On-line and off-line algorithms for the Dial a Ride Problem

Alessio Soldano, Fabiana Valandro

1 Introduction

The modern society constantly needs facilities for people transport and carriage of goods. Public transport services don't provide the required flexibility, dynamism and timeliness, so people prefer to use private means of transport, e.g. cars. This leads to traffic congestion in big cities and reduces population quality of life due to air and sound pollution.

This work deals about innovative transport systems that make the most of available resources, trying in same time to meet the requirements of every single customer. In particular it will focus on the emerging *Dial a Ride* system which represents an interesting solution to the problem described above. A fleet of vehicles, without fixed routes and schedules, carries people from the desired pickup point to the desired delivery point, during a prespecified time interval. The customers demand the service in calling a central unit and in specifying: the desired pick-up and delivery points (respectively, *origin* and *destination*), the number of passengers and some limitations on the service time (e.g., the earliest departure time or the latest arrival time). The central unit processes the requests, plans the vehicle routes and tells each customer if his request has been accepted or rejected. The planning of routes in a system like this is a *vehicle routing and scheduling problem* and needs a proper algorithm to be solved.

Due to their high complexity, these problems have been tacked in literature mainly through heuristic algorithms; in this work a *granular tabu search heuristic* will be used for the first time to solve the *Dial a Ride* problem. A system able to deal with all the main versions of this problem will be realized: customers can ask for the service in advance (*off-line* version) or call during the service time (*on-line* version); moreover the features of the road network

the problem is modeled on can be considered constant (*non time-dependent* version) or variable as a consequence of different traffic volumes during the day (*time-dependent* version). The system will prove to be effective and efficient in all conditions: for this reason, through some future developments, it will be possible to use it for practical applications in a real context.

The document is organized as follows: Section 2 describes the main features of the *vehicle routing and scheduling problems* and the algorithms proposed in literature to solve them; Section 3 formalizes the *Dial a Ride* problem; Section 4 describes the algorithm realized in details; finally, Section 5 presents computational results and conclusions.

2 Vehicle routing and scheduling problems

A *vehicle routing problem* models every situation in which some vehicles have to serve a set of requests. For each request there are two points in the space: they have to be visited by a vehicle and a service (for example load or unload of goods or people) has to be carried out there [1].

Routing problems have different kinds of constraints. There are *capacity constraints* if the vehicle providing the service has a finite load capacity: routes (which include load and unload points) have to be planned so that the maximum capacity is never exceeded.

In many cases, the vehicles are not available during the whole day and every point of a request has a *time window* indicating when the service has to be carried out [2]. A routing problem with these *time constraints* becomes a *routing and scheduling problem.*

A problem is called *many-to-many* if there are different pick-up and delivery points for all the requests; on the contrary if the pick-up (or delivery) point is always the same the problem is *one-to-many* (*many-to-one*).

To solve a *routing and scheduling problem* a subset of requests is chosen and assigned to each vehicle; then for each vehicle a route passing through all the points of the selected requests is constructed [3]. The solution composed by all the routes has to respect the constraints of the problem and to optimize one or more features like the duration or length of the routes, the number of vehicles, the travel time, the customer satisfaction or the profit of the company providing the service [2].

There are different kinds of *routing and scheduling problems*: a complete and accurate classification, as in [4], is far from the aims of this work, so only their main features will be presented here.

A vehicle routing problem is a *static optimization problem* when all the information required for its solution are available before the start of the resolution process [5][2]. On the contrary, in many real cases it's necessary to use a model based on a *dynamic optimization problem*: solutions must be found while time proceeds, concurrently with incoming information. This means that no *a-priori* solution can be found; nevertheless at the start of the optimization process it's possible to chose a *strategy* specifying what actions should be taken as a function of the system state [6].

Most of the models described in literature for vehicle routing problems are *not time-dependent*: they assume constant travel times during the day on the road network. In fact travel times between fixed points vary during the day, so *time-dependent* network models have been proposed: travel time depends not only on the distance traveled but also on the time of the day. Different speed are considered at different hours: in this way it's possible to take into account the traffic volume variations during the day [7][8][9].

2.1 Main problems

The routing and scheduling problems are always modeled on a *graph*. A graph $G = (N, A)$ consists of a set of *vertices* (or *nodes*) N and a set of *arcs* (or *edges*) A. Each element a of A connects two elements i and j of N. If each arc a is an ordered pair of nodes, the graph is *directed*. A sequence of consecutive edges connecting two vertices is called *path*. A graph is *connected* if there is a path for each pair of nodes. Finally a graph is *strongly connected* if there is an arc between each pair of nodes. Vertices and edges may have *weights* indicating for example distances or travel times.

Two different kinds of graph will be considered:

- *physical graph*: it's a model of the road network the considered problem is defined on. There's a node for every interesting place and an arc for each road between these places.
- *abstract graph*: it is used for the mathematical formulation of routing and scheduling problems; there's a node for every request point and arcs represent the shortest paths between them.

The main routing problems described in literature are:

- *Traveling Salesman Problem* (TSP): it's the problem of a salesman who has to visit his customers minimizing the travel cost. A single vehicle starts from the depot, reaches some nodes and then comes back to the depot; there are no time or precedence constraints.

- *Multiple Traveling Salesman Problem* (m-TSP): it's a TSP with a multiple vehicle fleet. The set of requests is divided into subsets and each of them is served by a vehicle. Every route starts and ends at the depot. Often one of the problem's objectives is to minimize the number of used vehicles.

- *Vehicle Routing Problem* (VRP): it's the m-TSP where a demand is associated with each customer and each vehicle has a finite capacity. The demands may be all loads or all unloads; vehicle loads must not exceed vehicle maximum capacities.

- *Pick-up and Delivery Problem* (PDP): it's a *many-to-many* version of the VRP. Each request has a source point where some goods have to be picked up and a destination point where they have to be delivered. For this reason there're both capacity and precedence constraints.

These *routing problems* become *routing and scheduling problems* if all the request points have a time window.

