Ant colony system for a dynamic vehicle routing problem

R. Montemanni*, L.M. Gambardella,
A.E. Rizzoli, A.V. Donati

Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)

Galleria 2, CH-6928 Manno-Lugano, Switzerland

Abstract

An aboundant literature on vehicle routing problems is available. However, most of the work deals with static problems, where all data are known in advance, i.e. before the optimization has started.

The technological advances of the last few years give rise to a new class of problems, namely the dynamic vehicle routing problems, where new orders are received as time progresses and must be dynamically incorporated into an evolving schedule.

In this paper a dynamic vehicle routing problem is examined and a solving strategy, based on the Ant Colony System paradigm, is proposed.

Some new public domain benchmark problems are defined, and the algorithm we propose is tested on them.

Finally, the method we present is applied to a realistic case study, set up in the city of Lugano (Switzerland).

Keywords: Dynamic vehicle routing, ant colony optimization.

*Corresponding author. Tel: +41 91 610 8671. Fax: +41 91 610 8661. Email: roberto@idsia.ch.

1 Introduction

In the *Vehicle Routing Problem* (*VRP*) a fleet of vehicles with limited capacity has to be routed in order to visit a set of customers at a minimum cost (generally the total travel time). In the static *VRP* all the orders are known *a priori*.

Dynamic Vehicle Routing Problems (*DVRP*), sometimes referred to as *Online Vehicle Routing Problems*, have recently arisen thanks to the advances in communication and information technologies that allow information to be obtained and processed in real time. In this case, some of the orders are known in advance before the start of the working day, but as the day progresses, new orders arrive and the system has to incorporate them into an evolving schedule.

The existence of a communication system between the dispatcher (where the tours are calculated, e.g. the headquarter of the company) and the drivers is assumed. The dispatcher can periodically communicate to the drivers the new visits assigned to them. In this way, during the day, each driver always has a knowledge about the next customers assigned to her/him.

In this paper we consider the case where vehicles do not have to go back to the depot in order to treat new orders assigned to them when they have already left the headquarter. In our model we also take into account capacity constraints for the vehicles. Mainly three families of real problems presenting these characteristics can be identified:

- **feeder systems**. Local dial-a-ride systems typically aiming at feeding another, wider area, transportation system at a particular transfer location (see, for example, Gendreau and Potvin [12]). Vehicles are empty when they leave the depot.

- **courier service problems** (e.g. Federal Express). Parcels are collected at customer locations and brought back to a central depot for further processing and shipping. Vehicles are empty when they leave the depot.

- **fungible items / consumable goods distribution** (e.g. distribution of fuel for heating plants). Vehicles are filled at the depot before leaving

and unload goods at customer locations.

We also assume that it is not possible to use statistical information to forecast future orders. This assumption excludes the application of methods developed for the *stochastic vehicle routing problem* (see, for example, Gendreau et al. [10] and [11]).

It is interesting to observe that the approach we propose can easily handle dynamic travel times update, based on real-time traffic information. Travel times are however keep constant in all the experiments presented in this paper.

A problem similar to the one we consider in this paper has been studied in Gendreau et al. [9]. The main difference between our model and that used in [9] is that we take into account vehicle capacities, that are not considered in [9], although apparently it would have not been difficult to incorporate them. In Ichoua et al. [16] the approach described in [9] is integrated with a vehicle diversion mechanism. In practice, it is possible to divert a vehicle away from its current destination in response to a new customer request. We do not consider this option in our algorithm.

Hvattum et al. [15] presented an approach for problems where statistical information about orders appearance is available.

Savelsbergh and Sol [22] (see also Sol [23]) presented a planning module designed for a transportation company, which embeds a dynamic VRP module.

Kilby et al [17], which adopt the same model we use, presented a study on how the modification of some parameters, concerning problem dynamism, impacts on the performance of a simple heuristic algorithm they implemented (more details are available in Section 4.1).

Guntsch and Middendorf [14] (see also [13]) propose an Ant heuristic algorithm for a *Dynamic Traveling Salesman Problem* (DTSP).

A survey on results achieved on the different types of $DVRP$s can be found in Gendreau and Potvin [12] (see also Psaraftis [19] and [20]).

In this paper we propose a solving technique that exploits some characteristics of the *Ant Colony System* optimization paradigm to smoothly save

information about promising solutions when the optimization problem evolves because of the arrival of new orders.

In Section 2 a formal description of the problem is given. Section 3 is devoted to the description of the approach we propose. In Section 4 a set of benchmarks is described and computational results are presented. A study on a real-world *DVRP* problem, set up on the road network of the city of Lugano, with customers data provided by a local fuel distribution company, is proposed in Section 5. Conclusions are given in Section 6.

2 Problem description

The *static vehicle routing problem* can be described as follows: n customers must be served from a (unique) depot. Each customer i asks for a quantity q_i of goods. A fleet of v vehicles, each vehicle a with a capacity[1] Q_a, is available to deliver goods. A service time s_i is associated with each customer. It represents the time required to service him/her. Therefore, a *VRP* solution is a collection of tours.

The *VRP* can be modelled in mathematical terms through a complete weighted digraph $G = (V, A)$, where $V = \{0, 1, \ldots, n\}$ is a set of nodes representing the depot (0) and the customers $(1, \ldots, n)$, and $A = \{(i, j)|i, j \in V\}$ is a set of arcs, each one with associated a minimum travel time tt_{ij}. The quantity of goods q_i requested by each customer i $(i > 0)$ is associated with the corresponding vertex. Labels $Q_1, \ldots, Q_v$, corresponding to vehicles capacities, are finally associated with the starting locations of the vehicles, i.e. vertex 0 (the depot).

