhancements to build new sophisticated neighborhood search structures, which operate on graphs. Then, a question is how the complex data structures of the very powerful OR heuristics for the TSP or VRP, often based on k-d-trees for nearest point search, managing neighborhood lists, "do not look bit" tables or various solution coding schemes, should be reused in a dynamic setting and be implemented in a distributed and parallel way. In our approach, there is no distinction between a construction phase and an improvement phase, as usual in metaheuristics, but rather a distinction between a deployment phases followed by an improvement phase with different intensities. The deployment phase is only responsible to deploy the network from scratch using a high intensity for the network moves. Tour construction and tour improvement are performed simultaneously at any moment, on both phases, based on closest point finding insertions and deformations. Hence, there is no need to introduce new insertion procedures to deal with the on-line arrivals of new demands, or to restart the algorithm, as we would do in order to apply traditional methods into a dynamic setting. As they arrive, new demands are simply inserted on-line in a buffer of requests, in constant time, leading to a very weak impact on the internal data structures and thus to the course of the optimization process. One of the goals of our work is to exploit the natural properties of the on-line SOM algorithm to be applied in a dynamic setting. An important

point of using an evolutionary framework is to allow simplicity and flexibility when designing the algorithm. We already applied the approach, which is called memetic SOM, to the static TSP [11], the static VRP [14] and VRPTW [10], and to several other extensions of these problems considering combination of clustering k-median and vehicle routing problems [11, 14]. Also, we link the approach to further possible parallel implementations by considering two strategic levels for the parallel computation: the population based "metaheuristic level" which corresponds to the cooperation of many autonomous local search processes, each one embedding a complete solution, and the "heuristic level" which corresponds to a cellular decomposition of the data, each cell possibly being associated with a processor and a part of the data. Also, the Euclidean nature of the problem is directly reflected into the Euclidean nature of the algorithm, thus allowing application to large instances, as this was done for the TSP [11] on instances with up to 85900 cities.

The following Section II states the dynamic VRP considered in this paper with its constraints and objectives and considered as a straightforward extension of the static VRP. Section III illustrates the philosophy of the intermediate structure concept on previous applications that were presented in earlier papers. We will find the details of the memetic SOM approach in Section IV. In Section V we shall present the realtime simulator able to gauge the efficiency of the method. Then, Section VI reports experiments carried out on the Kilby *et al.* benchmark and the comparisons made with a state-of-the-art ant colony approach and a genetic algorithm already studied on these benchmarks. It also presents a summary of the memetic SOM performances against classical heuristics on the TSP, VRP, and the dynamic VRP. The last section is devoted to the conclusion and further research.

14.2 Dynamic Euclidean Vehicle Routing Problem

As for static vehicle routing problems, a lot of versions of the dynamic problem exist depending on the application areas. For an overview and classification of the numerous versions of realftime routing and dispatching problems, we refer the reader to the general surveys and classifications given in [18, 19, 26, 31, 32]. One of the simplest versions is the standard dynamic VRP with capacity and time duration constraints [24], called "dynamic VRP" in this paper, which is a straightforward extension of the classical static VRP [3]. In this problem, the customers are the only elements who have a dependence on time. Customers 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. Problems fitting this model appear frequently in industry. Most parcel delivery services, replenishment of stocks in a manufacturing context, waste collection, and dispatch of emergency services can be modeled in this way. Geographically dispersed failures to be serviced by a mobile repairman also fit this model.

In the different versions of the dynamic VRP presented, there are a very few dynamic routing problems except the dynamic VRPTW or dynamic PDPTW that are recognized as standard problems well suited to allow comparative evaluations of heuristics and metaheuristics on a common set of benchmarks. For example, we only found two papers on the dynamic VRP that shared detailed results on a common test set. They are first an adaptation of the ant colony approach MACS-VRPTW [17] by Montemanni et al. [28], and second a genetic algorithm by Goncalves et al. [21]. They shared results on the Kilby et al. [24] test set with 22 problems of sizes from 50 with up to 385 customers. This paper tries to go one step further in that direction considering the dynamic VRP as a standard dynamic problem, and yielding a comparative study with these two methods on the Kilby et al. test set. Then, we restrict the scope of our work to the dynamic VRP, with capacity and time duration constraints.

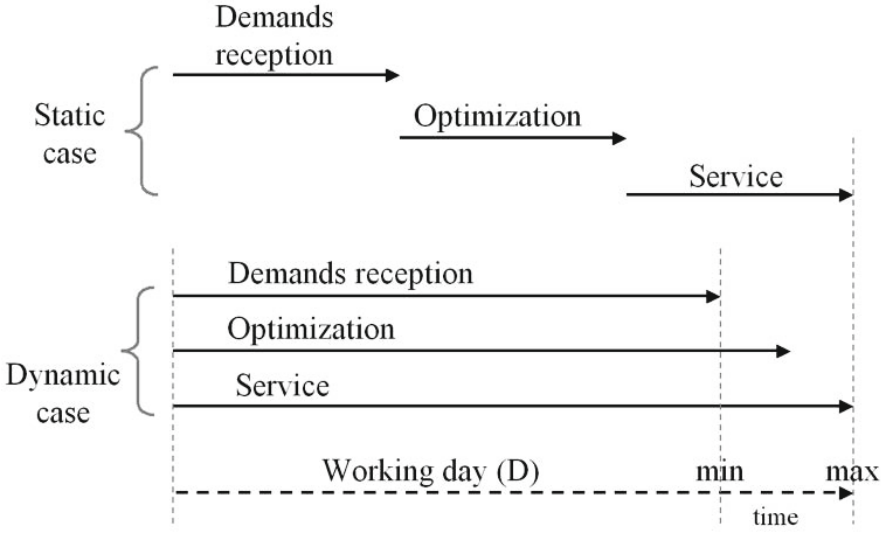


Fig. 14.1 Static vs dynamic VRP.

In the static VRP, vehicles must be routed to visit a set of customers at minimum cost, assuming that all orders for all customers are known in advance. In the dynamic VRP however, new tasks enter the system and must be incorporated into the vehicle schedules and served as the day progresses. In real-time distribution systems, demands arrive randomly in time and the dispatching of vehicles is a continuous process of collecting demands, forming and optimizing tours, and dispatching requests to vehicles in order to process requests at the required geographic locations. In the case of the static VRP, the three phases of demands reception, routes optimization, and vehicles traveling are clearly separated and sequentially performed, the output of a given phase being the input of the subsequent one. At the opposite, as

illustrated in Figure 14.1, we can see the dynamic VRP as an extension of the static VRP where these three time-dependent processes are merged into an approximately same period of time. This period of time is called the working day or planning horizon of length D. Here, we precisely define the working day length D as the length of the collecting period, knowing that the optimization period and the vehicle traveling period would have to be of approximately the same length.

The static VRP is defined on a set $V = v_0, v_1, ..., v_N$ of vertices, where vertex $v-0$ is a depot at which are based m identical vehicles of capacity Q, while 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. The VRP consists of designing a set of m vehicle routes of least total cost, each starting and ending at the depot, 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 mostly done in practice [5], we address the Euclidean VRP where each vertex vi 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 (Length) defined by

$$Length = \sum_{i=1,..,m} \left(\sum_{j=1,...,k_i-1} d\left(v_j^i, v_{j+1}^i\right) + d\left(v_0, v_1^i\right) + d\left(v_{k_i}^i, v_0\right) \right), \tag{14.1}$$

where $v_i^j \in V, 0 \leq j \leq k_i, 0 \leq ki \leq N$, are the ordered set of demands served by the vehicle i, $1 \leq i \leq m$, i.e. the vehicle route. The capacity constraint is defined by:

$$\sum_{j=1,...,k_i} q\left(v_j^i\right) \leq Q, i \in 1, ..., m \tag{14.2}$$

then, assuming without loss of generality that the vehicle speed has value 1 the time duration constraint is given by:

$$\sum_{j=1,...,k_i} s\left(v_j^i\right) + \sum_{j=1,...,k_i-1} d\left(v_j^i, v_{j+1}^i\right) + d\left(v_0, v_1^i\right) + d\left(v_{k_i}^i, v_0\right) \leq T, i \in 1, ..., m \tag{14.3}$$

The problem is NP-hard. Then, for large instances, using heuristics is encouraged in that they have statistical or empirical guaranty to find good solutions possibly for large scale problems with several hundreds of customers.

It is argued in the literature that in a real-life situation the objective function often consists of a mixture of customers, waiting time costs, or system response time, and travel (or routing) costs. The analysis made by several authors [2, 19] confirms that there is a trade-off between travel costs and system time in a dynamic routing system and that the travel costs can be reduced in return for an increase in the system time. In weakly dynamic systems the focus is on minimizing routing costs. On the other hand, when operating a strongly dynamic system, minimizing the expected system response time is the main objective. In dynamic settings, the waiting

time is often more important than the travel cost. We will then define the dynamic VRP as a multiobjective problem, since the "interesting" or "good" solutions are always a compromise between these two criteria. For example, the static problem case can be seen as a particular case of the dynamic problem where the waiting time is completely discarded, the optimization process starting whenever all requests to be served are known. We define the dynamic VRP as a bi-objective problem by adding to the classical objective and constraints of the standard VRP a supplementary objective which consists of minimizing the average 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):

$$WT = \sum_{i \in \{1,\ldots,N\}} W_i \Big/ N \tag{14.4}$$

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

In order to evaluate the customer waiting time we need to, not only consider travel distances and service times, but also consider the "real time" at which the service is really performed. Real time includes the possible extra times during which the vehicle may be waiting or driving back to the depot before some new requests are dispatched to it. Only the evaluation of (14.4) depends on a real-time realization. The evaluation of (14.1)-(14.3) only depends on the ordering of demands in a route, the same way as for the static VRP. We consider the waiting time as the essential criterion to gauge the dynamicity of the system. Hence, it is an important criterion to evaluate the effectiveness of algorithms on this problem.