2.1.1 Dial a Ride Problem

Demand-responsive transportation systems use vehicles whose routes and schedules change dynamically on the basis of the actual requests of users. By better exploiting vehicle capacity, they try to offer the comfort and flexibility of private cars and taxis at a lower cost. The *Dial a Ride* (DAR) is the more complete and flexible of these systems; it is suited to service sparsely populated areas, densely populated areas during weak demand periods or special classes of passengers with specific requirements (elderly, disabled).

The *Dial a Ride* can be modeled as a routing and scheduling problem with time windows almost like the PDPTW; the only difference is the object of the transport, that is people in DAR and goods in PDPTW.

Figure 1 represents the main elements defining the structure of a *Dial a Ride* system. The *fleet* is the group of vehicles available, defined by the service time, their capacity and the depot they belong to. The *requests* describe the customer demand with the origin and destination points, the number of passengers and the earliest departure time (or latest arrival time). The *constraints* regard the maximum capacity of the vehicles, that cannot be exceeded, and the precedence in visiting the pick-up point of each request before visiting its delivery point. Moreover some constraints assure the customers a minimum quality of service, avoiding them to wait too long before the pick-up and providing them with a trip as short as possible. Finally

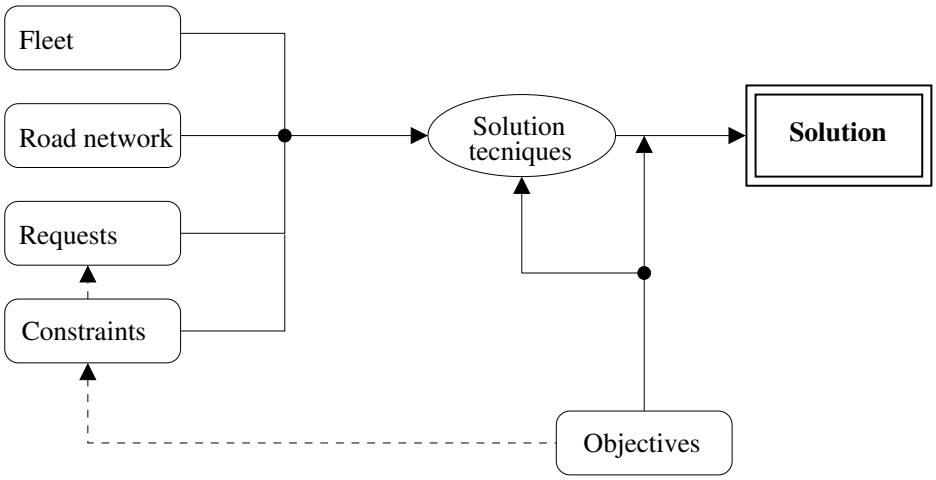

Figure 1: Structure of a *Dial a Ride* system.

the *objectives*: the *Dial a Ride* involves different interests and requirements. Customers are interested in a good quality of service, the company providing the service would like low running costs, while the public administration is interested in a service available in every places of the region. For these reasons different conflicting objectives have to be considered: the maximization of the quality of service, the minimization of the number of used vehicles, the maximization of the number of served customers.

2.2 Solution of a static routing and scheduling problem

The vehicle routing and scheduling problems can be solved using techniques based on two different approaches: *exact* and *heuristic* ones. Exact methods compute every possible solution until one of the best ones is reached, while heuristic methods produce good (non optimal) solutions within short time.

Exact methods are based on implicit enumeration algorithms like *dynamic programming*, *branch and bound* and *column generation*.

In [10] and [11] two exact methods based on dynamic programming and column generation are proposed for the *single-vehicle* DAR and the PDPTW resolution. These methods are unable to solve problems with several requests.

All the *vehicle routing and scheduling problems* have very high complexity; as a matter of fact Savelsbergh [12] proved that the TSPTW is an $\mathcal{NP}$-hard problem. This means, as a consequence, that both VRPTW and DAR are $\mathcal{NP}$-hard.

The results of the work mentioned above show that, due to this high com-

plexity, exact methods can't deal with large problems and thus are useless for real life applications. For this reason the research focused on heuristic methods which can be divided into two main groups: *classical heuristic* and *metaheuristic* algorithms.

2.2.1 Classical heuristic algorithms

Classical heuristics have been developed starting from 1960 and up to 1990 [13]. They are very simple and are based on both theory and common sense rules. Heuristic algorithms perform a relatively limited exploration of the search space and typically produce good quality solutions within modest computing times.

Constructive methods These methods gradually build a feasible solution while keeping an eye on its cost, but do not contain an improvement phase *per se*; they generate routes inserting request one at a time. They may be *sequential* if a route has to be completed before opening a new one, or *parallel* if many paths are constructed at the same time [14][15].

These methods can also follow a different approach in which two phases are considered: a *clustering phase* creates sets of requests that can be served by a single vehicle, a *routing phase* constructs the actual routes. Algorithms based on this approach are called *cluster-first, route second* or *route-first, cluster second* depending on the execution order of the two phases.

Interesting solutions using constructive methods have been proposed in [16][17] and [18].

Improvement methods Improvement methods try to optimize a solution made up of feasible routes and obtained for example through a constructive method. Most of the improvement methods for vehicle routing and scheduling problems are *local research* algorithms. They are based on the concept of *neighborhood* of a solution that is the set of all the feasible solutions reachable from it through a local transformation (*move*) [18][15]. A move is performed through an exchange: the position of one or more vertices in one or more routes is changed, therefore some arcs are removed from the solution while other ones are added to it. Figures 2 and 3 show the possible exchanges involving two arcs.

There are two neighborhood exploration strategies: *first-accept*, according to which the current solution is replaced by the first solution found having a better objective function value, and *best-accept*, according to which all the

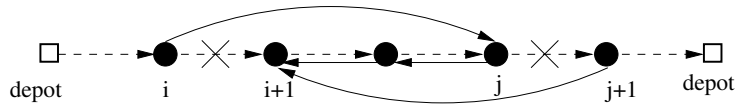

Figure 2: Exchange involving 2 arcs, on a single route.