The goal is to find a feasible set of tours with the minimum total travel time. A set of tours is feasible if each node is visited exactly once (i.e. it is included into exactly one tour), each tour a starts and ends at the depot (vertex 0), and the sum of the quantities associated with the vertices contained in it, never exceeds the corresponding vehicle capacity Q_a.

[1] In the classic *VRP* the capacity would be the same for all the vehicles.

The *dynamic vehicle routing problem* is strongly related to the static VRP. The main difference is that new orders arrive when the working day has already started, dynamically changing the optimization problem. The $DVRP$ can be consequently modelled as a sequence of static VRP-like instances (see Section 3). In particular each static VRP will contain all the customers known at that time, but not yet served.

3 The *ACS-DVRP* algorithm

The algorithm we propose for the $DVRP$ has been developed to run in a centralized fashion by the people in charge of orders dispatching. We refer to the phase where an order is communicated to a driver as *commitment phase*. In our strategy, the commitment cannot be retracted, i.e. once an order is committed to a driver, this assignment cannot be changed. It is important to observe that, on the other hand, our approach constantly provides a solution covering all the known orders. Among these orders, the assignment of those not yet committed, can be retracted.

Our approach is based on the idea of dividing the working day into n_{ts} time slices with equal length $\frac{T}{n_{ts}}$ - where T is the lenght of the working day - and to postpone the processing of each new order arrived during a time slice to the end of it. The idea has been proposed in Kilby et al. [17]. During each time slice, a problem very similar to a static VRP, but with vehicles with heterogeneous capacities and starting locations, is created, and optimization is carried out. In each of these problems, the aim is to minimize the total travel time while serving all the known orders.

The concept of time slice has been introduced to bound the time dedicated to each static problem. A different strategy may be to stop and restart the optimizer each time a new event occurs (i.e. a new order arrives or a decision has to be committed to a vehicle, see Gendreau et al. [9]). The drawback of such an approach is that the time dedicated to each static problem would not be known in advance, and consequently optimization may be interrupted before a

good local minimum is reached, producing unsatisfactory results. On the other hand, a strategy like the one adopted in [9] is more suitable for problems where urgent orders are likely to exist. This is not the case of the problems we treat, where time windows on orders are not handled.

The concept of *cut-off time* is also considered in our approach. Orders received after time T_{co}, which is a parameter defined by the user, are postponed to the following working day. This practice is very common in real companies. It is also important to observe that, in the model we present, all the received orders are accepted by the company. A mechanism to filter orders (i.e. to reject some of them) could be inserted in our architecture, but it does not exist in the current implementation.

An *advanced commitment time* T_{ac} is also consider in our system. In practice, an order has to be committed to a driver at least T_{ac} seconds prior to the planned time of departure from the last location visited before that of the order itself. This advanced commitment time has been considered to give the drivers an appropriate reaction time after having been committed new orders.

There are three main elements in our architecture. They are the following ones:

- **events manager**. It collects new orders and keeps trace of already served orders and of the position of the vehicles. The event manager uses these information to construct a sequence of static VRP-like instances. It is also in charge of the commitment of orders to the drivers.

- ***ACS*** (Ant Colony System) **algorithm**. It is used to heuristically solve the static VRP-like instances generated by the events manager.

- **pheromone conservation procedure**. It is a crucial element of the architecture we propose. It is used to efficiently pass on information about properties of good solutions from a time slice to the following one.

Figure 1 depicts the architecture we propose. The three main elements have a bold border, and their interactions with the other components are drawn.

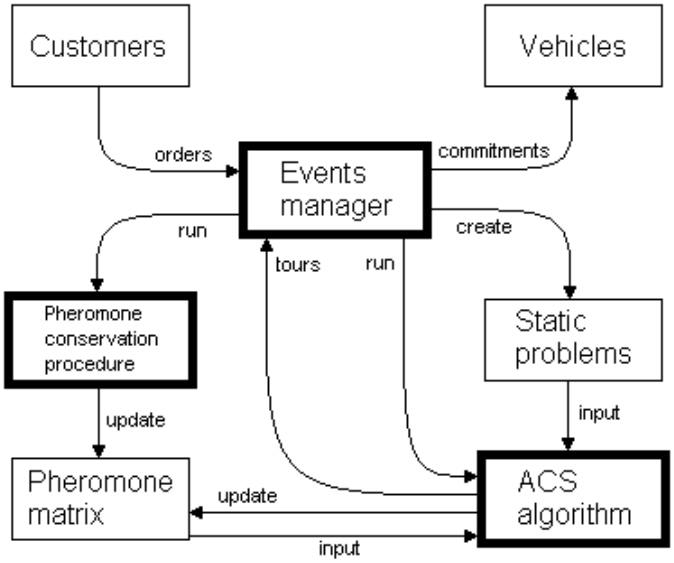

Figure 1: Architecture of the *ACS-DVRP* algorithm.

The following subsections are devoted to the detailed description of the three elements listed above.

3.1 Events manager

The events manager is the interface between the architecture and the external world. New orders from customers are handled by this module, and commitments of orders to drivers are managed by it.

Based on the division of the working day into time slices, and based on parameters T_{co} and T_{ac}, the events manager creates static problems and runs in sequence the pheromone conservation procedure and the ACS algorithm on these static problems. Based on the solutions provided by the ACS algorithm, the events manager finally decides about commitments.