14.3 Method Principle

In this section, we illustrate the "philosophy" of the neural network based approach proposed to address the Dynamic VRP. We present the main characteristics of the approach and explain why it may naturally deal with the dynamic and stochastic version of the VRP, without changing quite nothing in its implementation when passing from a static to a dynamic context. There is no need to introduce new insertion procedures or to design new mechanisms to deal with the on-line arrivals of new demands along the working day since the approach is already based on massive insertions to an "intermediate" independent structure representing routes. While the very powerful OR heuristics to the static VRP are often based on internal data structures difficult to implement and to modify dynamically, the advantage of our approach would be on the simplicity of updating the evolving internal data structures. Furthermore, we argue that a promising characteristic concerns its potential

for a parallel and distributed implantation on multi-processors or grid systems. We can distinguish two levels for the parallel computation that are, the heuristic level which deals with the problem dependent operations distributed in the plane and the metaheuristic level which deals with a population of independent local search processes.

14.3.1 Biologic Metaphor and the Intermediate Structure Paradigm

One way to explain the ”philosophy” of the approach may be by referring the reader to some well known concepts in the Artificial Intelligence domain like emergent computation, bio-inspired methods, and soft-computing concepts including neural network, evolutionary algorithms, or hybrid systems. The approach can be seen as following a biologic metaphor where customers constitute external stimuli to which a ”biologic organism”, the network of transport lines, may respond dynamically adapting its shape continuously to absorb, neutralize, or satisfy the external stimuli. More generally, we can exploit this metaphor to address a large class of spatially distributed problems of terrestrial transportation and telecommunications, such as facility location problems, vehicle routing problems, or dimensioning mobile communication networks [12, 13]. These problems involve the distribution of a set of entities over an area (the demand) and a set of physical systems (the suppliers) which have to respond optimally relatively to the demand. This optimal response constitutes the solution to the optimization problem. Thus, a distributed bioinspired heuristic to address such problems is a simulation process of such spatially distributed entities (vehicles, antenna, customers) interacting in an environment which produces the ”emergence” of a solution by the many local and distributed interactions. The approach presented involves an ”intermediate structure”, which is a network or a graph in the plane, representing the transport lines that continuously distort and modify their shapes according to a demand distribution. The important point is that tour construction and tour improvement operations are all based on massive and distributed insertions and line deformations. Customers are chosen randomly in the plane and are repeatedly presented online and many times to a simple insertion procedure based on nearest point search. The closest point search in the network structure is performed on the top of a cellular decomposition of the plane by a spiral search algorithm that is known to perform in constant time for bounded distributions [1]. This paradigm of ”intermediate structure” and quite ”instantaneous adaptation”, that we called ”adaptive meshing” in previous applications, has from a part its origin and inspiration from the Kohonen self-organizing map (SOM) neural network, which was applied to the TSP since a long time and which can address large size problems with up to 85900 cities [11]. The SOM algorithm is a neural network approach dealing, when applied in the plane, with visual patterns moving and adapting to distributed data. Its main ”emergent” property is to allow adaptation by density and topology preservation of a planar graph (the transport network) to an underlying data distribution (demand set). It can also be seen as a center based

clustering algorithm with topological relationships between cluster centers. Here, we generalize the SOM algorithm giving rise to a class of "closest point findings" based operators that are embedded into a population based metaheuristic framework. The structure of the metaheuristic is similar to the memetic algorithm, which is an evolutionary algorithm incorporating a local search [29]. The SOM is a (long) stochastic gradient descent performed during the many generations allowed, and used as a "local search" similarly as in a classical memetic algorithm. This is why the approach has been called memetic SOM in previous work and we will maintain the name in this paper. The approach follows two types of metaphors. It follows a self-organization metaphor at the level of the interacting problem components, or heuristic level, and an evolution based metaphor at the population based metaheuristic level. Since demands are conceptually separated from the routes representation, which is an independent network or graph in the plane which continuously adjusts itself to the data, this leads to a straightforward application from a static to a dynamic setting. As they arrive, new demands are simply inserted on-line in a buffer of demands, in constant time, leading to a very weak impact on the course of the optimization process.

Figure 14.2 (a-c) illustrates the application from a static to a dynamic setting. Figure 14.2(a) presents a bus transportation system where vehicle routes, modeled as paths with a common arrival point and depicted by lines in the figure, are adapted to a given distribution of customers and represented by dots in the figure. The application concerned a set of 780 employees of an enterprise located over a geographic area of $73km \times 51km$ around the towns of Belfort and Montbeliard in the East of France [8]. Vehicle routes and customers are shown juxtaposed in the figure over the underlying road network, represented by thin lines in the figure. The problem tackled, called VRP-Cluster, is a combination of the Euclidean k-median problem with a classical VRP. It consists of positioning bus stops, or cluster centers, according to customer locations (k-median problem) and simultaneously generating vehicle routes among bus-stops (VRP). Bus-stops define clusters where customers are grouped and to which they have to walk to take the bus. As illustrated in Figure 14.2 (b-c), application to a dynamic context mainly results from considering an evolving static VRP where the starting locations of the vehicles (the filled circles in the figure) evolve step by step as the vehicles move in the plane and perform their services along the working day. Hence, the system must monitor and update the vehicle locations, their capacities, and the vehicle travel durations on a rate defined by a decomposition of the day within many short time-slices. Figure 14.2 (b) presents a version of the dynamic case where vehicles perform the service as soon as possible, whereas Figure 14.2(c) presents a case where the vehicle starting times are slightly delayed giving rise to a longer horizon for route optimization, hence to longer vehicle paths in the figure. In this paper, we will experiment different degrees of dynamism and gauge different trade-offs between waiting time and length minimization by simply delaying the departure of vehicles.

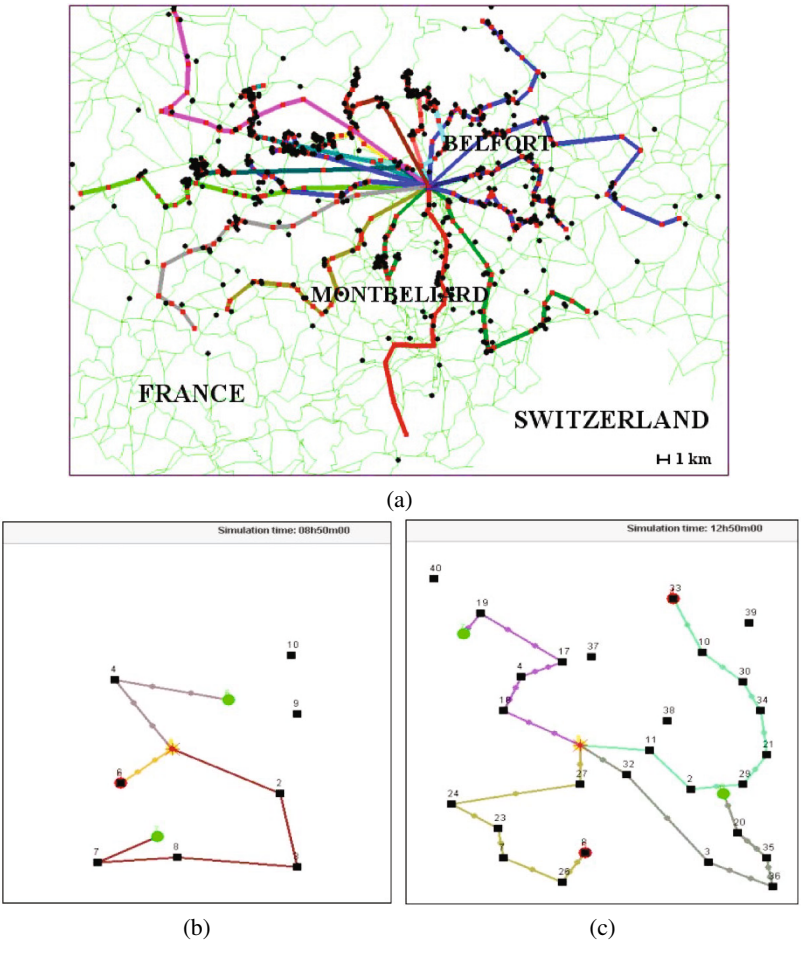


Fig. 14.2 (a) VRP-Cluster, (b-c) Dynamic VRP.

14.4 The Metaheuristic Embedding Framework

When introduced into a population based metaheuristic frame-work, the SOM is a long run process applied to a population of solutions. This process is interrupted at each cycle, called a generation, by the application of problem dependent operators. A generation occurs in such a way that at most $O(N)$ basic iterations are performed at each generation, N being the problem size. The main components of the method which are intended for driving the search are:

- a self-organizing map procedure based on closest point findings and route deformations as a low level stochastic process,
- problem-oriented insertion operators interleaving with SOM,

- a random perturbation with a decreasing intensity,
- a mapping operator which massively assigns requests to their closest vehicle route points,
- a fitness function incorporating constraints evaluation,
- a population based metaheuristic strategy with selection operators,
- a search performed within two phases (deployment, improvement).

