A new algorithm for a Dynamic Vehicle Routing Problem based on Ant Colony System

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, Switzerland

Abstract

An aboundant literature on vehicle routing problems is available. However, almost all 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.

The method has been tested on a set of benchmarks we have defined starting from a set of widely available problems. Computational results confirm the effectiveness and the efficiency of the strategy we propose.

Finally, an application to a case study, set up in the city of Lugano (Switzerland), is presented.

Keywords: Dynamic vehicle routing, ant colony optimization.

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*.

*Corresponding author. Tel: +41 91 610 8568. Fax: +41 91 610 8661. Email: roberto@idsia.ch.

Dynamic Vehicle Routing Problems (*DVRP*) are a new class of problems, which have arisen thanks to recent advances in communication and information technologies that allow information to be obtained and processed in real time. In these problems new orders arrive when a schedule has already been defined, so it must be updated during run time.

In the problem we consider in this paper, some of the orders are known in advance (i.e. before the trucks leave the depot), and an initial schedule is created covering these visits. As the day progresses, new orders enter the system and must be incorporated into an evolving schedule.

Let's assume that a communication system between the dispatcher (where the tours are calculated) and the drivers exists. The dispatcher can periodically inform the drivers about the next visit(s) assigned to them (*commitment*). According to this model of information transfer, every driver has, at each time step, a partial knowledge about the remainder of his/her tour.

In the problem treated in this paper, vehicles which are assigned new orders when they are already travelling, do not need to go back to the depot in order to process them. This formulation covers three families of dynamic vehicle routing problems. The first family is that of the so-called *feeder systems*, which typically are local dial-a-ride systems aimed at feeding another, wider area, transportation system at a particular transfer location (Gendreau and Potvin [14]).

The second application is to courier service problems (e.g. Federal Express), where parcels are collected at customer locations and brought back to a central depot for further processing and shipping. In these two applications the vehicles are empty when they leave the depot, and collect goods at customer locations.

In a third possible application, vehicles are filled at the depot and unload goods at customer locations. In this case goods must be fungible items or consumable goods (e.g. fuel).

As far as we are aware, problems similar to the one considered in this paper, have been treated only in Kilby et al. [19], Gendreau et al. [13] and Ichoua et al. [18].

The work presented in [19] is mainly a study on how the modification of some parameters of the problem (which in fact determine the dynamicity level) impacts on the performance of the simple heuristic algorithm they propose.

The main difference between the problem we study and the one formulated in [13], is that in [13] vehicle

capacities are not treated, while we consider them. The solution approach they propose differs from ours because we introduce the concept of *time slot* (see Section 3), which is not used in [13]. The method is based on a tabu search algorithm.

In [18] the algorithm described in [13] 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.

Most of the recent research on *DVRP*s is about real applications, and consequently very specific-purpose problems (different from the one we treat) are formulated. Caramia et al. [2] analyze a multi-cab metropolitan transportation system, and propose a heuristic technique based on dynamic programming. Coslovich et al. [5] present a two-phases insertion algorithm for an urban dial-a-ride problem. Krumke et al. [20] formalize a real-time dispatching of automobile service units and propose a solving technique based on column generation and set partitioning. Facchetti and Salani [8] describe a system for a dynamic dial-a-ride problem. Savelsbergh and Sol [24] (see also Sol [25]) present a planning module designed for a transportation company.

Guntsch and Middendorf [17] (see also [16]) propose an Ant heuristic algorithm for a *Dynamic Traveling Salesman Problem* (DTSP).

In Gendreau et al. [12] some neighborhood search heuristics for dial-a-ride problems are finally presented.

A survey on results achieved on the different types of *DVRP*s can be found in Gendreau and Potvin [14] (see also Psaraftis [22] and [23]).

We propose a solving technique based on the *Ant Colony System* paradigm (see Section 3.2). A sequence of static *VRP*s is solved, and a mechanism to transfer information about good solutions from a static *VRP* to the following one, is implemented.