The static problem considered during the first time slice (i.e. at the beginning of the working day) only deals with orders known from the previous working day. The next static problems will consider all the orders received by the system at the beginning of the time slices, and which have not been committed to drivers yet (committed orders are part of the past, but will be stored in the final solution). In these problems, each vehicle starts from the location of the last customer committed to it, with a starting time corresponding to the end of the servicing time for this customer, and with a capacity corresponding to the residual capacity of the vehicle, after it has served all the customers previously committed to it.

At the end of each time slice, the best solution found for the corresponding static problem is examined and orders with a processing time starting within the next $\frac{T}{n_{ts}} + T_{ac}$ seconds are committed to the respective drivers. Note that the processing time of an order starts when the vehicle to which it is assigned, has to leave from its previous customer. Note that no order can be planned to start in the first T_{ac} seconds of each time slot, since this would violate the advanced commitment time constraint.

An exception to the commitment strategy described above is represented

by return journeys to the depot. A return journey is committed to a vehicle only in two circumstances: the current time is greater than or equal to T_{co} and all the customers have been served, or the vehicle has used all its capacity. In practice, a vehicle will wait at its last committed customer in case its actual next destination is the depot and none of the two conditions described above is satisfied. This is done because tours may be replanned (due to new orders) and new customers might be committed to vehicles.

A pseudo-code for the event manager module is presented in Figure 2. The set *PendOrds* initially contains the orders known from the previous day. The variable *Time* is initialized to 0, while the location of all the vehicles is initially set at the depot. An iterative statement is then entered. A static problem (*StaticPro*), containing orders in *PendOrds* and covering the time window $[Time + T_{ac}; Time + T_{ac} + \frac{T}{n_{ts}}]$ ($[Time; Time + T_{ac} + \frac{T}{n_{ts}}]$ in case *Time*$= 0$), is created and solved (with the procedure that will be described in Section 3.2), and some commitments (*CommOrds*) are done accordingly to the solution of *StaticPro*. *PendOrds* is updated together with starting positions, capacities and travel times of the vehicles. The pheromone matrix is finally updated, according to the *Pheromone conservation* strategy that will be described in Section 3.3. These operations are repeated until *PendOrds* $= \emptyset$ and *Time* $> T_{co}$. After the completion of the iterative statement, the depot is committed as last destination to all the vehicles.

3.2 An *ACS* algorithm for VRP-like problems

The *Ant Colony System* (*ACS*) algorithm is an element of the *Ant Colony Optimization* (*ACO*) family of algorithms (Dorigo et al. [4], Bonabeau et al. [1]). The first *ACO* algorithm, *Ant System* (*AS*), has been proposed by Colorni, Dorigo and Maniezzo (see [3] and [5]) and is based on a computational paradigm inspired by the way real ant colonies function. The main underlying idea was to parallelize search over several constructive computational threads. A dynamic memory structure, which incorporates information on the effectiveness of pre-

Procedure EventsManager
$Time := 0$;
The starting position of each vehicle is set at the *depot*;
$PendOrds :=$ orders known from the day before;
While ($PendOrds \neq \emptyset$ or $Time < T_{co}$)
  If ($Time > 0$)
    $StaticProb :=$ problem with orders in $PendOrds$ and starting time $Time+T_{ac}$;
  Else
    $StaticProb :=$ problem with orders in $PendOrds$ and starting time $Time$;
  EndIf
  Run *ACS* on $StaticProb$;
  $CommOrds :=$ orders with processing time $\geq Time+\frac{T}{n_{ts}}+T_{ac}$;
  Commit orders in $CommOrd$;
  $PendOrds := PendOrds \setminus CommOrds$;
  $PendOrds := PendOrds \cup \left\{\text{orders appeared in the last } \frac{T}{n_{ts}} \text{ seconds}\right\}$;
  $Time := Time + T_{ts}$;
  Update starting positions, capacities of vehicles and travel times;
  Update pheromone matrix;
EndWhile
Commit the *depot* to all the vehicles;

Figure 2: Pseudo-code of the *Events manager* procedure.

viously obtained results, guides the construction process of each thread. The behavior of each single agent is inspired by the behavior of real ants.

The *ACS* algorithm has been originally proposed by Gambardella and Dorigo in [7]. We apply this paradigm to the static vehicle routing problems faced within a *DVRP*. The method is similar to the *MACS-VRPTW* algorithm described in Gambardella et al. [8], which is one of the state-of-the-art algorithm for the *vehicle routing problem with time windows* ($VRPTW$) and was able to provide the best known solutions for many benchmarks[2].

In order to simplify the description of the algorithm, we will consider v dummy depots (one for each vehicle of the fleet) and we will refer to them as $d_1, \ldots, d_v$. Solutions retrieved by ants will be represented as long, single tours. In this context, nodes contained within two consecutive dummy depots d_a and d_b (with $a, b \in \{1, \ldots, v\}$) form the (partial) tour associated with vehicle a. The

[2]Up to date results are available at:

http://www.sintef.no/static/am/opti/projects/top/vrp/benchmarks.html.

partial tour associated with vehicle b will start from the dummy depot d_b, which corresponds to the location of the last customer committed to vehicle b. The starting time from d_b corresponds to the end of the serving time for the last customer committed to vehicle b, while the capacity of b will be equal to the residual capacity of b, i.e. Q_b minus the quantity ordered by customers already committed to vehicle b.