The optimization process is divided within two phases, that are, a deployment phase followed by an improvement phase. It is worth noting that there is no distinction between a "construction" phase and an "improvement" phase, as usually done in OR metaheuristics, where the construction phase builds admissible solutions whereas improvement phase only improves the already constructed solution by swaps or exchanges of customers. Here, the distinction is only based on the intensity of the moves, since tour insertions and improvement operations operate simultaneously at any moment during the deployment and improvement phases. Hence, there is no need to add new specific insertion operators into the algorithm when passing from a static to a dynamic setting. The approach can be said "naturally on-line" and "intrinsically" customized for an application into a dynamic setting.

We claim that an interesting property of the approach is its intrinsic potential for parallel and distributed implementations in multi-processors, multi-core, grids, or P2P networks. The approach exhibits two strategic levels of parallel execution. On the one hand, we can exploit the "metaheuristic level" that corresponds to the cooperation of many autonomous local search processes, each one embedding a complete solution. It is worth noting that since the communication times at the level of the selection operators are relatively small, the long running times of independent local search processes favor parallel execution of the method. Also, this is why the optimization simulator has been structured as presented in Figure 14.4 (b). A population of agent-solvers embeds local search processes applied to the encapsulated solutions, whereas a meta-solver plays the role of a scheduler of the agent-solvers and applies selection operators to the population of agents. On the other hand, we could also exploit the "heuristic level" of the approach, which is problem dependent, and based on a cellular decomposition of the data. Each cell is associated with a part of the problem data and hence could be allocated to a given processor. Furthermore, this should favor the application to very large size problems for which the actual memory size of personal computers or workstations is notably insufficient.

14.5 The Evolutionary Algorithm Embedding Self-Organizing Maps

14.5.1 Memetic SOM

The approach is similar to a memetic algorithm [29], that is, a hybrid evolutionary algorithm embedding a local search. It is a simplified version of the approach presented in [14] which was applied to the static VRP. As illustrated previously by Figure 14.3(b), a population of Pop independent processes, called agentsolvers, perform

local search and other operations such as fitness evaluation and request insertions, each on a single encapsulated solution. A main loop, called metasolver loop, manages the scheduling of the local search processes and applies selection operators to the population of agent-solvers, i.e. of solutions. Each agent-solver exactly encapsulates one solution, which is defined by a graph of routes, called the network, where each route is represented by an independent path with $max(5N'/m, 5)$ vertices, m being the total number of vehicles available in the system, and N' the current number of available demands at time t, t being the current real-time. Each route starts at the vehicle location computed for the time $t + Topt$ and ends at the depot, $Topt$ being the optimization time-slice duration. Each route ends at the depot letting the vehicles go back to the depot each time they have no demand to be served in their schedule. The number of vertices by route corresponds to the maximum number of customers a vehicle can handle during a given optimization time-slice. It has been adjusted empirically to allow a good trade-off between the number of customers visited, equilibration of route lengths, and computation speed. The main loop manages two optimization phases, that are, a deployment phase followed by an improvement phase, each one executing a fixed number of iterations (called generations) which is set to the problem size N, N being the total number of demands received within a day. It is worth noting that the knowledge of the problem size N is considered in this paper as a reasonable assumption in order to adequately dimension the memory and computational resources. The meta-solver and agent-solver behaviors can be stated in pseudo-code as follows:

The deployment phase starts its execution with solutions having randomly generated vertex coordinates into a rectangle area containing the demands. The improvement phase follows the deployment phase. Then, once the improvement phase has finished, the algorithm restarts at the beginning. The main difference between the deployment and improvement phases is that the former is responsible for creating an initial ordering from random initialization. It follows that SOM processes embedded in the deployment loop have a larger initial neighborhood, proportional to N', N' being the number of demands that are currently in the system at the moment of parameters initialization. However, the improvement loop is intended for simply performing local improvements using SOM processes with smaller neighborhoods and applying fewer iterations. The parameter values of the SOM operators are set exactly as in [14], except that the tmax value and radius of neighborhoods α final and σ_{init} depend on the instantaneous number of available demands N' at the time of parameters initialization, rather than on the total number of demands N.

An important operator is the SOM algorithm. At each generation, a predefined number ($niter$) of basic SOM iterations, proportional to the current problem size N', are performed letting the long SOM decreasing run being interrupted and combined with the application of other operators. Such operators can be also a specialization of the SOM operator in order to perform request insertions, or to introduce perturbations, a mapping/assignment operator for generating admissible solutions, a fitness evaluation, and the selections at the population level. Below is a detailed description of the operators:

1. Self-organizing map operator. It is the standard SOM applied to the graph network.

   It is denoted by its name and its parameters, as
   $SOM(\alpha_{init}, \alpha_{final}, \sigma_{init}, \sigma_{final}, t_{max})$.

   A SOM operator is executed performing niter basic iterations by solution, at each generation. Parameter t_{max} is the number of iterations defining a long decreasing run ideally performed within N generations and applied to a given solution. When parameters initialization take place, it is stated as $t_{max} = N \times nite$ r, with $niter$ adjusted depending on the number of available demands as given in the pseudo-code above. Other parameters define the initial and final intensity and neighborhood for the learning law. The operator is used to deploy the network toward customers from scratch in deployment phase, or to introduce punctual moves to exit from local minima during the improvement phase.

Algorithm 14.1. Meta-solver main loop

```
Initialize population with Pop agent-solvers with routes randomly generated.
Initialize agent-solvers and their SOM parameters for the deployment phase.
Gen = 0
while not(a stop order is received from the company) do
    Look at the received messages from the company and update vehicles and request
    set according to the optimization protocol (see section 0).
    if a "request" order is received then
        add the new received demands to the end of the demand buffer
    end if
    if an "optimizer" order is received then
        update the vehicle locations, their capacities, travel duration, and route sizes,
        at the future time t + Topt, at the same time refresh the current request set
        according to the future time t + Topt
    end if
    Activate each agent-solver in turn, each one executing a single agent-solver
    generation
    Save the best solution encountered, and send it back to the company.
    Apply selection operator SELECT to the agent-solver population
    Apply elitist selection operator SELECT_ELIT.
    Gen = Gen + 1
    if Gen = N then
        swap agent-solvers and their SOM parameters to the improvement phase
    end if
    if Gen = 2N then
        Gen=0
        randomize population
        reset the best solution
        then swap agent-solvers and their SOM parameters to the deployment phase
    end if
end while
```

Algorithm 14.2. Agent-solver generation

1: In deployment mode only, apply a standard SOM operator, with parameters $(\alpha_{init}, \alpha_{final}, \sigma_{init}, \sigma_{final}, t_{max}) = (0.5, 0.5, max(2 \times N'/m, 5), 4, N \times niter)$, to the network, performing $niter = max(N'/4, 5)$ iterations.
2: In improvement mode only, apply the derived SOM operator, denoted SOMVRP, with parameters $(\alpha_{init}, \alpha_{final}, \sigma_{init}, \sigma_{final}, t_{max}) = (0.5, 0.5, 10, 4, N \times niter)$, to the network, performing $niter = max(N'/m, 5)$ iterations.
3: In improvement mode only, apply the derived SOM operator, denoted SOMVRP, with parameters
$(\alpha_{init}, \alpha_{final}, \sigma_{init}, \sigma_{final}, t_{max}) =$
$(0.9, 0.5, max(2 \times N'/m, 5), max(2 \times N'/m, 5)/2, N \times niter)$, to the network, performing $niter = max(N'/m, 5)$ iterations.
4: Apply mapping operator MAPPING to the solution network to assign each demand to its nearest vertex and move vertices to the demand locations.
5: Apply fitness evaluation operator FITNESS to the solution.
6: 5. Apply derived operator SOMDVRP, to perform greedy insertions of the residual demands according to the time duration constraint.

2. SOM derived operators. Two operators are derived from the SOM algorithm for dealing with the VRP. The first operator, denoted SOMVRP, is a standard SOM restricted to be applied to a single randomly chosen vehicle/route at each generation, using customers already inserted into the route. It helps to eliminate the remaining crossing edges in routes. While capacity constraint is greedily tackled by the mapping/assignment operator below, the second operator, denoted SOMDVRP, deals specifically with the time duration constraint. It performs few greedy insertion moves at each generation. Given a randomly chosen customer that is not yet already assigned to a vehicle, the competitive step selects to be the winner the vehicle vertex for which the route time increase is minimum, the route time duration constraint for that vehicle being satisfied. The evaluation of the route time increase is done moving the vertex to the customer location and including the customer into the route.
3. 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 first greedily maps customers to their nearest vertex for which the corresponding vehicle capacity constraint is satisfied, and to which no customer has been yet assigned. The capacity constraint is then greedily tackled by the customer assignment. Then, the operator moves the route vertices to the location of their assigned customer (if exist) and regularly dispatches (by translation) other vertices along edges formed by two consecutive customers in a route. The result is a vehicle route where assigned vertices alternate with the many more not assigned vertices. At this stage, few customers may not be inserted because of capacity constraint violation.