In Section 2 a formal description of the problem treated in this paper is given. Section 3 is devoted to the description of the approach we propose. In Section 4 a set of benchmarks is defined and computational results are presented. A realistic *DVRP* problem, set up in the city of Lugano, is studied in Section 5. Conclusions are given in Section 6.

2 Problem description

In this section the (classic) *VRP* is described. This problem definition is then used to introduce the dynamic version of the *VRP* we will study.

2.1 The static vehicle routing problem

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 a minimum travel time tt_{ij} associated. The quantity of goods q_i requested by each customer i $(i > 0)$ is associated with the corresponding vertex with a label. Labels $Q_1, \ldots, Q_v$, corresponding to vehicles capacities, are finally associated with 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.

2.2 The dynamic vehicle routing problem

In the dynamic vehicle routing problems, new orders arrive when the working day has already started, i.e. vehicles may have already left the depot.

In Section 3, it will be described how a *DVPR* can be seen as a sequence of static *VRP*-like instances. It can consequently be formalized in mathematical terms as a sequence of digraphs like the one described in Section 2.1. Each digraph contains all the already known customers which have to be served within the day. Associated with each one of these graphs there is also a set of (eventually empty) partial tours already

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

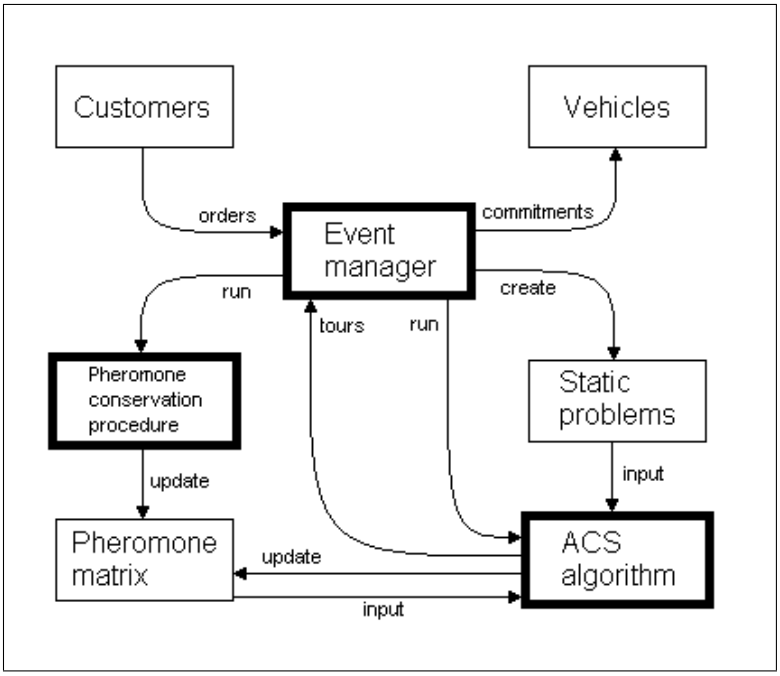

Figure 1: Architecture of the *ACS-DVRP* algorithm.

committed to the vehicles.

3 The *ACS-DVRP* algorithm

The algorithm we propose for the *DVRP* is based on three main elements.

First, there is an **event manager**, which collects new orders and keeps trace of the already served orders and of the current position of each vehicle. The event manager uses these information to construct a sequence of static *VRP*-like instances, which are solved heuristically by an ***ACS*** (Ant Colony System) **algorithm**, the second element of our architecture. The third element, the **pheromone conservation procedure**, is strictly connected with the *ACS* algorithm. It is used to pass information about characteristics of good solutions from a static *VRP* 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 represented.

The three main elements and the components are described in details in the next subsections. The complete algorithm is summarized by a pseudo-code, which is presented in Section 3.4. A final remark on the approach

we propose is given in Section 3.5.

3.1 Event manager

The approach we adopt is based on the solution of a sequence of static *VRP*-like problems, which are optimized in order to minimize the travel time required in each of them.