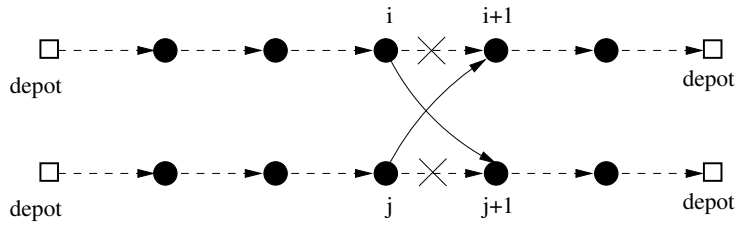

Figure 3: Exchange involving 2 arcs between two routes.

neighborhood's solutions are tested and then the one causing the maximum improvement is selected [14].

2.2.2 Metaheuristic algorithms

Many metaheuristic algorithms have been proposed and successfully used in the last decade [19]. These techniques differ from classical heuristics because they accept *worsening* and *unfeasible* solutions during the research process. In this way these methods perform a deep exploration of the most promising regions of the solution space, trying not to stop at the occurrence of the first local minimums. Metaheuristics produce better solutions than classical heuristics do.

The main elements of some famous metaheuristics applied to vehicle routing and scheduling problems are described in the following paragraphs; for a complete and in-depth analysis of all the methods refer to [20].

Ant Colonies Systems The *Ant Colony System* (ACS) algorithms are based on a computational paradigm inspired by the way real ant colonies function. Ants communicate information regarding shortest path to food through pheromone trails they create when moving on the ground. 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. For this reason, after a short time the best paths are marked with a high level of pheromone.

This led to a computational paradigm in [21] for the resolution of optimization $\mathcal{NP}$-hard problems. Ants are represented by simple computational

agents which individually and iteratively explore the solution space. The objective function value depends on many elements, including the level of the trails followed by the agents (ants).

Interesting ACS algorithms have been proposed for the resolution of the TSP [21][22], the VRP [23][24][25] and the VRPTW [26].

Genetic Algorithms *Genetic Algorithms* (GA) employ the mechanics of natural selection and natural genetics to evolve solutions to problems. They don't work with a single solution but evolve a population of candidate solutions by creating new generations of offspring. Each solution is encoded as a string of bits (*chromosomes*) and has a *fitness* function indicating its reproduction probability. *Mutation* and *crossover* may follow the reproduction of the solutions: a single bit of a string is changed or whole sequences of bits are swapped between solutions.

For solving a vehicle routing and scheduling problem, each solution is represented by one chromosome which is a chain of integers, each of them representing a customer or a vehicle. Mutation and crossover must be defined so that the problem's constraints are always respected after every change [27][28].

Genetic Algorithms have been successfully used mainly for the TSP resolution [27]; in [29] a VRPTW is solved through an hybrid system based on a genetic algorithm and a *greedy* heuristic.

Simulated and Deterministic Annealing The *Simulated Annealing* (SA) derives from the statistical mechanics. It is based on an analogy from the annealing process of solids: a solid is heated to a high temperature and then gradually cooled in order for it to crystallize in a low energy configuration.

The energy level represents the objective function, while temperature T indicates the dimension of the portion of solution space the algorithm can explore. The solution is modified step by step through *random* perturbations; changes improving the objective function value are always accepted, worsening changes are accepted with probability $e^{-\frac{\Delta}{T}}$. The algorithm starts with a high T value; then it is gradually reduced according to a *cooling scheduling* (which also defines the algorithm's stopping criteria).

The *Deterministic Annealing* (DA) differs from the *Simulated Annealing* because of its deterministic rule for the changes acceptance.

Some interesting SA algorithms for the VRP [30] and the VRPTW [31] have been proposed in literature.

Tabu Search The *Tabu Search* (TS) metaheuristic, invented by Glover [32][33], performs a deep exploration of the solution space, allows worsening solutions and thus tries to escape from local minimums. The optimization process starts from an initial solution usually built through a constructive heuristic. At each iteration of the algorithm the best move[1] of the current neighborhood is selected and executed. To avoid cycling a memory called *tabu list* is used: all the solutions that have been recently visited are classified as forbidden (*tabu*) for some iterations (*tabu tenure*). A forbidden move can be executed if it leads to a solution with an objective function value better than any other found during the research process (*aspiration criteria*). In order to improve the effectiveness of the TS method, *intensification* and *diversification* techniques try to focus the research on promising portions of the solution space or to move it to other ones [34].

Many different algorithms exploiting the *Tabu Search* method have been proposed for the VRP [35][19]; just some of the most interesting ideas and algorithms proposed for the solution of several vehicle routing and scheduling problems are presented here:

- In [36] an *Adaptive Memory Procedure* is used to solve both the VRP and the VRPTW. A set of good solutions is created during the research process; at intervals some elements (representing the vehicle routes) are taken from this set and recombined to build a better solution. This technique can be seen as a *diversification* strategy if used during the optimization process or as an *intensification* strategy if used at the end of it.

- In [37] an interesting TS algorithm for the DAR is described. The presented research process allows *unfeasible solutions* through some penalizations on the objective function. Moreover an effective *diversification* technique based on the *frequency* of moves execution is introduced.

- The *Granular Tabu Search* (GTS) proposed in [38] is a promising version of the standard TS: it provides solutions almost as good as the ones produced by TS but requires less computation time. This is achieved using a drastically restricted neighborhood (*granular neighborhood*) obtained from the standard one by removing the moves that involve only elements which are not likely to belong to high-quality feasible solutions. The authors applied this technique to the VRP; they observed that arcs whose length is higher than a *granularity threshold* are not likely to

[1] The move leading to the best solution reachable from the current one.

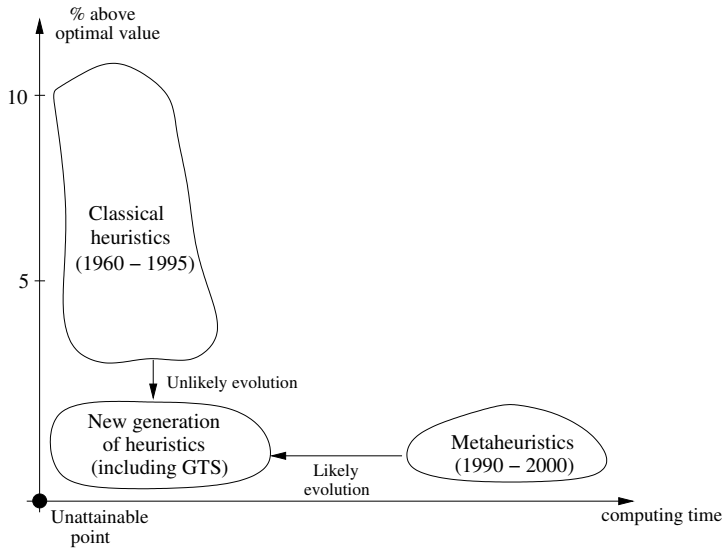

Figure 4: Evolution of heuristics

belong to high quality solutions and thus should not be considered in the research process.