4. Fitness operator denoted $FITNESS$. Once the assignment of customers to routes has been performed, this operator evaluates a scalar fitness value for a given solution. This value has to be maximized, it is used by the selection operator at the population level. Taking care of time duration constraint, the fitness value is sequentially computed following routes one by one and removing a customer from the route if it leads to a violation of the time duration constraint. The value returned is $fitness = sat - 10 - 5 \times Length$, where sat is the number of customers that are successfully assigned to routes, and Length is the length of the routes defined by the ordering of such customers. The value sat is then considered as a first objective and admissible solutions are such that $sat = N'$, N' being the current number of customers in the system at a given optimization time-slice.
5. Selection operators. Based on fitness maximization, at each generation the operator denoted $SELECT$ replaces $Pop/5$ worst solutions, which have the lowest fitness values in the population, by the same number of best solutions, which have the highest fitness values in the population. An elitist version $SELECT_{ELIT}$ replaces $Pop/10$ worst individuals by the single best individual encountered during the run.

14.5.2 Spiral Search Algorithm

By the evolutionary dynamics, the goal is to make the closest point assignment coincide to the right assignment, which minimizes objectives and satisfies constraints. The algorithm can be seen as a massive and parallel insertion method to the nearest points. To perform N closest point findings in expected $O(N)$ time for uniform distributions, we have implemented the spiral search algorithm of Bentley, Weide and, Yao [19] 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, with $O(N)$ space complexity. Hence, a cell based decomposition of the area within $O(N1/2 \times N1/2)$ cells is performed during the initialization phase of the memetic algorithm. Each cell has a (non null) memory capacity proportional to an estimation of the number of 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. Vertices of the network are introduced into the cells and the subsequent (at most) $O(N)$ closest point findings will be based on their content. The cell contents are not updated after each move. This may introduce a relaxation on the requirement of finding the true nearest neighbor. But this drawback is balanced by the limited number of iterations performed and by the fact that vertex coordinates are modified after each move.

The choice of a spiral search algorithm based on a geometric partitioning of the area, rather than a standard k-d tree search method or a Delaunay-Voronoïmethod [30] was drawn from ”heuristic” arguments. We did not perform evaluations to yield a firm conclusion about the superiority of a method over another in the context of the SOM. Here, the closest point findings concern the network

vertices, rather than customers as usual for swapping operations in standard local search approaches. The insertions of the network vertices into the cells are to be done many times, more precisely, before each application of a given operator to a solution. The insertions take $O(N)$ computation time, for at most $O(N)$ closest point findings subsequently performed. Using a k-d tree would require $O(N \times log(N))$ computation time to build the balanced tree at each time. Thus, it is not clear whether the amortized computation cost would be inferior in that case. Also, we link the spiral search to further possible parallel implementations of the approach, in order to deal with very large instances for example. The k-d tree constitutes a hierarchical structure which is adequate in a context of shared memory. On the contrary, geometric partitioning according to a given topology may have some advantages when dealing with multiprocessor implantations. Here, a given cell would only have to "communicate" with its fixed 8 neighboring cells, each one being associated to a given part of the data and/or the network.

14.5.3 Algorithm Complexity

In our experiments, a given working day is divided into $O(N)$ time slices with constant computation time each, N being the problem size. Hence, the computation time allowed in experiments is $O(N)$. The number of generations performed, as the problem size grows, will depend on the complexity of the closest point findings based operators. With a constant population size, a SOM neighborhood proportional to N, and N basic iterations performed by generation, the time complexity to execute a given generation is $O(N^2)$ in the worst case. However, we claim that the spiral search mechanism considerably improves the nearest search. This point was confirmed empirically in [11], a linear time was achieved for constant neighborhood size operators applied on some unstructured TSPLIB test cases. The memetic SOM space complexity is $O(N)$, as usual for SOM. It is worth noting that this space complexity allows dealing with large size instances of several thousands of customers, on standard computer workstations.

14.6 Real-Time Simulation and Optimizer

This section presents the real-time simulator developed in Java which allows the dynamic solving of a dynamic VRP. To make things concrete, we assume that a transport company centralizes the optimization procedure, receives the orders from the environment, monitors the vehicle locations, and dispatches the continuously generated and optimized routes to the vehicles. Hence, we assume the existence of a communication system between the company, the customers and the drivers, and that communication times are negligible relatively to the rest of the real-time activities. In this section, we detail the simulator structure and the main parameters that will allow controlling the dynamic optimization process. It is worth noting that the simulator can be described independently of any optimization al-gorithm or policy that could be applied.

14.6.1 Simulator Architecture

The simulator consists of two main components implemented within two Java threads which 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, in order to simulate the vehicles activities implemented as simple state machines. The second thread plays the role of a background task which encapsulates the optimization process which continuously optimizes routes using the remaining CPU resources. During each time-slice, the optimization process solves a continuously evolving static VRP, with evolving vehicle capacities and starting locations, and with an evolving set of currently available requests. The idea has been discussed many times in the literature [21, 24, 26] and is clearly different from the rough strategy which consists in restarting the optimizer each time a new event occurs.

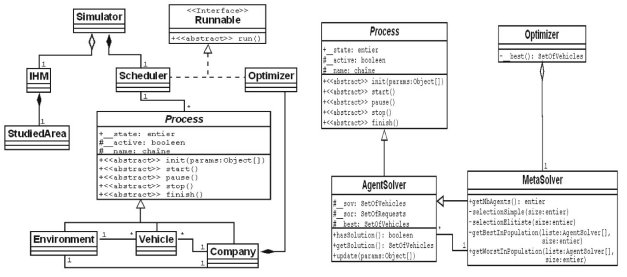


Fig. 14.3 Real-time simulator and asynchronous optimizer.

The architecture of the simulator is presented in Figure 14.3 in the UML class diagram style. Figure 14.3(a) presents the real-time scheduler while Figure 14.3(b) presents the structure of the optimizer. Three types of real-time processes are implemented and scheduled based on the timer clock. They are the Environment, Company, and Vehicle objects in Figure 14.3(a). The company is the center entity which receives demands from the environment, centralizes the de-mands, controls the optimization task, and dispatches orders to the vehicles. Figure 14.3(b) shows the structure of the asynchronous optimization process which manages a population of agents, called AgentSolver objects, that are responsible for generating solutions and constitute the population of the metaheuristic approach. Each agent solver then encapsulates a single evolving solution. A particular agent, called MetaSolver, plays the role of a scheduler of the agent solvers activities; it performs a selection between the solutions in a similar way of an evolutionary algorithm. Since this section is mainly devoted to the real-time simulation and not to the optimization algorithm,

we shall only focus here to the real-time aspects of the system and to the communication protocols. The solving policy and heuristics will be detailed further in the paper.

14.6.2 Asynchronous Protocol

On the one hand, the company receives new orders from the environment and communicates with the vehicles in a synchronous way, as both processes share the same real-time clock. On the other hand, the communication between the company and the optimizer is asynchronous, using mailboxes to exchange information. Here, the asynchronous execution mode of the optimizer is intended to allow the consumption of all the remaining available CPU resources. Figure 14.1 shows the structured information shared by the two asynchronous processes. The company controls the optimization process following a master/slave scheme. We can distinguish simple orders and structured orders sent by the company. Simple orders are the start, stop commands performed respectively at the beginning or at the end of a working day, and the dispatch of the new customer demands as soon as they arrive in the system. Structured orders concern the transfer of a complete solution, in the *SetOfVehicles* object, as well as, from time to time, the transfer of all the available requests once removing the ones already served, in the *SetOfRequests* object. Hence, the mailboxes have a size proportional to the number of requests in $O(N)$, N being the total number of demands arriving within a day.

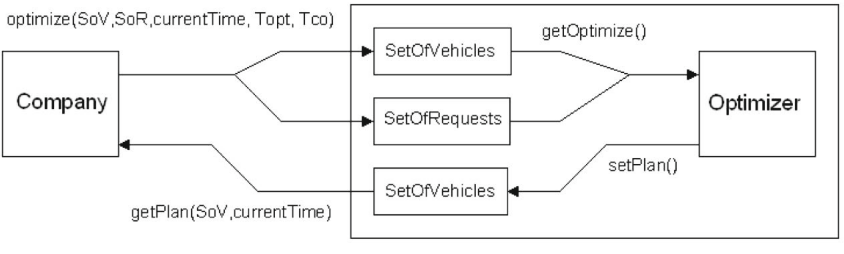


Fig. 14.4 Asynchronous data transfers using mailboxes.

It should be noted that the exchanges, when considering the direction from the company to the optimizer, mainly concern the updates of the vehicle locations, together with the arrival of the new requests. On the contrary, the exchanges from the optimizer to the company concern transfers of the built vehicle routes. But since it can occur in real life situations that autonomous vehicles could by their own modify their routes or themselves participate to the optimization process and modify their plans, we choose to implement a complete bi-directional exchange of vehicle routes. Then, the optimizer systematically chooses as a result the best solution between the received solution by the company and its best generated solution. However in this paper, the vehicles are supposed to strictly follow the optimized routes

provided by the optimizer, and not try to optimize the routes by themselves. Hence, the simulator has a general structure allowing possible distributed computations as done in some multi-agent approaches like [30], where the company and the vehicles simulate a market based protocol to construct the routes in a distributed way. Bidirectional exchange of vehicle routes is intended to allow further developments where a competitive solving could take place between the company and vehicles on the one hand, and the optimizer on the other hand.