All new orders received before a given *time of cutoff*, T_{co}, are accepted and processed during the day[2]. Orders received after time T_{co} are postponed to the following day.

The first static problem considers only unserved customers from the day before, and optimization is carried out in order to minimize the total travel time to satisfy the requests of these customers.

The working day is then divided into n_{ts} time slots, each one long $T_{ts} = \frac{T}{n_{ts}}$ seconds, where T is the total length of the working day in seconds. The idea of dividing the working day into time slots has been already adopted in Kilby et al. [19].

New incoming orders received during each time slot are taken into account only at the end of that time slot.

The concept of time slot has been introduced in order to limit 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. [13]). The drawback of such an approach is that the time dedicated to each static problem would not be known in advance, and then optimization may be sometimes interrupted before a good local minimum is reached, producing unsatisfactory results.

An *advanced commitment time*, T_{ac}, is also considered by our strategy. At the end of each time slot the best solution found for the corresponding static problem is examined, and orders with a processing time starting within the next $T_{ts} + T_{ac}$ seconds are committed to the respective drivers. Note that the processing time of an order o starts when the vehicle to which it is assigned leaves the previous customer in its tour.

An exception to the commitment strategy described above is represented by return travels to the depot. A return travel is committed to a vehicle only in two circumstances: all the customers are served or the vehicle

[2]The number of available vehicles is overestimated in the benchmarks we consider (see Section 4.1). For this reason all the orders can be accepted.

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 new static problem, where each vehicle starts from its last committed customer at the time when the respective service terminates, and with a capacity equal to the residual capacity, is then considered. All new orders received during the last time slot are also inserted into this new problem.

The *ACS* algorithm described in the next section is used to solve the static *VRP*s.

3.2 An *ACS* algorithm algorithm for the static problems

The *Ant Colony System* (*ACS*) algorithm is an element of the *Ant Colony Optimization* (*ACO*) family of algorithms (Dorigo et al. [6], Bonabeau et al. [1]). The first *ACO* algorithm, *Ant System* (*AS*), has been initially proposed by Colorni, Dorigo and Maniezzo (see [4] and [7]) 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 previously obtained results, guides the construction process of each thread. The behavior of each single agent is inspired by the behavior of real ants.

3.2.1 Inspiration from Nature

The paradigm is based on the observation, made by ethologists, that the medium used by ants to communicate information regarding shortest paths to food, consists of *pheromone trails* (Goss et al. [15]). A moving ant lays some pheromone (in varying quantities) on the ground, thus making a path by a trail of this substance. While an isolated ant moves practically at random, an ant encountering a previously laid trail can detect it and decide, with high probability, to follow it, thus reinforcing the trail with its own pheromone. The collective behavior that emerges is a form of *autocatalytic* process where the more the ants follow a trail, the more attractive that trail becomes to be followed. The process is thus characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that previously chose the same path. The algorithms of the *ACO* family are inspired by this process.

3.2.2 The algorithm

The *ACS* algorithm has been originally proposed by Gambardella and Dorigo in [10]. We apply this paradigm to the static vehicle routing like problems faced within a *DVRP*. The method is similar to that described in Gambardella et al. [11], which solves very efficiently the *VRP* with time windows.

In the remainder of this section we will not distinguish between customers and the depot, in order to simplify descriptions. A customer can then also indicate the a-th copy, d_a, of the depot (associated with vehicle a).

A solution retrieved by an ant is represented as a long, single tour. In this context, when node d_a is visited, a tour is considered as completed to the depot, and a new tour is started from the current position of vehicle a (i.e. the address of the last customer committed to a), at the starting time associated with vehicle a (i.e. the time a finishes to service the last customer committed to it). The capacity of a will also reflect its residual capacity.

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, the last visited customer, depends on the combination of two values:

- the attractiveness μ_{ij} of arc (i, j), as computed by some heuristic indicating the *a priori* desirability of that move. In our case $\mu_{ij} = tt_{ij}$, i.e. it depends directly 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:

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

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 μ_{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:

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

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).