2.2.3 Conclusions

The choice of the optimization algorithm to be used in this work for the DAR problem was based on the analysis of the presented resolution methods for the static *vehicle routing and scheduling problems*. In particular the *Granular Tabu Search* was chosen due to its great efficiency and applied for the first time to the DAR. We tried to realize an algorithm both effective and efficient, following the evolution of the VRP heuristic resolution techniques described in [19] and shown in Figure 4.

In this way it was possible to deal with all the DAR problem's versions, *static* and *dynamic*, *time-dependent* and *not time-dependent*.

2.3 Solution of a dynamic routing and scheduling problem

Most of the algorithms for the resolution of *dynamic routing and scheduling problems* make use of the approaches described for the static ones. In particular *real-time* context requires the generation of solutions within short time, thus heuristics and metaheuristics become absolutely necessary [39][3], while exact methods are useless [40].

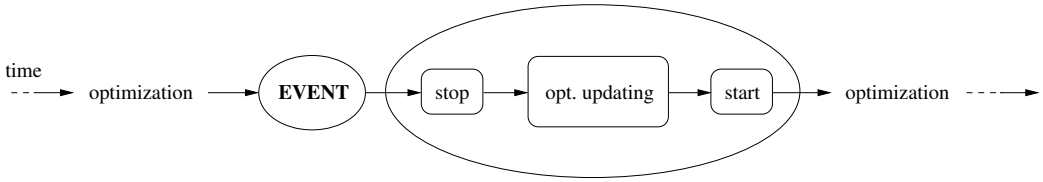

Figure 5: *Single Event Optimization* strategy

The resolution of a dynamic routing and scheduling problem requires the definition of a *strategy* indicating which actions have to be performed depending on the current state of the system. It says how a dynamic problem can be split into a set of static sub-problems which will be solved separately [6].

The simplest strategy is called *Single Event Optimization*: a new static problem is defined every time an *event* occurs; an event can be the occurrence of a new request or the end of service at a customer location. Each sub-problem contains all the unserviced requests known at the current time. The operation of a system based on this strategy is shown in Figure 5: the optimization procedure (which can be a TS applied to a dynamic VRPTW [41] or dynamic PDPTW [42]) solves the series of static problems running between couples of events [6].

Many other strategies have been proposed, including *First Come First Served policy*, *Stochastic Queue Median policy*, *Nearest Neighbor policy* and TSP *policy* [43].

2.4 Solution of a time-dependent routing and scheduling problem

Time-dependent vehicle routing and scheduling problems are based on network models according to which the time needed to go from a node to another one changes during the day. Due to different traffic volumes at different hours, the travel time between two locations depends on both the distance and the departure time [8].

Time-dependent problems can be solved using the same algorithms described in 2.2, but suitable techniques and a time-dependent network model are needed to properly compute travel times.

In most of the time-dependent models the horizon of interest is discretized into small *time intervals* (or *time fences*) [8][9][7]. Then travel time [8] or speed [9][7] for each network's arc is assumed to be step function of the starting time at the origin node. In this way these models try to approximate real world conditions where travel times vary continuously over time.

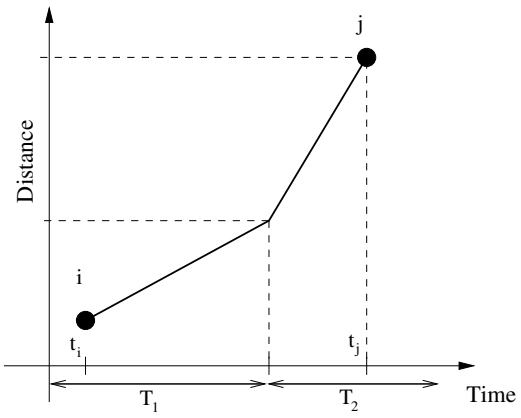

Figure 6: Changing travel speed over time

All the models based on discrete travel times and many of those based on discrete speeds proved to be ineffective as they don't respect the FIFO *property* [7]. This property, also known as *non-passing property* (NPP), implies that two vehicles traveling on the same network's arc will arrive at its end in the same order as they start, even if some congestion occurs during the travel.

Ichoua, Gendreau and Potvin suggested a time-dependent network model consistent with the NPP [7]: the main point is that they do not assume a constant speed over the entire length of an arc; rather speed changes when the boundary between two time intervals is crossed. Figure 6 shows the distance run over arc (i, j) by a vehicle starting at t_i in time interval T_1 and arriving at t_j in time interval T_2. The authors provided a travel time calculation procedure that was also used in [44] for a modified Dijkstra algorithm able to compute shortest paths on time-dependent networks.

3 Dial a Ride problem definition

This section formalizes the DAR problem and presents the procedures used to obtain all the information required for its solution (time windows, distances and travel times between request points).

3.1 DAR problem formulation

Let $U = \{1 \dots n\}$ be a set of requests (customers). For each request i two nodes (i^+ and i^-) are defined: a load q_i must be taken from i^+ to i^-. Let

$N^+ = \{i^+ | i \in U\}$ be the set of pick up nodes and $N^- = \{i^- | i \in U\}$ denotes the set of delivery nodes. A positive amount $q_{i^+} = q_i$ is associated to the pick up node, a negative amount $q_{i^-} = -q_i$ to the delivery node. A time window is also associated to each node, both to a pick up node $[e_{i^+}, l_{i^+}]$ and to a delivery node $[e_{i^-}, l_{i^-}]$. The fleet of vehicles is denoted as M; all vehicles have the same capacity Q and time window $[e_0, l_0]$.