The main element of the algorithm are *ants*, simple computational agents that individually and iteratively construct solutions for the problem. At each step, every ant k computes a set of feasible expansions to its current partial solution and selects one of these probabilistically, according to a probability distribution specified as follows. For ant k the probability p_{ij}^k of visiting customer j after customer i (i.e. the last visited customer) depends on the combination of two values:

- the attractiveness η_{ij} of arc (i, j), as computed by some heuristic indicating the *a priori* desirability of that move. In our case $\eta_{ij} = \frac{1}{tt_{ij}}$, i.e. it depends on the travel time between customer i and customer j;

- the pheromone level τ_{ij} of arc (i, j), indicating how proficient it has been in the past to visit j after i is a solution; it represents therefore an *a posteriori* indication of the desirability of that move.

Pheromone trails are updated at each iteration. The level of those associated with arcs contained in "good" solutions are increased. The specific formula for defining the probability distribution makes use of a set $\mathcal{F}_i^k$ which contains feasible customers to extend the current partial solution of ant k.

The probability for ant k to append arc (i, j) to its partial solution is then given by:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}(\eta_{ij})^\beta}{\sum_{l \in \mathcal{F}_i^k}(\tau_{il}(\eta_{il})^\beta)} & \text{if } j \in \mathcal{F}_i^k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where the sum is over all the feasible moves, β is a parameter controlling the relative importance of the trail τ_{ij} of arc (i, j) versus the actual attractiveness

η_{ij} of the same arc. In this manner p_{ij}^k is a trade-off between the apparent desirability and information from the past. Ant k will select customer $j := \text{argmax}_{l \in \mathcal{F}_i^k}\{p_{il}^k\}$ (*exploitation*) with probability q, while with probability $(1 - q)$ each move (i, j) is selected with a probability given by (1) (*exploration*). Parameter q $(0 \leq q \leq 1)$ determines the relative importance of exploitation versus exploration.

When ant k moves from i to j, a local updating is performed on the pheromone matrix, according to the following rule:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\tau_0 \tag{2}$$

where τ_0 is the initial value of trails (defined by the user) used for the first static *VRP* and for entries of the pheromone matrix involving new customers in the following problems (see Section 3.3), and ρ $(0 \leq \rho \leq 1)$ is a parameter regulating pheromone evaporation. It mimics what happens in the case of real ants.

An interesting aspect of the local updating is that while edges are visited by ants, equation (2) makes the trail intensity diminish, making them less and less attractive, and favoring therefore the exploration of not yet visited edges and diversity in solution generation.

Once a complete solution is available, it is tentatively improved using a local search procedure. We used a very simple greedy algorithm, which iteratively selects a customer and tries to move it into another position within its tour or within another tour. A parameter regulating the maximum computation time for this local search, t_{ls}, has be specified by the user.

Once the m ants of the colony have completed their computation, the best known solution is used to globally modify the pheromone trail. In this way a "preferred route" is memorized in the pheromone trail matrix and future ants will use this information to generate new solutions in a neighborhood of this preferred route. The pheromone matrix is updated as follows:

```
Procedure ACS
BestCost := ∞;
For each arc (i, j)
   τ_ij := τ_0;
EndFor
While (computation time < T/n_ts)
   For k := 1 to m
      While (Ant k has not completed its solution)
         Select the next customer j;
         Update the trail level τ_ij (Equation (2));
      EndWhile
      Run a local search (maximum computation time = t_ls);
      Cost := Cost of the current solution;
      If (Cost < CostBest)
         CostBest := Cost;
         BestSol := current solution;
      EndIf
   EndFor
   For each move (i, j) in solution BestSol
      Update the trail level τ_ij (Equation 3);
   EndFor
EndWhile
```

Figure 3: Pseudo-code of the *ACS* procedure.

$$\tau_{ij} = (1-\rho)\tau_{ij} + \frac{\rho}{CostBest} \quad \forall (i,j) \in BestSol \tag{3}$$

where *CostBest* is the total travel time of solution *BestSol*, the best tour generated by the algorithm since the beginning of the computation.

The process is iterated by starting again m ants until a termination condition is met. In our simulations the natural termination criterion is to set a maximum computation time of $\frac{T}{n_{ts}}$ seconds, which is defined as the length of each time slice in our application, i.e. we use all the available time.

Pseudo-code of the *ACS* procedure for the static problems faced in a *DVRP*, is presented in Figure 3.

3.3 Pheromone conservation procedure

The use of the *ACS* paradigm to solve the static problems produces a very important side-effect, which is a key-element of our algorithm for the *DVRP*.

Once a time slice is finished and the respective static problem has been solved by the *ACS* algorithm, the pheromone matrix contains encrypted information about characteristics of good solutions for this problems. In particular, pairs of customers which are visited in sequence in good solutions, will have high values in the corresponding entries of the pheromone matrix.

This information can be passed on to the static problem corresponding to the next time slice, since the two problems are potentially very similar. This operation prevents optimization to restart each time from scratch and heavily contributes to the good performance guaranteed by the approach we propose (see Section 4.2).

The strategy we follow to transfer information is inspired by Guntsch and Middendorf [13] and [14]. A new parameter γ_r is introduced to regulate pheromone conservation. For each pair of customers which appear both in the old and in the new static problem, the corresponding pheromone matrix entry is initialized to the following value:

$$\tau_{ij} = (1 - \gamma_r)\tau_{ij}^{\text{old}} + \gamma_r \tau_0 \tag{4}$$

where τ_{ij}^{old} is the value of τ_{ij} in the old static problem. In fact, pheromone values are not completely reinitialized, but a trace of old values remains.