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 maximum computation time for the local search, t_{ls}, must be specified.

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_acs)
    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 2: Pseudo-code of the *ACS* procedure used to solve static problems.

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

where ρ ($0 \leq \rho \leq 1$) is a parameter and *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 termination criterion is a maximum computation time of t_{acs} seconds. In the reality t_{acs} should be equal to the time slot, but it will be much shorter in the simulations presented in Section 4.

Pseudo-code of the *ACS* procedure for the static problems faced in a *DVRP*, is presented in Figure 2.

3.3 Pheromone conservation procedure

Once a time slot is finished and the respective static problem has been solved by an *ACS* algorithm, the pheromone matrix contains information about good solutions for this problems, i.e. pairs of customers which are visited in sequence in good solutions will have higher values in the corresponding entry of the pheromone matrix.

```
Algorithm ACS-DVRP

Time := 0;
VirtTime := 0;
PendOrds := orders known from the day before;
While (PendOrds ≠ ∅ or Time < T_co )
   If (Time > 0)
      VirtTime := Time + T_ac;
   EndIf
   StaticProb := problem with orders in PendOrds and starting time VirtTime;
   Run ACS on StaticProb;
   CommOrds := orders with processing time ≥ Time+T_ts+T_ac;
   Commit orders in CommOrd;
   PendOrds := PendOrds \ CommOrds;
   PendOrds := PendOrds ∪ {orders appeared in the last T_ts seconds};
   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 3: Pseudo-code of the *ACS-DVRP* algorithm.

This information should be somehow passed on to the static problem corresponding to the next time step, since the two problems are potentially very similar. This operation can prevent optimization to restart each time from scratch.

The strategy we follow to transfer information is inspired by Guntsch and Middendorf [16] and [17]. 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.

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

3.4 Pseudo-code

The solving strategy outlined in the previous subsections is summarized with the help of the pseudo-code presented in Figure 3.

The set *PendOrds* initially contains the orders known from the previous day. *Time* and *VirtTime* are

initialized to 0. An iterative statement is then entered. If $Time > 0$ then $VirtTime$ is updated in order to take into account the advanced commitment time. A static problem ($StaticPro$) is created and solved, and some commitments ($CommOrds$) are done accordingly to the solution of $StatitPro$. $PendOrds$ is updated together with starting positions, capacities and travel times of the vehicles. The pheromone matrix is finally updated. These operations are repeated until $PendOrds = \emptyset$ and $Time > T_{co}$. Finally the depot is committed as last destination to all vehicles.

3.5 Remark

An intrinsic property of dynamic problems must be pointed out. Every strategy based on the optimization of a sequence of static problems does not guarantee the best possible results to be achieved, even in case all the static problems are solved to optimality. Consider a static problem P solved to optimality (solution O). The commitment of some decisions is associated with this solution. Consider now a sub-optimal solution (S) of the same problem P, which implies the commitment of a completely different set of decisions. An ad-hoc new order C may now be created. It could be inserted without extra travel time into S, but not in O, because of the different commitments associated with the two solutions. The extra travel time paid to insert C into O may produce a very high total travel time at the end of the day, higher than the one obtained with solution S, vanishing consequently the effort to retrieve O instead of S for problem P.