Let $G(N, A)$ be a directed graph, whose set of vertices is defined as $N = N^+ \cup N^- \cup \{0\}$, where node 0 represents the vehicle depot. The set of arcs A is defined as $A = \{(i, j) : i, j \in N, i \neq j\}$ and each arc $(i, j) \in A$ has associated a distance $d_{i,j}$, a travel time $t_{i,j}$ and a cost function $c_{i,j}$. Another set $E = \{(i, j) : i, j \in N^+ \cup N^-, i \neq j\}$ represents the subset of arcs whose extremes are customer nodes.

The problem is to find a set of routes starting and ending at the depot, such that all the customers are satisfied and the pick up node of each customer is visited before the delivery node. Moreover, the solution should be feasible with respect to the capacity and the time window constraints. The variables $x^m_{i,j}$ are equal to 1 if vehicle m uses arc $(i, j) \in A$ and equal to 0 otherwise; p_i represents the departure time from node $i \in N^+ \cup N^-$; y_i is the load of the vehicle leaving node i. The problem's constraints are:

$$\sum_{m \in M} \sum_{j \in N} x^m_{i,j} \leq 1 \qquad \forall i \in N^+ \tag{1}$$

$$\sum_{j \in N} x^m_{i,j} - \sum_{j \in N} x^m_{j,i} = 0 \qquad \forall m \in M, \quad \forall i \in N^+ \cup N^- \tag{2}$$

$$\sum_{j \in N} x^m_{i^+,j} - \sum_{j \in N} x^m_{i^-,j} = 0 \qquad \forall m \in M, \quad \forall (i^+, i^-) \in N^+ \cup N^- \tag{3}$$

$$x^m_{i,j}(y_i + q_j) \leq y_j \qquad \forall m \in M, \quad \forall (i, j) \in E \tag{4}$$

$$q_i \leq y_i \leq Q \qquad \forall i \in N^+ \tag{5}$$

$$0 \leq y_i \leq Q - q_i \qquad \forall i \in N^- \tag{6}$$

$$x^m_{i,j}(p_i + t_{i,j}) \leq p_j \qquad \forall m \in M, \quad \forall (i, j) \in E \tag{7}$$

$$e_i \leq p_i \leq l_i \qquad \forall i \in N \tag{8}$$

$$p_{i^+} + t_{i^+,i^-} \leq p_{i^-} \qquad \forall i = \left(i^+, i^-\right) \in N^+ \cup N^- \tag{9}$$

$$x^m_{i,j} \in \{0, 1\} \qquad \forall m \in M, \quad \forall (i, j) \in A \tag{10}$$

The first three groups of constraints ensure that each customer is serviced by at most one vehicle. Indeed, constraints (1) make sure that, at most,

one vehicle exits from each origin node i^+, constraints (2) impose that the number of vehicles entering and exiting each node be the same, and constraints (3) that the same vehicle, if any, visits the pickup and the delivery node. Constraints (4), (5) and (6) ensure the feasibility of the loads. The number of passengers in a given vehicle varies according to the number of people boarding it or getting off it. The maximum capacity of the vehicle cannot be exceeded. The last three classes of constraints impose the feasibility of the schedule. Constraints (7) represent the compatibility requirements between routes and schedules. Constraints (8) ensure that the departure time takes place during the time window: when the vehicle arrives at node i before e_i the driver must wait; it is unfeasible to arrive at node i after l_i. Finally, constraints (9) imply that for each trip the delivery node is visited after the pickup node. Constraints (7) and constraints (4) can be linearized respectively as $M(1 - x_{ij}^m) \geq p_i + t_{ij} - p_j$ and $M(1 - x_{ij}^m) \geq y_i + q_j - y_j$. The former equations are a generalization of the classical *TSP subtour* elimination constraints proposed by Miller, Tucker and Zemlin [45].

3.1.1 Objective function

As shown in 2.1.1, *Dial a Ride* involves different interests and requirements, thus is a *multi-objective optimization problem*. The usual concept of optimal solution does not apply directly in the multi-objective case. Two kinds of solution can be found: *dominated* solutions, whose objective function values are all worse than another solution's ones, and *Pareto optimal* (*efficient*) solutions, which have a better value for at least one of the objective functions.

A positively weighted convex sum of the objectives has been minimized to solve the DAR problem described in this work:

$$\min \left(\alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3 \right) \tag{11}$$

The weights α_1, α_2 and α_3 are assigned decreasing values.

The primary objective has to guarantee that many customers can make use of the service; the number of served requests is given by

$$\sum_{i \in N^+} \sum_{m \in M} \sum_{j \in N} x_{i,j}^m \tag{12}$$

thus the first term of the objective function[2] is

$$f_1 = \frac{|U| - \sum_{i \in N^+} \sum_{m \in M} \sum_{j \in N} x_{i,j}^m}{|U|} \tag{13}$$

[2]The value has been normalized between 0 and 1.

The second objective tries to meet the company's expectations minimizing the number of vehicles used. The value is normalized between 0 and 1:

$$f_2 = \frac{\sum_{m \in M} \sum_{j \in N^+} x_{0,j}^m}{|M|} \tag{14}$$

Finally the third term of the objective function measures the average *level of service* (LOS) over the set of serviced customers. It is given by the difference between the service time offered by the DAR system for the trip (that includes travel times and waiting times) and the minimum time the customer would need to go from the origin i^+ to the destination i^- [1]. More precisely, the LOS for customer i is defined as

$$\text{LOS}_i = \frac{D_{i^+} + \sum_{k=i^+}^{i^- - 1} t_{k,k+1} + \sum_{k=i^+}^{i^-} w_k}{t_{i^+,i^-}} \tag{15}$$

where D_{i^+} is the departure delay and w_k are the waiting times. The minimum value of LOS_i is reached when the vehicle departs from i^- at time e_{i^-}, while the maximum one is reached when the vehicle leaves node i^- at the end of the time window (l_{i^-}). This means that the normalized LOS for customer i can be defined as [1]

$$\text{LOS}_i = \frac{p_{i^-} - e_{i^-}}{l_{i^-} - e_{i^-}} \tag{16}$$

so the third term to be considered in the objective function is

$$f_3 = \frac{\sum_{i \in N^+} \sum_{m \in M} \sum_{j \in N} \left(x_{i,j}^m \text{LOS}_i \right)}{\sum_{i \in N^+} \sum_{m \in M} \sum_{j \in N} x_{i,j}^m} \tag{17}$$

3.2 Time windows computation

The time windows computation process must guarantee that a vehicle moving from i^+ to i^- breaks no constraint. Moreover the service time offered by the DAR system for a trip should be quite as short as the one the customer would need to go directly[3] from the origin to the destination.