Entries of the new pheromone matrix corresponding to pairs of customers involving new customers are initialized to τ_0.

4 Computational results

This section is devoted the the experimental evaluation of the *ACS-DVRP* algorithm on some simulated scenarios.

The benchmarks adopted will be described in detail in Section 4.1; Section 4.2 will document the parameter tuning phase for *ACS-DVRP*, while in Section 4.3 the results achieved by the algorithm will be presented together with those of a basic algorithm based on the *GRASP* paradigm.

The algorithms have been coded in ANSI C, and all the tests have been carried out on a 1.5GHz/256MB Intel Pentium 4 machine.

4.1 Benchmarks description

The dynamic problems adopted in this paper have been originally proposed in Kilby et al. [17][3]. They are derived from some very popular static *VRP* benchmark datasets, namely 12 problems are taken from Taillard [24], 7 problems are from Christofides and Beasley [2] and 2 problems are from Fisher et al. [6]. These problems range from 50 to 199 customers. The number of customers can be inferred from the name of each instance. In order to obtain dynamic problems, Kilby et al. added to these problem the following features:

- **length of the working day**. We will refer to this parameter as T, like anticipated in Section 3.
- **appearance time of each order**. It contains, for each order, the moment of the working day, when the order becomes known to the dispatcher.
- **duration of each order**. It represent, for each order, the time required to serve the corresponding customer.
- **number of vehicles**. It contains the dimension, in number of trucks, of the fleet available for serving the customers. The number of vehicles is set to 50 for each problem. This setting guarantees that it is possible to serve all the orders for the problems considered.

More details about the dynamic problems can be found in Kilby et al [17].

Kilby et al. [17] presented a study on how changes in parameters T_{co} (i.e. the cut-off time, after which orders are postponed to the following day) and T_{ac} (i.e. the advanced commitment time, modelling how much in advance orders have to be committed to drivers) affect the performance of dynamic algorithms in general. Since our target is to univocally define a set of benchmarks on which

[3]The problems are available at http://www.dcs.st-and.ac.uk/~apes/apedata.html.

dynamic algorithms can be compared (such a set of problems is not available in the literature), we fixed the value of these two parameters. We adopted the following values, which have been chosen according to the suggestions provided in Kilby et al. [17]: $T_{co} = 0.5 \cdot T$ and $T_{ac} = 0$.

In the application of the methodology we propose to a real problem, *ACS-DVRP* is suppose to run for the whole working day. Since in this section our aim is to run simulations, and it would be very time consuming to carry out only a single run of the algorithm per day, we chose to map each working day into 1500 seconds of CPU time (i.e. $T = 1500$ seconds and customer appearance and serving times are changed proportionally).

4.2 Parameter setting for the *ACS-DVRP* algorithm

Some parameters have to be tuned in the *ACS-DVRP* framework. Most of them are those of the *ACS* algorithm used to tackle the static *VRP*-like problems generated by the events manager.

From a previous study presented in Gambardella et al. [8] for the *MACS-VRPTW* algorithm, it is known that a good parameter setting for *ACS* algorithms applied to classic vehicle routing problems is the following one: $q_o = 0.9$, $\beta = 1$, $\rho = 0.1$, a small value for m (number of ants), e.g. $m = 3$, and $\tau_0 = \frac{1}{n \times Cost(PI)}$, where *Cost(PI)* is the cost of a solution retrieved by a greedy heuristic algorithm. Since each one of our static problems is (almost) a classic *VRP*, we use these settings and in particular we initially solve each static problem with a greedy post-insertion algorithm (like the one described in [8]) in order to obtain *Cost(PI)*, and consequently τ_0.

Parameters t_{ls}, which model the time dedicated to local search during each iteration of the *ACS* algorithm, has also to be set. We decided to set $t_{ls} = \frac{T}{6 \cdot n_{ts}}$, and to consequently tune parameter n_{ts}. This last parameter is crucial for algorithm *ACS-DVRP*, since too large values would imply that the optimization is restarted too often, without local minima can be reached. On the other hand, too small values would force the method to carry out long optimizations on

problems which are not up to date, because the most recent information would be ignored. In this case very good local minima might be reached for the problems investigated, but these problems do not contain updated information, and consequently the optimization effort is somehow vanished. For this reason, a careful tuning for parameter n_{ts} is required. We carried out some tests, that are summarized in Table 1. For these experiments we set $\gamma_r = 0.3$ (this value was suggested by some preliminary tests).

Three values for n_{ts} (i.e. 10, 25 and 50) and three problems are considered in Table 1. For each combination (*problem, value of* n_{ts}) five runs of algorithm *ACS-DVRP* have been carried out and three values are reported in the corresponding entry of the table: *Min*, *Max* and *Avg*, that respectively represent the best, the worst and the average total travel times found over the five runs.

Table 1: Calibration of parameter n_{ts} (number of time slices).

n_{ts}	$\frac{T}{n_{ts}}$	t_{ls}		c100	f71	tai75a
10	150	25	Min	1004.58	311.95	1880.11
			Max	1145.20	399.26	2105.14
			Avg	1083.64	362.93	1963.19
25	60	10	Min	973.26	311.18	1843.08
			Max	1100.61	420.14	2043.82
			Avg	1066.16	348.69	1945.20
50	30	5	Min	1131.95	333.25	1966.92
			Max	1228.97	452.73	2133.87
			Avg	1185.25	417.74	2019.82

Table 1 suggests that a good tradeoff between reactivity to dynamic events and accurate optimization of the static VRP-like problems, is reached with $n_{ts} = 25$. This setting guarantees the best performance of algorithm *ACS-DVRP* for the benchmarks considered. Consequently we will adopt this setting in the remainder of this section.