On the other hand, as we have no information about future events, the best we can do is to apply a greedy strategy like the one described in this section.

Computational results presented in Section 4.3 will however confirm the effectiveness and the efficiency of our strategy.

4 Computational results

In this section some computational results are presented in order to evaluate the performance of the algorithm described in Section 3.

All the tests have been carried out on a personal computer equipped with a Intel Pentium 4 processor 1.5 GHz and 256 MB of memory. The algorithm has been coded in ANSI C.

Section 4.1 is devoted to the description of the benchmarks adopted, Section 4.2 will document the tuning of som parameters, while in Section 4.3 the results achieved by the *ACS* algorithm will be compared with those obtained by a multi-start local search algorithm.

4.1 Benchmarks description

The problems adopted in this paper have been originally proposed by Kilby et al. [19][3]. They are derived from some very popular static *VRP* benchmark datasets, namely 12 problems are taken from Taillard [26], 7 problems are from Christofides and Beasley [3] and 2 problems are from Fisher et al. [9]. These problems range from 50 to 199 customers. The number of customers can be inferred from the name of each instance.

Kilby et al. added to these problem the concept of *length of the working day*, and to each customer an *appearance time*, namely the time when the order becomes available (during the working day) and a *duration*, namely the time required to perform an order once reached the customer. They also fixed the number of available vehicles to 50 for each problem. More details can be found in Kilby et al [19].

Starting from these problems, a set of benchmark for the dynamic vehicle routing problem can be defined. Some parameters, not specified by Kilby et al. [19] must be fixed. They are T_{co} and T_{ac}, already introduced in Section 3.1. As a matter of fact, all the tests reported in [19] are about the change in the performance of the algorithm they propose, when different values for these parameters are considered. Taking into account these tests, we will set the parameters as described in the following paragraphs.

In the remainder of this section we will refer to the length of the working day as T.

The time after which the orders are postponed to the day after, T_{co}, is fixed to $T \times 0.5$, while the advance commitment time, T_{ac}, is fixed to $T \times 0.01$.

By fixing parameters as described above, a set of benchmarks has been defined. Our aim is to propose to the research community a set of widely available set of problems on which to compare the algorithms of different authors. Such a set of benchmarks does not exist for dynamic vehicle routing problems yet, and every author tends to use his/her own random data.

[3]The problems are available at http://www.cs.strath.ac.uk/~apes/apedata.html.

4.2 Parameter setting for the *ACS-DVRP* algorithm

From a previous study by Gambardella et al. [11], it is known that a good parameter setting for *ACS* algorithms applied to 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. We use these settings and in particular we initially solve each static problem with a post-insertion algorithm (like the one described in [11]) in order to obtain *Cost(PI)*, and consequently τ_0.