In order to control the computation time allowed to simulate a working day, we decompose it into many time slices. Two parameters, denoted To and ToR next in the paper, define the amount of computation time allowed to simulate a basic time slice. They respectively control the compression of the working day into a small period of computation time, and the real-time precision of the system. Parameters To and ToR are respectively expressed in computation time units and real-time units. Since the arrivals of new requests have to be tackled as soon as they arrive, ToR can be seen as the basic unit of the real-time clock. It discretized the arrival of the re-quests along the working day. Here, this value is chosen to be in $O(D/N)$. Whereas, the To value corresponds to the few milliseconds of computation time allowed to simulate a period of ToR units of real-time. In the experiments presented in this paper, the ToR value will be adjusted to an integer value compatible with the benchmark test cases unit of time, taking ToR as the smallest integer greater than $0.1 \times D/N$. The To parameter will be adjusted to evaluate the performance of the system from large to very short computation time allowed, hence choosing To from $To = 200ms$ to $To = 20ms$.

The unit of real-time being defined, we now introduce the main parameters governing the frequency of the route updates between the company and the optimizer. They are the optimization time-slice Topt and the commitment horizon Tco, which are both expressed in real-time units. The optimization time-slice Topt defines the time between two consecutive route updates occurring between the company and the optimizer. The commitment horizon Tco is a period of time which defines the requests in routes that cannot be reallocated to other vehicle routes. The period starts from the current time, it is a commitment to the drivers that cannot be changed. We necessarily have $Topt \leq Tco$, and $ToR \leq Topt$. Each $Topt$ units of time, the company respectively gets back the new routes generated by the optimization process, and sends the current vehicle routes containing the actual vehicle positions. Hence, each time a route update is sent by the company at time t using the '"optimize()" procedure, the optimizer anticipates the vehicle positions at their future positions at time $t + Topt$. This is done within the "getOptimize()" procedure. Then, the optimization is performed with the anticipated solution at time $t + Topt$, when the solution will get back by the company using the "getPlan()" procedure. As well, the optimizer anticipates the vehicle positions at time $t + Tco$ since only the part of the routes behind this point can evolve through the optimization process. The optimizer regularly updates the mailbox with the new solution following its own internal rate Tg by calling the "setPlan()" procedure. The new requests are sent to the optimizer whenever they appear, using the "request()" procedure.

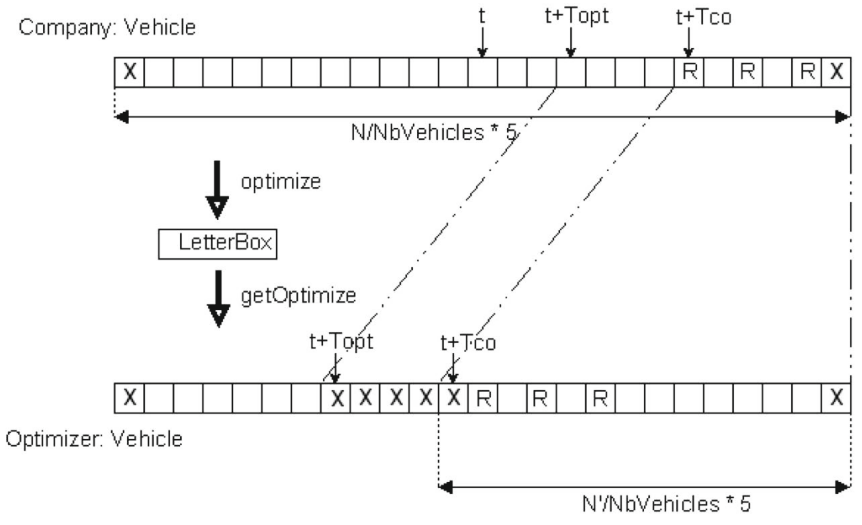


Fig. 14.5 Data structure for transfer from the company to the optimizer.

Figure 14.5 illustrates how the data structure representing a given vehicle route is affected by the transfer from the company to the optimizer. Since a route is defined as an ordered set of requests, the lookahead consists in finding the vehicle position at time $t + Topt$, and then fixing the requests from that date to the date $t + Tco$ since the committed requests will not be affected by the optimization procedure. It should be noted also in Figure 14.5 that the vehicle buffer size is adjusted each Topt to the value $5 \times N'/NbVehicles$, with N' the number of currently available requests in the system at time $t + Tco$, in order to be sufficiently large to insert new requests as they arrive during the next time-slice. The sizes of the Topt and Tco windows can be fixed independently from each other. But in a real time setting, and as showed in the Kilby *et al.* paper [24], the reactivity of the system drastically diminishes with the augmentation of the commitment horizon Tco. In this case, further requests will not be inserted into the committed portion of a route and then will be served later. In order to ensure a maximum of reactivity and dynamism of the system, we set $Tco = Topt$ in all the experiments presented in the paper, with Topt as small as possible. The Topt value is set to $Topt = 10 \times ToR$, thus taking an optimization time-slice in $O(D/N)$, allowing a single request occurrence on average for a single optimization time-slice.

14.7 Experimental Results

14.7.1 Experiments Overview

In this paper, we define the dynamic VRP as a straightforward extension of the static VRP. The length objective (1), the constraints of capacity (2) and time duration (3)

are defined exactly the same way as for the static case problem. This allows compatible evaluations according to static optimal values and allows a standard formulation of the problem. To take into account the degree of dynamism of the optimization process, a second objective is defined related to the real-time execution. It is the customer waiting time WT defined in (4). However, in order to give a supplementary insight into the real-time execution, an auxiliary criterion that we think useful to consider is the maximum vehicle finishing time MT defined by (5), i.e. the date of arrival at the depot of the last vehicle once all demands have been served. We think that this auxiliary criterion will help to gauge the excess part of the vehicle services performed behind the working day, once all the demands have been already received, and indirectly to evaluate the part of the instance that is solved as a static problem due to system congestion.

The proposed memetic SOM was programmed in Java and has been ran 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.

$$\%Length = (meanLength - Length*) \times 100/Length \tag{14.5}$$

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 also reported based on 10 runs. The average customer waiting time (4) and the maximum vehicle finishing time (5) are expressed as a fraction of the working day in order to compare data with different working days. The waiting time is expressed as a percentage of the working day length D by

$$\%WT = meanWT \times 100/D, \tag{14.6}$$

whereas the maximum finishing time is expressed as an excess deviation to the working day by:

$$\%MT = (meanMT - D) \times 100/D \tag{14.7}$$

This setting also guarantees that it is possible to serve all the demands for the problems considered. Finally, to make things concrete and realistic, the vehicle speed defined in the benchmarks of 1 distance-unit by 1 time-unit can be seen as a vehicle speed of 1 km/mn, or equivalently of 60 km/h. In order to be concrete, we will express the real-time in minutes and the distances in km when reported by their absolute values in some graphics. The working days are roughly between 4 hours and 17 hours, with an exception of a single test case having a 195 hours working day. It is worth noting that the parameter N and the total load of the demands are known before optimization in order to adequately dimension the system. Hence, the working day D can be decomposed into the many required time-slices. We assume that such values are necessarily known in advance in order to model a concrete real-life situation where a limited number of vehicles are intended to serve a maximum

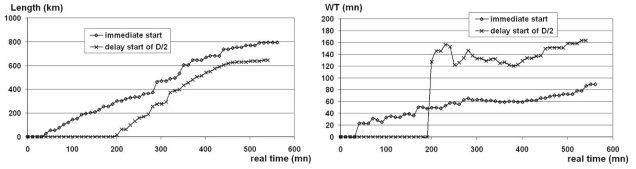
amount of demands, and to reasonably dimension the real-time simulator memory and the optimization system.

14.7.2 Influence of the Main Simulation Parameters

In this chapter, we apply to the dynamic VRP a simplified version of the memetic SOM algorithm that was studied in [14] for the static VRP. Since an analysis of the role of the operators and of the algorithm internal parameters was previously performed in the above mentioned paper, we will restrict the focus to an analysis of the few parameters that have an important impact to the dynamic and real-time implementation of the approach. In this section, we study the influence of three parameters and their impact to the length objective and waiting time trade-offs. 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, of the real-time simulator. The degree of dynamism is adjusted by simply delaying the starts of vehicles, from immediate starts (maximum dynamism) to an half of the day starts (medium dynamism). To respectively implement a high or medium degree of dynamism, delay start is expressed with two values: immediate start at time 0, or delay start at time $D/2$. A delay start at D or $2D$ is used to simulate a static VRP solver in a second set of experiments. The computation times are fast or long depending on the choices $To = 30ms$ or $To = 200ms$ to simulate a given time slice of a working day. Three population sizes are considered: $Pop = 1$, $Pop = 10$, and $Pop = 50$. The experiments were done with the 22 dynamic instances of Kil-by *et al.* performing 10 runs by instance and reporting the average lengths and average waiting times. These experiments are for a part reported with details in Table 1 and Table 2 next in the chapter.

14.7.3 Trace Analysis

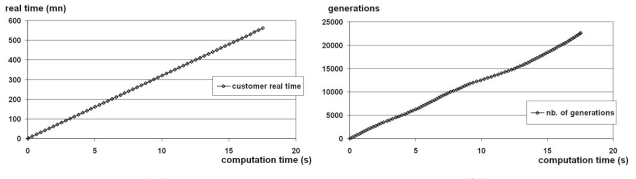
We analyze the trace execution of a typical simulation run in order to illustrate how real-time services are adequately simulated and performed along a working day, and verify that the CPU computation time is uniformly distributed along the day. The execution traces presented in Fig.6(a-b) illustrate how the route length and waiting time are evolving for the two cases of dynamism, that is, a maximum and a medium degree of dynamism, respectively, simulated by an immediate start of vehicles or a delay start of D/2. The problem considered is the c50 test case of the benchmark set, having a working day of $D = 351mn$, simulated with the timer-clock $To = 30ms$, and using a metaheuristic population size of $Pop = 10$. In Figure 14.6(a) is shown the length evolution, whereas in Figure 14.6(b) is shown the waiting time evolution for both cases of dynamism. The tradeoff between these two criteria suggests that one would have to choose between two incompatible scenarios: choose to drastically minimize the customer waiting time or choose to minimize length and drivers working time at an expense of the customer waiting time.