Let MWT be the maximum waiting time at the origin point of a request and maxLOS a measure of the minimum level of service granted to a customer[4].

[3] That is t_{i^+,i^-} and can be obtained following the shortest path.

[4] maxLOS depends on the minimum travel time t_{i^+,i^-}; many different functions can be used, like exponential or step functions.

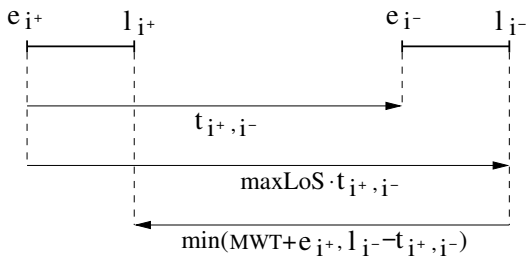

Figure 7: Time windows computation procedure for a customer specifying e_{i^+}

The user can specify the earliest departure time e_i^+ or the latest arrival time l_i^-; in first case, time windows are defined as follows [1]:

$$\begin{cases} e_{i^+} = e_{i^+} \\ l_{i^-} = e_{i^+} + \text{maxLOS} \cdot t_{i^+,i^-} \\ e_{i^-} = e_{i^+} + t_{i^+,i^-} \\ l_{i^+} = \min\left(e_{i^+} + \text{MWT}, l_{i^-} - t_{i^+,i^-}\right) \end{cases} \tag{18}$$

Figure 7 shows this procedure.

Time windows of customers specifying the latest arrival time l_{i^-} are instead defined in this way:

$$\begin{cases} l_{i^-} = l_{i^-} \\ e_{i^+} = l_{i^-} - \text{maxLOS} \cdot t_{i^+,i^-} \\ e_{i^-} = e_{i^+} + t_{i^+,i^-} \\ l_{i^+} = \min\left(e_{i^+} + \text{MWT}, l_{i^-} - t_{i^+,i^-}\right) \end{cases} \tag{19}$$

3.3 Distances and travel times computation

An abstract graph has been used to formulate the DAR problem in 3.1. Each arc (i, j) of this graph connects couples of request points and is defined by two values: the minimum distance $d_{i,j}$ and the minimum travel time $t_{i,j}$. These information come from the physical graph.

The procedure is described in Figures 8 and 9, which show physical and abstract graphs with the latter having distances $d_{i,j}$ on each arc. Nodes 1 and 9 are the first request's origin and destination points, while nodes 4 and 8 are the second request's ones.

Travel time $t_{i,j}$ can not be determined without further information regarding the network.

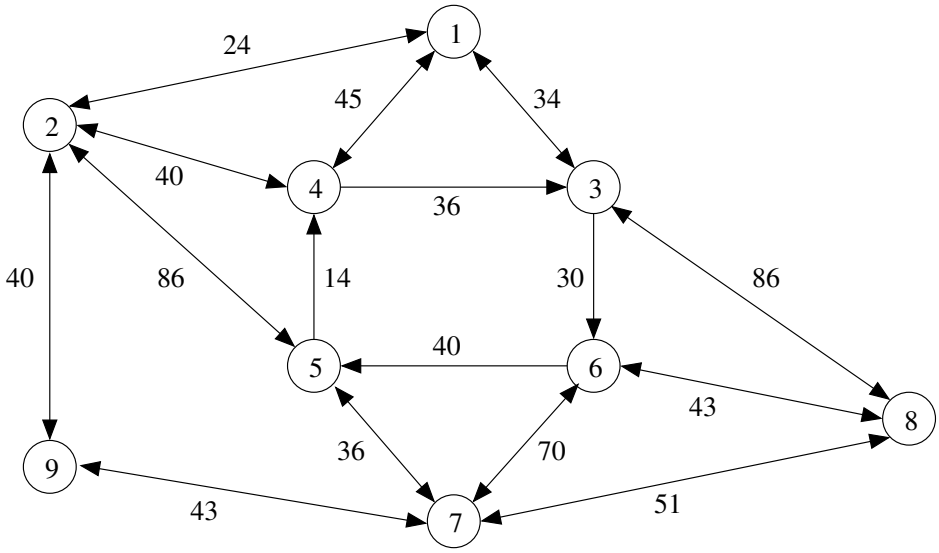

Figure 8: Physical graph

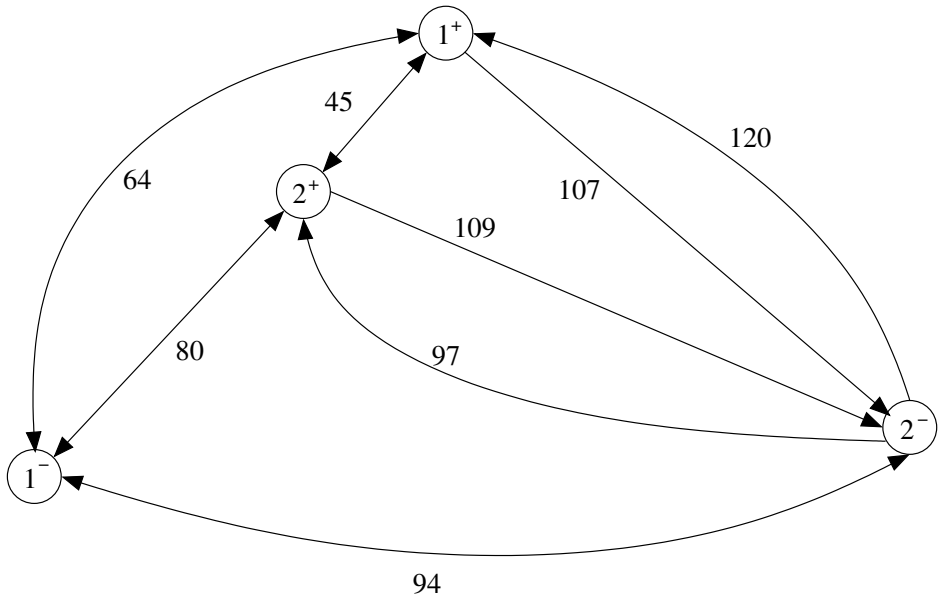

Figure 9: Abstract graph

3.3.1 Shortest paths

Several shortest path algorithms have been proposed in literature [46][47], however it's an hard task to say which is the best one as their performances depend on a lot of elements, including the features of the problem they are applied to.