In the simulations reported in this paper we also mapped the working day of each problem into 1500 seconds of CPU time, in order to shorten the total computation time. This choice affects the setting of some parameters, as described in the following paragraphs.

We decided to maintain constant at 6 the ratio between t_{acs} and t_{ls} (some preliminary tests support this choice). Consequently we adopt the following setting for the remaining of this section: $t_{acs} = \frac{1500}{n_{ts}}$ seconds and $t_{ls} = \frac{250}{n_{ts}}$ seconds.

Two more parameters have to be specified: n_{ts} and γ_r. Parameter n_{ts} specifies the number of time slots in which the working day is divided. It is a crucial parameter for algorithm *ACS-DVRP*, because 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 recent information are ignored. In this case very good local minima might me reached for the problems investigated, but these problems do not contain updated information, and consequently the optimization effort is vanished.

In the tests presented in Table 1 we have set $\gamma_r = 0.3$, according to some preliminary tests (the tests summarized in Table 2 will confirm that $\gamma_r = 0.3$ is a good choice).

Three different values for n_{ts}, 10, 25 and 50, are considered in Table 1 for three benchmarks. For each of these values, five runs of algorithm *ACS-DVRP* have been carried out and, for each pair (*problem*, n_{ts}) three values are reported: *Min*, *Max* and *Avg*, which respectively represent the best, the worst and the average total travel times found over the five runs.

Table 1 suggests that $n_{ts} = 25$ guarantees the best performance of algorithm *ACS-DVRP* on the benchmarks considered. This setting will be adopted in the remainder of this section.

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

n_{ts}	t_{acs}	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

The last parameter which requires to be specified is γ_r, the parameter used within the pheromone conservation procedure (see Section 3.3). We carried out some tests, summarized in Table 2.

In Table 2 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 and for each pair (*problem, value of* γ_r) three values 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, the 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 it results that 0.3 is the best value for parameter γ_r. In the remaining tests γ_r will consequently be set to 0.3.

The gap between the results obtained with $\gamma_r = 1.0$ and $\gamma_r = 0.3$ gives also a measure of the contribution to the performance of the algorithm, of the pheromone conservation strategy described in Section 3.3.

4.3 Results

In Table 3 the results achieved by the *ACS* algorithm described in Section 3 are compared with the results obtained by a multi-start local search algorithm obtained from the *ACS* algorithm by not updating the pheromone levels. This algorithm will be referred to as *No pheromone* because in fact it is the *ACS* algorithm *without* pheromone (i.e. $\rho = 0$).

Comparison with algorithms of other authors is not possible, as nobody else used the benchmarks we adopt. The only exception is represented by Kilby et al. [19], where the problems are the same, but no numerical result is reported.

For each problem, five runs of each algorithm have been considered (the computation times considered have been specified in Section 4.2). In Table 3 the best (*Min*), the worst (*Max*) and the average (*Avg*) travel time retrieved over the five runs is reported. Row *Total* reports the sum over each column.

Table 3: Computational results.

Problem	No pheromone			ACS		
	Min	Max	Avg	Min	Max	Avg
c100	1080.33	1169.67	1124.04	973.26	1100.61	1066.16
c100b	978.39	1173.01	1040.99	944.23	1123.52	1023.60