(a) (b)

Fig. 14.6 Trace analysis.

As well, in order to verify that the CPU computation time is uniformly distributed along the simulated working day, we report in Figure 14.7(a) the simulated real-time as a function of the processor time, measured by Java system calls every period of Topt units of real-time. The test case is the same as above with immediate vehicle starts and clock $To = 30ms$. The straight line obtained in the figure clearly shows that the computation time is uniformly distributed along the day. We also report in 14.7(b) the number of generations performed by the memetic SOM algorithm as a function of the measured computation time. Again, the number of generations performed at each time step looks roughly proportional to the computation time allowed, with smooth variations on the curve probably due to the varying problem size along the day, or the varying CPU consumption of the vehicles activities implemented as simple state machines.



(a) (b)

Fig. 14.7 Repartition of the CPU computation time.

14.7.4 Comparative Evaluation

We report detailed results of the experiments performed on the Kilby et al. [24] benchmarks in Table I and Table II. Here, such results are mainly given in order to allow further comparisons with heuristic algorithms for the dynamic VRP. In Table 14.1 are given the results when considering a high degree of dynamism implemented by vehicle immediate starts. Results in Table I are only given for further comparisons. In Table 14.2 , results are presented against the two other approaches found in the literature [21, 28] that have used the benchmark set with a medium degree of dynamism, considering that half of the demands were known in advance. It is worth noting that we simulate the same degree of dynamism by a vehicle delay start time at D/2. As we argued along this paper, we consider the degree of

dynamism as a property of the system rather than a property of the instance. The first column "Name-size" of each table indicates the name and size of the instance. The second column "D" indicates the working day length, and the third column the best known value obtained for the static problem. Then, results are given within five columns for a given algorithm configuration. The columns "%Length", "%WT", and "%MT" are respectively defined by equations 14.5, 14.6, and 14.7, as the percentage routes length, percentage average customer waiting time, and percentage maximum finishing time. The column "±%CI" is the 95% confidence interval for the routes length. Finally, the column "Sec" reports the computation times in seconds. Two algorithm configurations are considered respectively with fast ($To = 30ms$) and long ($To = 200ms$) computation times. The metaheuristic population size was set to $Pop = 10$.

Table 14.1 Evaluation on the 22 instances of Kilby et al (1998) with maximum degree of dynamism.

			memetic SOM (fast, immediate start, Pop=10)					memetic SOM (long, immediate start, Pop=10)				
Name-size-veh	D	Best	%Length	±%CI	Sec[a]	%WT	%MT	%Length	±%CI	Sec[a]	%WT	%MT
c50	351	524.61	44.77	3.04	17.56	26.15	59.86	48.03	3.24	111.57	26.78	56.24
c75	346	835.26	38.19	2.73	16.60	21.74	53.21	35.62	2.77	109.94	22.35	56.18
c100	399	826.14	44.21	2.79	18.05	22.11	44.51	37.63	4.17	118.37	24.42	45.84
c100b	468	819.56	33.06	5.21	18.89	16.73	29.00	24.74	4.18	124.04	17.35	30.30
c120	794	1042.11	23.41	1.71	29.21	13.25	17.61	20.70	1.89	190.21	12.92	17.83
c150	399	1028.42	62.88	2.50	17.85	19.57	42.88	58.29	1.47	119.71	20.44	47.49
c199	399	1291.29	60.70	3.17	17.05	18.34	36.49	55.30	2.04	111.65	18.37	37.54
f71	211	237	45.15	3.54	9.04	22.68	36.73	42.11	4.38	58.56	23.51	36.21
f134	11741	11620	75.03	7.59	47.22	3.41	15.75	62.65	4.24	316.19	3.81	19.27
tai75a	769	1618.36	24.72	2.17	16.47	17.72	36.80	20.13	2.91	107.14	18.25	36.96
tai75b	905	1344.62	28.35	3.69	18.87	10.98	33.26	26.55	7.73	123.77	11.23	34.48
tai75c	782	1291.01	31.13	2.68	16.69	15.17	36.34	27.08	1.67	108.72	15.60	36.68
tai75d	789	1365.42	25.06	3.21	16.26	15.84	31.63	20.50	3.76	106.81	16.44	33.08
tai100a	897	2041.34	28.13	2.92	39.59	16.23	41.15	27.23	2.75	255.73	16.26	40.27
tai100b	799	1940.61	30.29	4.59	36.02	18.77	44.17	25.41	2.12	234.20	19.73	44.21
tai100c	905	1406.2	45.29	4.24	34.33	10.16	21.29	35.12	4.10	232.15	10.88	26.20
tai100d	782	1581.25	43.13	3.59	32.62	14.14	33.35	40.86	3.99	213.99	14.72	34.62
tai150a	1062	3055.23	29.01	3.31	44.95	15.06	35.35	27.31	1.85	291.58	15.87	35.09
tai150b	988	2656.47	40.56	3.21	37.39	14.35	21.00	35.23	2.70	249.37	15.07	24.18
tai150c	1081	2341.84	52.64	3.97	41.45	10.55	22.63	46.53	2.45	275.21	11.28	25.26
tai150d	1025	2645.39	40.19	2.37	38.60	14.24	20.41	35.44	5.15	253.74	14.72	21.80
tai385	4816	24431.44	55.40	3.76	100.50	10.56	33.51	46.75	2.06	650.69	10.40	33.01
Average without tai385			40.28	0.70	26.89	16.06	33.97	35.83	0.69	176.79	16.67	35.12
Average all			40.97	0.68	30.24	15.81	33.95	36.33	0.66	198.33	16.38	35.22

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

When looking at the results of Table 14.1 and Table 14.2 , one should observe the different tradeoffs between route lengths (%Length) and waiting times (%WT), and note that the maximum finishing times (%MT) have similar values for both degrees of dynamism, corroborating the trace analysis made above. Then, a medium degree of dynamism will favor the drivers working period to be smaller, but at an expense of the customer waiting time. In Table 14.2, the approach is compared with an ant colony approach, that is, an adaptation of the well known MACS-VRPTW approach of Gambardella *et al.* [17] that is considered as one of the best performing approaches to the static VRP. The application to the dynamic VRP is due to Montemanni et al. [28]. Also, it is compared with the genetic algorithm of Goncalves *et al.* [21]. In Table II, the memetic SOM is compared with the ant colony approach of Montemanni *et al.* [28] and to the genetic algorithm of Goncalves *et al.* [21]. It is

Table 14.2 Comparative evaluation on the 22 instances of Kilby et al. (1998) with medium dynamism.

			memetic SOM (fast, start time D/2, Pop=10)					memetic SOM (long, start time D/2, Pop=10)					Montemanni et al. (2005)		Goncalves et al. (2007)	
Name-size-veh	D	Best	%Length	±%CI	Sec[a]	%WT	%MT	%Length	±%CI	Sec[a]	%WT	%MT	%Length	Sec[b]	%Length	Sec[c]
c50	351	524.61	18.58	2.48	16.52	45.49	50.37	17.82	2.14	110.82	46.45	55.19	30.00		13.99	
c75	346	835.26	24.59	2.88	17.00	39.59	56.99	22.65	2.20	106.01	39.53	50.58	24.75		15.89	
c100	399	826.14	14.33	2.68	18.44	42.14	47.69	13.39	2.06	118.60	42.13	46.12	29.03		24.07	
c100b	468	819.56	9.76	4.19	19.49	34.82	33.06	12.88	2.02	127.54	34.92	33.97	24.95		13.60	
c120	794	1042.11	9.47	2.22	29.33	29.26	18.11	7.95	2.62	189.07	29.32	17.12	46.34		37.22	
c150	399	1028.42	32.05	2.20	17.89	36.98	43.23	32.14	1.54	114.52	36.50	41.08	41.58		30.59	
c199	399	1291.29	33.23	1.72	17.22	35.71	37.89	30.29	2.18	113.69	35.35	40.05	42.88		30.18	
f71	211	237	30.89	3.99	9.73	48.76	47.20	27.03	2.94	63.19	49.07	47.01	47.26		19.41	
f134	11741	11620	53.14	8.37	48.38	18.12	18.60	46.98	5.34	318.72	18.59	20.21	38.42		35.29	
tai75a	769	1618.36	19.99	3.02	17.10	31.41	42.08	15.40	2.07	108.87	31.21	39.17	20.18		15.18	
tai75b	905	1344.62	23.93	4.01	18.76	30.75	32.49	22.54	3.69	123.42	31.07	34.08	26.73		13.86	
tai75c	782	1291.01	20.67	2.93	16.72	34.55	36.57	21.93	2.68	108.63	34.70	36.55	28.12		25.64	
tai75d	789	1365.42	17.26	2.94	17.13	33.10	38.71	14.89	3.28	112.53	33.54	40.20	11.98		10.22	
tai100a	897	2041.34	15.98	3.67	40.12	36.97	43.04	13.40	2.21	262.21	37.35	43.81	18.94		18.65	
tai100b	799	1940.61	17.13	2.78	36.35	39.01	45.48	15.19	2.13	239.28	39.05	47.32	20.99		15.48	
tai100c	905	1406.2	23.39	2.99	34.92	25.74	23.38	24.58	3.11	229.39	26.44	24.69	17.76		24.02	
tai100d	782	1581.25	28.21	2.73	33.90	31.78	38.61	24.71	3.68	221.12	31.75	39.09	30.34		20.66	
tai150a	1062	3055.23	19.75	2.52	44.73	34.89	34.71	20.72	3.05	291.58	34.70	35.08	25.69		20.51	
tai150b	988	2656.47	21.36	3.22	38.83	32.79	25.69	20.89	2.38	254.67	32.69	26.81	25.24		24.49	
tai150c	1081	2341.84	21.30	1.44	42.40	27.87	25.40	18.61	2.01	277.16	27.53	26.14	28.79		24.43	
tai150d	1025	2645.39	20.16	3.06	40.44	35.30	26.15	16.22	1.87	269.60	35.20	29.40	21.12		20.55	
tai385	4816	24431.44	38.03	2.21	95.76	29.33	27.03	38.86	1.65	623.00	28.64	27.33	-		-	
Average without tai385			**22.63**	0.63	**27.40**	34.52	36.45	**19.82**	0.53	**179.08**	34.62	36.84	28.62	1500	21.62	1500
Average all			23.33	0.66	30.51	34.29	36.02	20.58	0.51	199.26	34.35	36.41				