There are two classes of algorithms to solve shortest path problems: *label-setting* and *label-correcting* algorithms. Both the algorithms are iterative and assign tentative distance labels to nodes at each step, which are estimates of the *upper bounds* on the shortest path distances. Label setting algorithms designate one label as permanent (optimal) at each iteration. Label correcting algorithms consider the labels as temporary until the final step when they all become permanent.

One algorithm for each class has been used in this work: the Dijkstra algorithm (*label-setting*) and the Pape-Levit version of the Bellman-Ford-Moore algorithm (*label-correcting*). As a matter of fact, Dijkstra algorithm (as well as any other label-setting one) can be very efficient when a *one-to-one* shortest path is required as the procedure can be stopped when the destination node is reached and definitively labelled. The Bellman-Ford-Moore algorithm implemented using a *dequeue* data structure (Pape-Levit version) proved to be very efficient when applied to sparse networks [46] in spite of its not so good worst-case computational complexity [47].

3.3.2 Minimum travel time paths

The speed for each arc of the graph is required in order to compute travel times over the network. The simplest solution is to consider constant speed for every arc: in this way shortest paths are also minimum travel time paths. The time required for a trip can be determined dividing the distance by the constant speed.

Unfortunately this solution does not suit to real networks consisting of many kinds of road; in fact fastest paths are not always the shortest ones. For this reason the graph's arcs should be divided into C groups (or *categories* [7]), each of them having different speed v_c, with $c \in [1, \ldots, C]$.

The algorithms described in 3.3.1 can still be used: the weight of each arc is now equal to its length divided by the speed of the category it belongs to.

3.3.3 Time-dependent network case

Time-dependent network conditions add another complication to the minimum travel time paths calculation due to the varying traffic volumes.

Many time-dependent network models have been presented in 2.4; the FIFO model described in [7] has been used in this work since other solutions does not provide reliable results in practice.

The horizon of interest is discretized into K time intervals. Each arc of the graph (or category of arc) is given a different speed for each time interval $F_k = [\underline{t}_k, \overline{t}_k]$, $k \in [1, \ldots, K]$.

The arrival time at node j starting from node i at time $\widetilde{t} \in F_k$ can be determined using the following function [7][44]:

Function ArrivalTime
Begin
  $remainingDistance = c_{ij} - v_{cF_k} * (\overline{t}_k - \widetilde{t})$
  while $remainingDistance > 0$
    $k = k + 1$
    $remainingDistance = remainingDistance - v_{cF_k} * (\overline{t}_k - i_k)$
  return $(\overline{t}_k + remainingDistance / v_{cF_k})$
End

assuming that arc (i, j) belongs to category c and its speed is v_{cF_k}.

Each node's label is equal to the earliest time that node can be reached, thus using this procedure the shortest path algorithms presented in 3.3.1 are still usable on time-dependent graphs. In this way even minimum travel times of trips taking place during more than one time interval can be correctly computed.

3.4 Handling of time window constraints

Time windows are among the most difficult constraints to handle in local research procedures, which are the core of most heuristics for vehicle routing problems. A typical approximation algorithm generates a neighborhood applying some moves (usually arc exchanges) to the current solution. Then, it substitutes the current solution with another one chosen from its neighborhood. This can be done pursuing profitability (classical local research) or using different strategies which can accept also worse solutions (for example Tabu Search, Simulated Annealing, etc.). In any case, local search algorithms visit a large number of solutions whose feasibility and value they

have to estimate. Applying a straightforward evaluation takes a time $O(n)$, which affects any algorithm requiring such a test.

Savelsbergh [12] proved that, if solutions are explored in a given order, this goal can be achieved by calculating suitable quantities and updating them each time a new solution is considered. This takes only a constant time $O(1)$, which is a great saving.

In [48] an evolution of Savelsbergh's method to handle solutions is proposed: the idea is to reduce the original graph contracting entire sequences of nodes into objects (*macronodes*) with a limited number of properties summing up all the information needed to evaluate solutions. The new graph is much like the old one, that is a time window is associated to each macronode, a travel time to each arc.

The advantages of this framework are:

- a very simple and general way to evaluate feasible neighborhood solutions;
- a lower computational effort: less operations are needed to evaluate each new solution, as a consequence of a neater mathematical formalism;
- macronodes' properties are unrelated to the solution they belong to: when it changes, most macronodes keep unchanged. Moreover, they are independent from the neighborhood structure and from how it is explored.

This method has been used in this work to efficiently evaluate solutions. However it does not hold when time-dependent network models are used: travel times between nodes may vary if sequences are changed, thus macronodes' properties computed considering the current solution are not valid to evaluate all the neighborhood solutions.

The method has been therefore further developed in this work to deal with *time-dependent* problems: given the current solution, the only macronodes computed and saved are the sequences of nodes belonging to the same time interval[5]. If the main algorithm requires information regarding a sequence which does contain nodes belonging to different time intervals, that sequence is split into known macronodes. The travel times over arcs crossing the boundary between two time intervals are computed for each solution evaluation; then they are used to combine the known macronodes and obtain the needed information.

[5]A set of criteria has been defined to decide whether a node belongs to a given time interval, according to its time window and position in the route.