c120	1546.50	1875.35	1752.31	1416.45	1622.12	1525.15
c150	1468.36	1541.54	1493.06	1345.73	1522.45	1455.50
c199	1774.33	1956.76	1898.20	1771.04	1998.87	1844.82
c50	693.82	756.89	722.15	631.30	756.17	681.86
c75	1066.59	1142.32	1098.85	1009.38	1086.65	1042.39
f134	16072.97	17325.73	16866.79	15135.51	17305.69	16083.56
f71	369.26	437.15	390.48	311.18	420.14	348.69
tai100a	2427.07	2583.02	2510.29	2375.92	2575.70	2428.38
tai100b	2302.95	2532.57	2406.91	2283.97	2455.55	2347.90
tai100c	1599.19	1800.85	1704.40	1562.30	1804.20	1655.91
tai100d	2026.82	2165.39	2109.54	2008.13	2141.67	2060.72
tai150a	3787.53	4165.42	3982.24	3644.78	4214.00	3840.18
tai150b	3313.03	3655.63	3485.79	3166.88	3451.69	3327.47
tai150c	3090.47	3635.17	3253.08	2811.48	3226.73	3016.14
tai150d	3159.21	3541.27	3323.57	3058.87	3382.73	3203.75
tai75a	1911.48	2140.57	2012.13	1843.08	2043.82	1945.20
tai75b	1634.83	1934.35	1782.46	1535.43	1923.64	1704.06
tai75c	1606.20	1886.24	1695.50	1574.98	1842.42	1653.58
tai75d	1545.21	1641.91	1588.73	1472.35	1647.15	1529.00
Total	53454.54	59060.81	56241.50	50855.94	57645.52	53784.02

Table 3 provides a measure of the improvements guaranteed by the use of pheromone (i.e. *ACS* algorithm). The results contained in the last row indicate that the best results found over the five runs has improved of

the 4.86% by using *ACS*, the worst results of the 2.40%, and the average of the 4.37%.

It is also interesting to observe that the best solution found by the *ACS* algorithm over five runs, is always better of the one retrieved by the algorithm which does not use pheromone. The same happens for the average travel time.

Finally, it is important to observe that better results on the static *VRP*s (obtained by using pheromone, i.e. *ACS* algorithm), lead to better results for the *DVRP*s, confirming the effectiveness of our strategy (see Section 3.5).

5 A case study

In this section the *ACS-DVRP* algorithm is tested on a realistic case study, set up in the city of Lugano, Canton Ticino, Switzerland.

In Figure 4 the road network of Lugano is depicted. For each arc r of the network an interval $[T_r^{lb}, T_r^{ub}]$ of possible travel times is available.

A depot (white square in Figure 4) and 50 customers (black circles in Figure 4) have been randomly chosen. Travel times among them have been calculated as robust shortest path using the algorithm described in Montemanni and Gambardella [21][4]. 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 randomly appear during the working day with random requests ranging in $[1, 31]$. A fleet of 50 vehicles with a capacity of 160 is finally considered. Also in this case $T_{co} = T \times 0.5$ (i.e. $T_{co} = 14400$ seconds) and $T_{ac} = T \times 0.01$ (i.e. $T_{ac} = 288$ seconds). Parameter n_{ts} will be varied, being part of the study we propose.

Parameters q_0, β, ρ and γ_r and m are set up as described in Section 4.2. Parameters t_{acs} and t_{ls} will vary.

In Table 4 we presents the results obtained by the *ACS-DVRP* algorithm on different experiments, where the number of time slots (namely parameter n_{ts}) is varied. Parameters t_{acs} and t_{ls} are adjusted according to the values of n_{ts} (i.e. $t_{acs} = T_{ts} = \frac{28800}{n_{ts}}$ and t_{ls} is set up consequently). In particular the ratio between t_{acs} and t_{ls} is, in this case, always around 10.

In Table 4 the first three rows define the setting of the experiments, i.e. the values of parameters n_{ts}, t_{acs}

[4]For every arc r on a selected path, the travel time considered is T_r^{ub}.

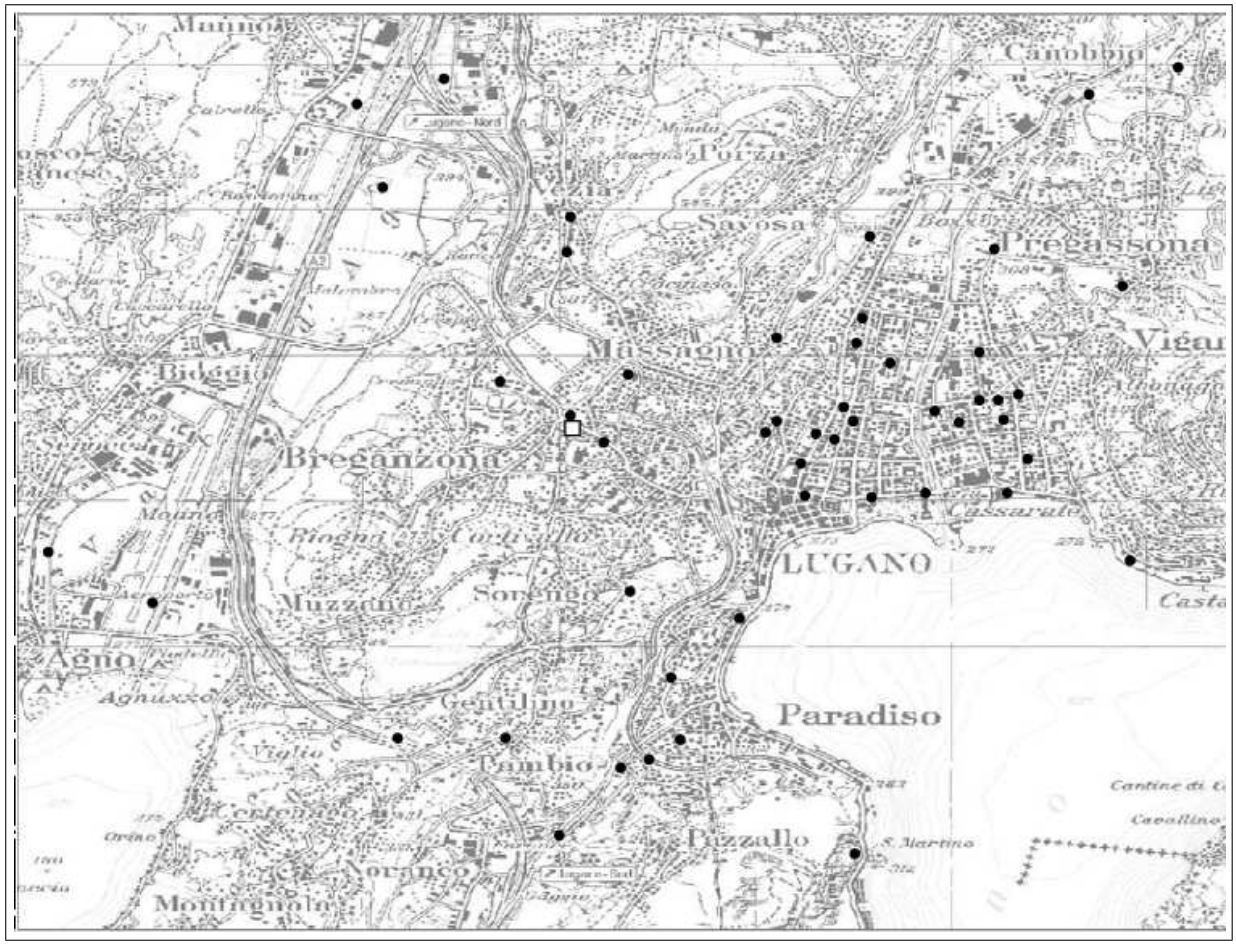

Figure 4: Case study in the city of Lugano.

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
t_{acs}	144	288	576	1152	2880	5760