The last parameter which requires to be tuned is γ_r, the parameter used by the pheromone conservation procedure (see Section 3.3). Some tests have been carried out for experimentally finding good values for the parameter. They are summarized in Table 2, where the results obtained on three benchmarks with

four different values of γ_r are presented. In particular the values 0.1, 0.3, 0.5 and 1.0 are considered. For each of these values, five runs have been carried out on each problem and for each pair (*problem, value of* γ_r) three quantities are reported: *Min*, *Max* and *Avg*. They are respectively the best, the worst and the average total travel times found over the five runs.

Table 2: Tuning of γ_r (parameter for pheromone conservation).

γ_r		c100	f71	tai75a
0.1	Min	1072.44	352.77	1928.18
	Max	1157.43	419.21	2220.75
	Avg	1116.13	383.27	2016.78
0.3	Min	973.26	311.18	1843.08
	Max	1100.61	420.14	2043.82
	Avg	1066.16	348.69	1945.20
0.5	Min	1039.36	360.20	1847.41
	Max	1135.26	443.16	2054.91
	Avg	1087.90	389.00	1962.66
1.0	Min	1079.12	367.32	1873.69
	Max	1133.15	431.53	2102.58
	Avg	1098.99	399.18	1971.91

From Table 2 the setting $\gamma_r = 0.3$ clearly appears to be the most promising. In the remaining tests γ_r will consequently be set to 0.3.

It is also interesting to compare the results obtained with $\gamma_r = 0.3$ and $\gamma_r = 1.0$ (i.e. no pheromone conservation). When $\gamma_r = 0.3$ is used, the results are always considerably better and this highlights the important role played by pheromone conservation in our algorithm.

4.3 Comparison with other algorithms

In this section we aim to evaluate the computational results of the *ACS-DVRP* algorithm. For comparison reasons we implemented a basic *GRASP* algorithm (we will refer to it as $GRASP - DVRP$). The algorithm *GRASP-DVRP* will be described in detail in Section 4.3.1, while the computational results of the two algorithms will be presented in in Section 4.3.2.

4.3.1 Algorithm *GRASP-DVRP*

The approach is based on an architecture similar to that described in Section 3.2 for the *ACS-DVRP* algorithm. The main differences are that here a GRASP (Greedy Randomized Adaptive Search Procedure, see Resende and Ribeiro [21]) module is used instead of the ACS algorithm to tackle the static VRP-like problems, and that, consequently, no pheromone conservation strategy exists. In particular, the $GRASP$ procedure works by repeatedly carrying out the following operations for the time corresponding to a time slice:

- initial tours are generated by iteratively selecting the next customers to visit at random among those that have a travel time from the last selected location in the interval $[tt_{min}, tt_{min} + \delta_G(tt_{max} - tt_{min})]$, where δ_G is a parameter and tt_{min} and tt_{max} are the minimum and maximum feasible (in terms of vehicle capacity) travel times out of the last selected location. The procedure is repeated until a complete solution is build. Notice that when $\delta_G = 0$ the construction strategy is completely greedy, while $\delta_G = 1$ means that the next customer is selected completely at random among the feasible ones. For the experiments reported in Section 4.3.2, we will set $\delta_G = 0.75$, according to some preliminary tests that indicated this setting as the most promising one.

- the same local search procedure described in Section 3.2 for the ACS algorithm is run for t_{ls} on the solution so obtained, in order to improve it.

- if the solution so obtained is the best retrieved so far, it becomes the new best solution.

It is important to observe that a better designed, and perhaps more sophisticated implementation of the $GRASP$ algorithm (see, for example, Kontoravdis and Bard [18]) for the construction phase, would probably bring to better results.

4.3.2 Results

In Table 3 the results obtained by the algorithms described in Sections 3.2 and 4.3.1 are compared. For each problem, five runs of each algorithm have been considered. In Table 3 the best (*Min*), the worst (*Max*) and the average (*Avg*) travel time retrieved over the five runs are reported.

Table 3: Computational results.

Problem	GRASP-DVRP			ACS-DVRP		
	Min	Max	Avg	Min	Max	Avg
c100	1080.33	1169.67	1119.06	973.26	1100.61	1066.16
c100b	978.39	1173.01	1022.12	944.23	1123.52	1023.60
c120	1546.50	1754.00	1643.15	1416.45	1622.12	1525.15
c150	1468.36	1541.54	1501.35	1345.73	1522.45	1455.50
c199	1774.33	1956.76	1898.20	1771.04	1998.87	1844.82
c50	696.92	757.97	719.56	631.30	756.17	681.86
c75	1066.59	1142.32	1079.16	1009.38	1086.65	1042.39
f134	15433.84	17325.73	16458.47	15135.51	17305.69	16083.56
f71	359.16	429.64	376.66	311.18	420.14	348.69
tai100a	2427.07	2583.02	2510.29	2375.92	2575.70	2428.38
tai100b	2302.95	2636.05	2512.27	2283.97	2455.55	2347.90
tai100c	1599.19	1800.85	1704.40	1562.30	1804.20	1655.91
tai100d	1973.03	2165.39	2087.55	2008.13	2141.67	2060.72
tai150a	3787.53	4165.42	3899.16	3644.78	4214.00	3840.18
tai150b	3313.03	3655.63	3485.79	3166.88	3451.69	3327.47
tai150c	3110.10	3635.17	3219.27	2811.48	3226.73	3016.14
tai150d	3159.21	3541.27	3298.76	3058.87	3382.73	3203.75
tai75a	1911.48	2116.95	2005.44	1843.08	2043.82	1945.20
tai75b	1582.24	1934.35	1758.88	1535.43	1923.64	1704.06
tai75c	1596.17	1859.71	1674.37	1574.98	1842.42	1653.58
tai75d	1545.21	1641.91	1588.73	1472.35	1647.15	1529.00
Total	52731.63	58986.36	55562.64	50855.94	57645.52	53784.02