[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.

worth noting that the authors do not report the customer waiting time. Nevertheless, we tried to follow the same experimental setting. 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, a medium degree of dynamism is achieved when half of the demands are considered as known in advance. 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 14.2, 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 an hundred times lesser.

Finally, we report in Figure 14.8(a) synthetic presentation of the evaluations previously performed with the memetic SOM in [11, 14], as well as of the ones of this paper, against state-of-the-art operations research heuristics considering the trade-offs between objective minimization and computation time. Starting to look at the figures from 14.8(d) and in reverse order to figure 14.8(a), the results are given from standard static routing problems to the already studied more complex dynamic VRP. The problems are respectively, the static TSP, the static VRP with capacity constraint only, the static VRP with time duration constraint, and the dynamic VRP, all problems being Euclidean problems. The aim is to suggest how a massive and distributed insertion method, originally based on the neural network self-organizing map algorithm, can be adequately applied in a dynamic setting and to complex problems in a way competitive with the very sophisticated operations research heuristics specifically dedicated to deal with a given problem at hand.

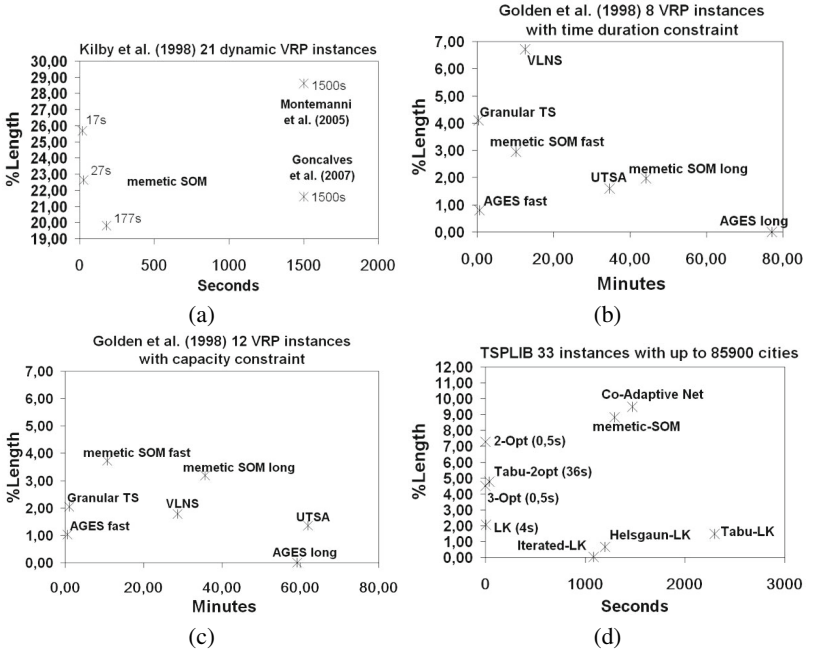


Fig. 14.8 Evaluation of the memetic SOM against state-of-the-art heuristics and metaheuristics. Dynamic VRP (a), static VRP with time duration constraint (b), static VRP with capacity constraint only (c), and static TSP (d).

Results in Figure 14.8(d) illustrate the performance of the memetic SOM on the 33 TSPLIB problems [33] of size larger than 1000 cities, with up to 85900 cities. It is worth noting that these problems were used in the last DIMACS challenge from which are reported the results, the computation times being normalized here to our AMD Athlon (2GHz) computer using Dongarra's factors [15]. Certainly, and as illustrated in Figure 14.8(d), neural networks based approaches do not compete with state-of-the-art OR heuristics for the TSP, such as the 2-Opt, 3-Opt and Lin and Kernighan local search heuristics, which are from a long time the best performing approaches to the TSP according to both length minimization and computation time spent. For example, referring to the Johnson and McGeoch paper [23], the new implementation of the Lin and Kernighan heuristic by Helsgaun [22] is clearly one of the most effective approaches for the TSP.

Nevertheless, and as it is the case also for the SOM based approach called Co-Adaptive Net algorithm [4], our approach was evaluated on many more test problems than previously considered in the literature for neural networks and more importantly on larger TSP's. To be competitive, solution quality produced by neural networks, as well as computation time, would have to be improved both by at least a factor of ten. Other OR powerful heuristics for the TSP are, in the most often

cases, a reuse of local search techniques embedded in a restarting or metaheuristic framework, as, for example, the Iterated-LK, Tabu-2-opt, or Tabu-LK, as given in the DIMACS challenge and reported in the Figure 14.8(d). However, the design and implementation of classical OR heuristics is not trivial since there are many implementation and design decisions to be made that have a great influence on performances. Here, on the contrary, we have focused on a heuristic which follows a metaphor in biologic systems, hence which exhibits a high degree of intrinsic parallelism and which may be considered intuitive and easy to implement.

In Figure 14.8(b-c) we turn to the static VRP, reporting results for the large size 20 test problems of Golden *et al.* [7] with sizes from 240 with up to 480 customers. The benchmark contains twelve problems with the capacity constraint only and height problems with the supplementary constraint of time duration. Comparison is presented against some of the recent heuristics presented in the survey paper [5] that cover the global range of metaheuristic performances for the static VRP. We used the numerical results reported in [5] with computation times normalized to our computer. The selected approaches are the Active Guided Evolution Strategy (AGES) [27], the Granular Tabu Search (GTS) [34], the Unified Tabu Search Algorithm (UTSA) [6], and the Very Large Neighborhood Search (VLNS) [16]. Two configurations of the memetic SOM "fast" and "long" are reported for respectively short and long computation times. From what we know, the AGES approach is, at the date of writing, the overall winner considering both solution quality and computation time for the static VRP. AGES is however considered complicated. On the contrary, UTSA is recognized to be simple (easy to understand and implement) and flexible (easy to extend) but more time consuming. Considering the instances with the capacity constraint in Figure 14.8(c), the memetic SOM is less efficient on accuracy than the other approaches, computation times being comparable or lesser than the ones of UTSA and VLNS. Considering the instances with the time-duration constraint in Figure 14.8 (b), the length value becomes closest to the one of UTSA for slightly spending more computation time. For such instances, GTS yields worst quality results but computes very quickly. The memetic SOM performs better than VLNS considering both quality solution and computation time. Hence, the more complex the problem becomes with new constraints added, the more competitive the memetic SOM becomes with OR powerful heuristics.

Finally, Figure 14.8(a) presents a summary of the experiments done on the dynamic VRP. The memetic SOM looks to be very faster and more efficient than the few approaches that were applied to the Kilby *et al.* benchmarks. The results are reported for different computation times allowed, set by a timer clock at respectively $To = 20, 30$, and 200ms, thus illustrating the "anytime" nature of the algorithm that is able to yield competitive results even for very short computation times.

As we explained in the introduction, the results corroborate the idea that the approach presented in this paper was from the beginning designed to be applied in a dynamic and Euclidean setting. Hence, the approach looks more simple and flexible than traditional OR approaches based on neighborhood swapping operators that need complex implementation tricks to yield their effective power [23] when applied in the Euclidean plane. Here, the performances can be explained by the many nearest

point searches performed in the plane in a distributed way by spiral search. Furthermore, there is no distinction between a construction and an improvement procedure as usual, but rather a distinction between a deployment and improvement phase where solution construction and solution improvement are a unified process with a decreasing intensity operating on an intermediate structure that continuously adapts and distorts itself in the plane to an underlying distribution of the demands. No new insertion procedure needs to be added to a previously designed improvement procedure in order to dynamically add the new demands in real-time, as usually done with improvement heuristics based on swapping operators that are subsequently applied to dynamic versions of a vehicle routing problem [24, 31].