t_{ls}	15	30	60	120	240	480
Travel time	12702	12422	10399	9744	10733	11201

Table 4 suggests that, for the case study analyzed, good values for n_{ts} are between 10 or 50. In particular 25 seems to be the best choice (the same value 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 slots. A sequence of static vehicle routing problems is then generated. An Ant Colony System algorithm has been used to solve these problems. A technique to transfer information about good solutions, from a static problem to the following one has been developed.

Some benchmarks have been defined and used to test the performance of the algorithm we propose.

Computational results confirm that performance of our algorithm are strictly connected with the strategy to transfer information about good solutions we developed. Other computational results suggest that the algorithm we propose is very efficient, also when compared with other heuristic techniques.

A case study, set up in the city of Lugano, confirms that our method can be applied to real world instances.

Acknowledgements

Data on the road network of Lugano have been provided by the Commissione Regionale dei Trasporti del Luganese (CRTL).

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, July 2000.

[2] M. Caramia, G.F. Italiano, G. Oriolo, A. Pacifici, and A. Perugia. Routing a fleet of vehicles for dynamic combined pick-up and deliveries services. Universitá di Roma "Tor Vergata", 2002.

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

[4] 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.

[5] L. Coslovich, R. Pesenti, and W. Ukovich. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. Universitá degli Studi di Trieste, 2002.

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

[7] 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.

[8] M. Facchetti and M. Salani. Sviluppo di modelli, algoritmi di ottimizzazione e di un'architettura modulare a componenti per un sistema "dial-a-ride". Tesi di Laurea. Universitá degli Studi di Milano, June 2001. In Italian.

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

[10] 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.

[11] 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.

[12] M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin. Neighborhood search heuristic for a dynamic vehicle dispatching problem with pick-ups and deliveries. Université de Montréal.

[13] 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, November 1999.

[14] 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.

[15] S. Goss, S. Aron, J.-L. Doneubourg, and J.M. Pasteels. Self-organized shortcuts in the argentine ants. *Naturwissenschaften*, 76:579–581, 1989.

[16] 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.

[17] 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.

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

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

[20] S.O. Krumke, J. Rambau, and L.M. Torres. Real-time dispatching of guided and unguided automobile service units with soft wime windows. Technical Report 01-22, Konrad-Zuse-Zentrum für Informationstechnik Berlin, September 2001.

[21] R. Montemanni and L.M. Gambardella. An algorithm for the relative robust shortest path problem with interval data. Technical Report IDSIA-05-02, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, February 2002.

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

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

[24] M.W.P. Savelsbergh and M. Sol. DRIVE: Dynamic Routing of Independent VEhicles. Georgia Institue of Technology.

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

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