Table 3 shows that the *ACS-DVRP* algorithm is able to provide higher quality solutions than *GRASP-DVRP*. They are, on average, 3.56% better in terms of best results, 2.27% better in terms of worst results, and 3.20% better in terms of average results.

It is also interesting to observe that, for all the problems considered, the best solution found by the *ACS-DVRP* algorithm over five runs, is always better of

the best one retrieved by the *GRASP-DVRP* algorithm.

5 A case study

In this section we summarize the results obtained by the *ACS-DVRP* algorithm on a realistic case study, set up in the city of Lugano, Canton Ticino, Switzerland. The aim of the study is twofold: first we want to show that the approach we propose is suitable to be applied to real world problems. Second we want to empirically show that parameter n_{ts} - that regulates the number of time slices - has to be tuned also in case of real world problems, in order to obtain the best possible results.

In Figure 4 the road network of Lugano is depicted. The locations of a depot (white square) and of 50 customers (black circles) have been provided by a local fuel distribution company. Travel times among them have been calculated as classic shortest paths over the road network. A working day of 8 hours (28800 seconds) is considered, while a service time of 10 minutes (600 seconds) is set up for each customer. Customers appear during the working day with orders ranging in $[1, 31]$. A fleet of 10 vehicles with capacity 160 is finally considered. The dimension of the fleet is calibrated on the orders expected by the fuel distribution company, and it results to be well dimensioned for the case study presented here. Cut-off time T_{co} has been set to 14400 seconds, while the advanced commitment time T_{ac} has been set to 288 seconds. Parameter n_{ts} will be varied, being the argument of the study we propose.

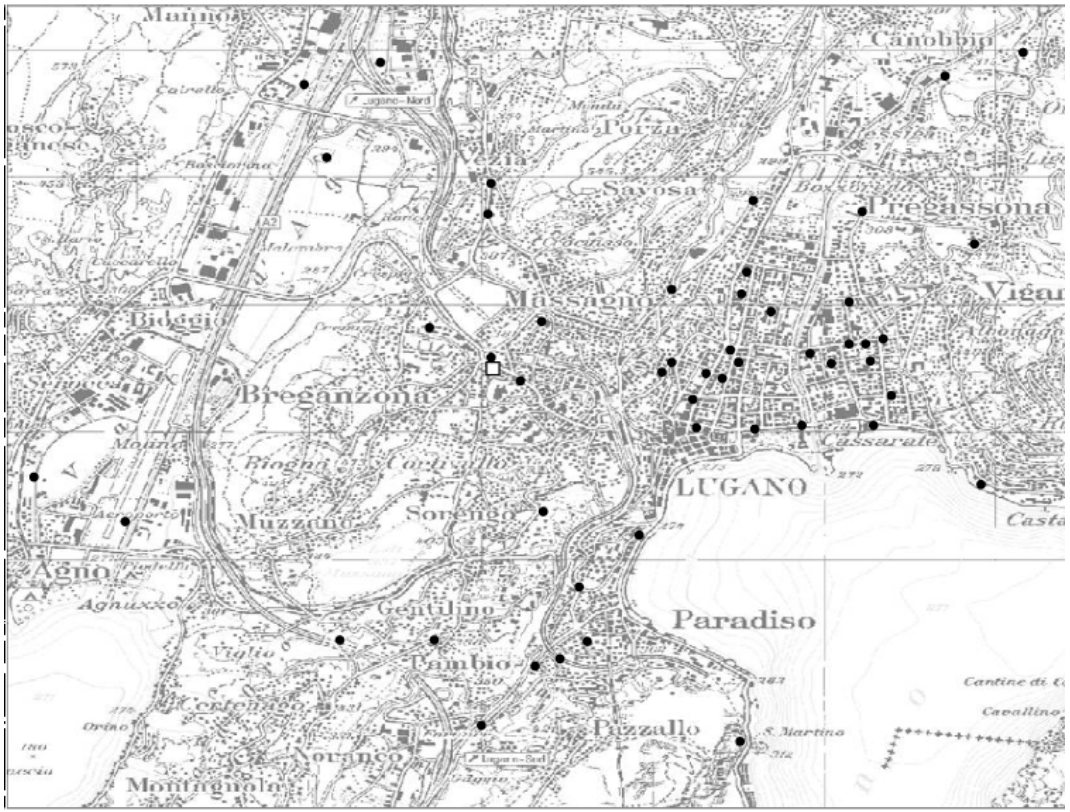

Figure 4: Case study in the city of Lugano.

Parameters q_0, β, ρ and γ_r and m are set up as described in Section 4.2. In Table 4 we presents the results obtained by the *ACS-DVRP* algorithm in different experiments, where the number of time slices (namely parameter n_{ts}) is varied. Parameter t_{ls} is adjusted according to the values of n_{ts} in such a way that $t_{ls} \approx \frac{T}{10 \cdot n_{ts}}$.