Furthermore, the internal data structures of the memetic SOM are weakly impacted by the arrivals of new demands, an insertion into the memory being done in constant time $O(1)$. Subsequent insertions are then done naturally by the repeated massive and distributed nearest point searches. Similar implementation mechanisms look not to be considered in the ant colony and genetic algorithm when compared to our approach. For example, an ant would have to perform a complete tour in order to introduce a new demand in a route, thus theoretically performing $O(N^2)$ operations to do so. As well, adding a new demand in the memory theoretically needs modifying the graph structure by adding as many edges as necessary. And such conclusion arises also for the genetic algorithm where a complete examination of a solution-chromosome structure is needed to perform an insertion. The conclusion is that the ant colony algorithm and the genetic algorithm look far from being the best candidates for an application into a dynamic setting without saying anything about how their structures are adequately updated in the dynamic case.

14.8 Conclusions

We have presented the dynamic VRP as a straightforward extension of the classic and standard VRP, and a hybrid heuristic approach to address the problem using a neural network procedure as a search process embedded into a population based evolutionary algorithm, called memetic SOM. Based at the origin on the standard self-organizing map algorithm, the memetic SOM reuses the concept of an intermediate structure representing routes that adapt to an underlying distribution of demands by the many route distortions performed in the plane. By extension, these mechanisms become opera-tors in the population based metaheuristic. They lead to route improvements performed at the same time of customer insertions. Massive insertions are performed based on a spiral search algorithm for the nearest point search implemented on the top of a cell decomposition of the plane. That is why we think that the approach naturally deals with dynamic and real-time arrivals of demands distributed in the plane with a weak impact on the evolving structures.

This paper concludes a set of studies where the memetic SOM was successively applied to many routing problems. It was applied to the static TSP with problem sizes with up to 85900 cities, to the static VRP and the VRPTW, and to different combined bus-stop positioning and routing problems. While the approach looks far

from being competitive on the TSP when compared to the very powerful operations research heuristics, it becomes more competitive when considering more complex problems such as the static VRP with duration constraint, and specifically competitive when applied to the dynamic VRP. The results look encouraging in that the approach clearly outperforms the few heuristic approaches already applied to the dynamic VRP and evaluated in an empirical way on a common benchmark set. We claim that the memetic SOM is simple to understand and implement, as well as flexible in that it can be applied from a static to a dynamic setting with slight modifications. Also, we think that the memetic SOM is a good candidate for parallel and distributed implementations at different levels, at the level of the population based metaheuristic and at the level of the cellular partition of the plane.

Further research should focus on a better evaluation of the method against simple policies, or heuristics and metaheuristics approaches that were applied to more complex dynamic vehicle routing problems, such as the dynamic VRP with time-windows or the dynamic pick-up and delivery problem with time-windows. These approaches would be easily customized to the standard and simplest dynamic VRP presented in this paper. Hence, this paper has reported evaluations to allow further comparisons on the basis of a standard formulation of the dynamic VRP and a standard test set. It would be of interest to better study and normalize the dynamic and real-time benchmarks in a similar way that is done for the static problems, in order to favor future empirical evaluations of algorithms on dynamic unstructured large size problems.

References

[1] Bentley, J.-L., Weide, B.W., Yao, A.C.: Optimal expected-time algorithms for closest point problems. ACM Trans. Math. Softw. 6(4), 563–580 (1980)
[2] Bertsimas, D.J., Levi, S.D.: A New Generation of Vehicle Routing Research: Robust Algorithms, Addressing Uncertainty. Operations Research 44(2), 286–304 (1996)
[3] Christofides, N., Mingozzi, A., Toth, P.: The vehicle routing problem, pp. 315–338. Wiley (1979)
[4] Cochrane, E.M., Beasley, J.E.: The co-adaptive neural network approach to the euclidean travelling salesman problem. Neural Network 16(10), 1499–1525 (2003)
[5] Cordeau, J.-F., Gendreau, M., Hertz, A., Laporte, G.T., Sormany, J.-S.: New heuristics for the vehicle routing problem. In: Langevin, A., Riopel, D. (eds.) Logistics Systems: Design and Optimization, pp. 279–297. Springer, US (2005)
[6] Cordeau, J.-F., Laporte, G., Mercier, A.: A unified tabu search heuristic for vehicle routing problems with time windows. The Journal of the Operational Research Society 52(8), 928–936 (2001)
[7] Metaheuristics in Vehicle Routing. In: Crainic, T.G., Laporte, G. (eds.) Fleet Management and Logistics, pp. 33–56. Kluwer, Boston (1999)
[8] Creput, J.-C., Koukam, A.: Clustering and routing as a visual meshing process. Journal of Information and optimization sciences 28(4), 573–601 (2007)
[9] Creput, J.-C., Koukam, A.: Interactive meshing for the design and optimization of bus transportation networks. Journal of Transportation Engineering 133(9), 529–538 (2007)

[10] Creput, J.-C., Koukam, A.: Self-organization in evolution for the solving of distributed terrestrial transportation problems. In: Prasad, B. (ed.) Soft Computing Applications in Industry. STUDFUZZ, vol. 226, pp. 189–205. Springer, Heidelberg (2008)
[11] Creput, J.-C., Koukam, A.: A memetic neural network for the euclidean traveling salesman problem. Neurocomputing 72, 1250–1264 (2009)
[12] Creput, J.-C., Koukam, A., Hajjam, A.: Self-organizing maps in evolutionary approach for the vehicle routing problem with time windows. International Journal of Computer Science and Network Security 7(1), 103–110 (2007)
[13] Creput, J.-C., Koukam, A., Lissajoux, T., Caminada, A.: Automatic mesh generation for mobile network dimensioning using evolutionary approach. IEEE Trans. Evolutionary Computation 9(1), 18–30 (2005)
[14] Creput, J.-C., Koukam, A.: The memetic self-organizing map approach to the vehicle routing problem. Soft Computing - A Fusion of Foundations, Methodologies and Applications 12, 1125–1141 (2008)
[15] Dongarra, J.: Performance of various computers using standard linear equations software. Technical Report CS-89-85, Department of Computer Science, University of Tennesse, US (2006)
[16] Ergun, O., Orlin, J.B., Steele-Feldman, A.: Creating very large scale neighborhoods out of smaller ones by compounding moves: A study on the vehicle routing problem. MIT Sloan Working Paper No. 4393-02 (October 2002)
[17] Gambardella, L.M., Taillard, É., Agazzi, G.: Macs-vrptw: A multiple colony system for vehicle routing problems with time windows. In: New Ideas in Optimization, pp. 63–76. McGraw-Hill (1999)
[18] Gendreau, M., Laporte, G., Potvin, J.-Y.: Metaheuristics for the capacitated VRP, pp. 129–154. Society for Industrial and Applied Mathematics, Philadelphia (2001)
[19] Ghiani, G., Guerriero, F., Laporte, G., Musmanno, R.: Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. European Journal of Operational Research 151 (2003)
[20] Glover, F.: Optimization by ghost image processes in neural networks. Computers and Operations Research 21(8), 801–822 (1994); Heuristic, Genetic and Tabu Search
[21] Gonçalves, G., Hsu, T., Dupas, R., Housroum, H.: Une plate-forme de simulation pour la gestion dynamique de tournées de véhicules. Journal Européen des Systèmes Automatisés 41(5), 515–539 (2007)
[22] Helsgaun, K.: An effective implementation of the lin-kernighan traveling salesman heuristic. European Journal of Operational Research 126(1), 106–130 (2000)
[23] Johnson, D., McGeoch, L.: Experimental analysis of heuristics for the stsp. In: Du, D.-Z., Pardalos, P.M., Gutin, G., Punnen, A. (eds.) The Traveling Salesman Problem and Its Variations of Combinatorial Optimization, vol. 12, pp. 369–443. Springer, US (2004)
[24] Kilby, P., Prosser, P., Shaw, P.: Dynamic vrps: a study of scenarios. Technical Report APES-06-1998, University of Strathclyde, UK (1998)
[25] Kohonen, T.: Self-organization and associative memory, 3rd edn. Springer, New York (1989)
[26] Larsen, A., Madsen, O.B.G., Solomon, M.M.: Recent developments in dynamic vehicle routing systems. In: Sharda, R., Voß, S., Golden, B., Raghavan, S., Wasil, E. (eds.) The Vehicle Routing Problem: Latest Advances and New Challenges. Operations Research/-Computer Science Interfaces Series, vol. 43, pp. 199–218. Springer, US (2008)
[27] Mester, D., Braysy, O.: Active-guided evolution strategies for large-scale capacitated vehicle routing problems. Computers and Operations Research 34(10), 2964–2975 (2007)

[28] Montemanni, R., Gambardella, L., Rizzoli, A., Donati, A.: Ant colony system for a dynamic vehicle routing problem. Journal of Combinatorial Optimization 10, 327–343 (2005)
[29] Moscato, P.: A gentle introduction to memetic algorithms. In: Handbook of Metaheuristics, pp. 105–144. Kluwer Academic Publishers (2003)
[30] Preparata, F.P., Shamos, M.I.: Computational geometry: an Introduction. Springer, New York (1985)
[31] Psaraftis, H.N.: Dynamic vehicle routing: Status and prospects. Annals of Operations Research 61, 143–164 (1995)
[32] Psaraftis, H.N.: Dynamic vehicle routing problems, pp. 223–248. Elsevier Science Ltd. (1998)
[33] Reinelt, G.: Tsplib - a traveling salesman problem library. ORSA Journal on Computing 3(4), 376–384 (1991)
[34] Toth, P., Vigo, D.: The granular tabu search and its application to the vehicle-routing problem. INFORMS Journal on Computing 15(4), 333–346 (2003)