In Table 4 the first three rows define the settings of the experiments, i.e. the values of parameters n_{ts}, $\frac{T}{n_{ts}}$ and t_{ls}. The forth row shows the total travel time of the solutions found by the *ACS-DVRP* algorithm.

Table 4: Experimental results on the case study of Lugano.

n_{ts}	200	100	50	25	10	5
$\frac{T}{n_{ts}}$	144	288	576	1152	2880	5760
t_{ls}	15	30	60	120	240	480
Travel time	12702	12422	10399	9744	10733	11201

Table 4 confirms, first of all, that the approach we propose in this paper is suitable to be applied to real world problems. Table 4 also suggests that a careful tuning of parameter n_{ts} can lead to better results. In particular, it is shown that, for the case study analyzed, good values for n_{ts} are in the range $[10, 50]$. The setting $n_{ts} = 25$ seems to be the best choice. It is finally interesting to observe that the setting $n_{ts} = 25$ was the most promising also for the problems studied is Section 4. As explained in Section 4.2, too large values for n_{ts} do not lead to satisfactory results because optimization is restarted too often, without good local minima can be reached. On the other hand, when n_{ts} is too small the system is not able to take advantage of new information.

6 Conclusion

A dynamic vehicle routing problem has been studied in this paper. A solving strategy for this problem has been described. It is based on the partition of the working day into time slices. A sequence of static vehicle routing problems is then generated. An Ant Colony System algorithm has been used to solve

these problems. The properties of ACS have been also exploited to transfer information about good solutions from a time slice to the following one.

A computational study on a newly defined set of benchmarks, finally shows that the method we propose is able to achieve good results both on artificial and real problems.

Acknowledgements

Data on the road network of Lugano have been provided by the Commissione Regionale dei Trasporti del Luganese (CRTL). Information about customers for the case study have been provided by Pina Energia SA.

This work was co-funded by the European Commission IST project "MOSCA: Decision Support System For Integrated Door-To-Door Delivery: Planning and Control in Logistic Chain", grant IST-2000-29557. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from social insect behaviour. *Nature*, 406:39–42, 2000.

[2] N. Christofides and J. Beasley. The period routing problem. *Networks*, 14:237–256, 1984.

[3] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In Elsevier Publishing, editor, *European Conference on Artificial Life 1991 (ECAL91)*, pages 134–142, 1991.

[4] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.

[5] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 26(1):29–41, 1996.

[6] M. Fisher, R. Jakumar, and L. van Wassenhove. A generalized assignment heuristic for vehicle routing. *Networks*, 11:109–124, 1981.

[7] L.M. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *IEEE Conference on Evolutionary Computation (ICEC96)*, pages 622–627, 1996.

[8] L.M. Gambardella, É. Taillard, and G. Agazzi. MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. in *New ideas in optimization*. D. Corne et al. editors. Pages 63-76, 1999.

[9] M. Gendreau, F. Guertin, J.-Y. Potvin, and É. Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33(4):381–390, 1999.

[10] M. Gendreau, G. Laporte, and R. Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88:3–12, 1996.

[11] M. Gendreau, G. Laporte, and R. Séguin. A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44:469–477, 1996.

[12] M. Gendreau and J.-Y. Potvin. Dynamic vehicle routing and dispatching. in *Fleet management and logistic*. T.G. Crainic and G. Laporte editors. Pages 115-226, 1998.

[13] M. Guntsch and M. Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In E.J.W. Boers et al., editor, *Application of evolutionary computing: Proceedings of EcoWorkshops 2001*, volume Lecture Notes in Computer Science 2037, pages 213–222, 2001.

[14] M. Guntsch and M. Middendorf. Applying population based ACO to dynamic optimization problems. In M. Dorigo et al., editor, *ANTS 2002*, volume Lecture Notes in Computer Science 2463, pages 111–122, 2002.

[15] L.M. Hvattum, A Lokketangen, and G. Laporte. A heuristic solution method to a stochastic vehicle routing problem. In *Proceedings of TRISTAN V - The Fifth Triennial Symposium on Transportation Analysis*, 2004.

[16] S. Ichoua, M. Gendreau, and J.-Y. Potvin. Diversion issues in real-time vehicle dispatching. *Transportation Science*, 34(4):426–438, November 2000.

[17] P. Kilby, P. Prosser, and P. Shaw. Dynamic VRPs: a study of scenarios. Technical Report APES-06-1998, University of Strathclyde, U.K., 1998.

[18] G. Kontoravdis and J.F. Brand. A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7(1):10–23, 1995.

[19] H. Psaraftis. Dynamic vehicle routing problems. in *Vehicle Routing: methods and Studies*. B.L. Golden and A.A. Assad editors. Pages 223-248, 1988.

[20] H. Psaraftis. Dynamic vehicle routing: status and prospects. *Annals of Operations Research*, 61:143–164, 1995.

[21] M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures. in *Handbook of Metaheuristics*. F. Glover and G. Kochenberger editors. Pages 219-249, 2003.

[22] M.W.P. Savelsbergh and M. Sol. DRIVE: Dynamic Routing of Independent VEhicles. *Operations Research*, 46:474–490, 1998.

[23] M. Sol. *Column generation techniques for pickup and delivery problems*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 1994.

[24] É. Taillard. Parallel iterative search methods for vehicle-routing problems. *Networks*, 23(8):661–673, 1994.