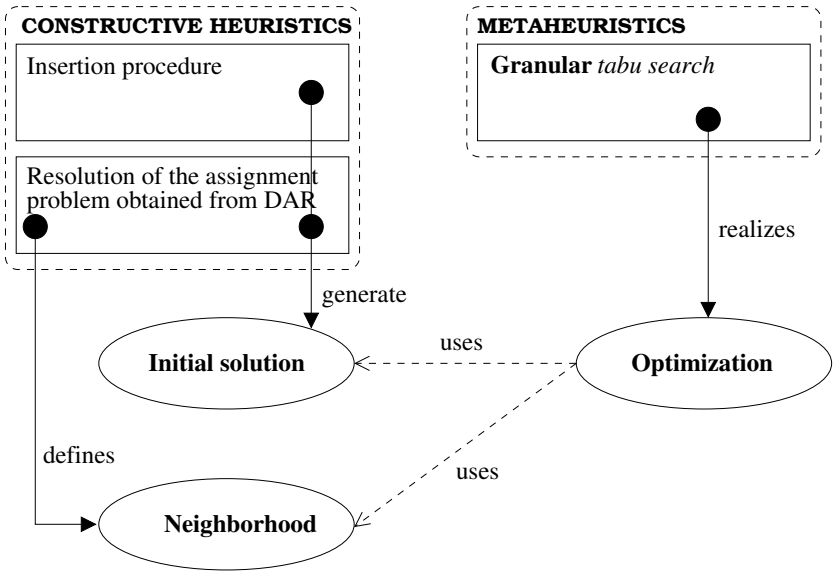

Figure 10: Basic elements and functionalities of the algorithm

4 The algorithm

4.1 Algorithm structure

The algorithm realized in this work for the DAR problem resolution is based on three elements: two constructive heuristics, used for the initial solution generation, and a metaheuristic. The heuristics are a simple insertion procedure and an algorithm for the resolution of an assignment problem obtained from a relaxation of the original DAR problem [1][18]. A *tabu search* metaheuristic was chosen for the solution optimization since it is one of the most effective algorithms used for *vehicle routing and scheduling* problems [19][38][35]; in particular a *granular* version was realized [38][49].

As shown in Figure 10, three functionalities have been obtained using these elements: the initial solution generation, the neighborhood definition for the granular tabu search and the optimization system. Both the static and dynamic version of the DAR problem can be solved combining these three functionalities in different ways.

In the *off-line* version of the algorithm only one execution of each functionality is needed as the customers demand is completely known when the process starts. On the contrary, a strategy like the *single event optimization* (described in 2.3) has been chosen for the *on-line* version; thus in this case the three functionalities are executed many times, one for each static

sub-problem.

One of this work's objectives is to realize an optimization system able to produce good results within short time so that it can be used in both *off-line* and *on-line*[6] algorithms. For this reason the *granular tabu search* has been chosen: this method is based on using a drastically restricted neighborhood obtained from the standard one by removing the moves that involve only elements which are not likely to belong to high-quality feasible solutions. In this way few moves have to be evaluated during each tabu search iteration so the research is quite fast.

4.2 The initial solution

4.2.1 Insertion procedure

A fast insertion procedure has been realized to answer customer in real time. The procedure selects the first feasible insertion position for each request: this allows to answer instantaneously to the customer, but disregards completely any optimization phase.

The insertion procedure can also be used in the *off-line* algorithm.

4.2.2 Assignment heuristic

This heuristic provides the main information required for the definition of the granular neighborhood used by *tabu search*; moreover it can be used to create an initial solution of the problem.

The method can be divided into four steps:

1. an auxiliary graph smaller and simpler than the one defined in 3.1 is built;
2. the graph is completed adding some information regarding the vehicles used;
3. an assignment problem is defined on this graph and solved;
4. the resulting unfeasible paths are made feasible for the original DAR problem.

[6]The optimization is performed during the time period between couples of request arrivals.

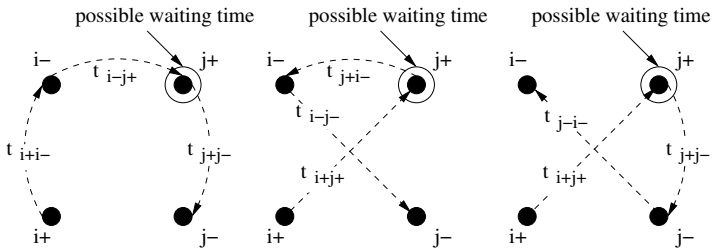

Figure 11: Possible paths to go directly from customer i to customer j.

The auxiliary graph Let $G\left(U, \bar{A}\right)$ be a directed auxiliary graph. U is the set of requests; the set of arcs is defined as $\bar{A} = \{(i,j) : i,j \in U, i \neq j\}$. Each arc $(i,j) \in \bar{A}$ has associated an ad-hoc coefficient $\bar{p}_{i,j}$. The purpose of the value $\bar{p}_{i,j}$ is to measure the spatial and temporal distance between customer i and customer j. The value $\bar{p}_{i,j}$ sums up most of the information necessary to evaluate the feasibility and the cost of any possible path to go directly from customer i to customer j, starting with node i^+: $\{i^+, i^-, j^+, j^-\}$, $\{i^+, j^+, i^-, j^-\}$ and $\{i^+, j^+, j^-, i^-\}$, as shown in Figure 11.

To compute $\bar{p}_{i,j}$ constraints (7) and (8) are used; they says that, considering two nodes i and j in N, the departure time from j is $p_j = \max\{p_i + t_{i,j}, e_j\}$. When the sequence is made by only two nodes, the equation becomes $p_j = \max\{e_i + t_{i,j}, e_j\} = e_i + t_{i,j} + \tilde{w}_{i,j}$ where $\tilde{w}_{i,j} = \max\{0, e_j - e_i - t_{i,j}\}$ is the waiting time at node j.

For a generic sequence of s nodes $\Pi = \{\pi_1, \ldots, \pi_s\}$, the departure time in node s can be determined using the following equation:

$$p_{\pi_s} = p_{\pi_1} + T_{\pi_1,\pi_s} + W_{\pi_1,\pi_s} \tag{20}$$

where T_{π_1,π_s} is the total travel time along the sequence and W_{π_1,π_s} is the total waiting time. Applying this equation to the sequence shown in Figure 11, the following values can be determined:

$$\begin{cases} p^1_{j^-} = e_{i^+} + t_{i^+,i^-} + t_{i^-,j^+} + t_{j^+,j^-} + w_{j^+} \\ p^2_{j^-} = e_{i^+} + t_{i^+,j^+} + t_{j^+,i^-} + t_{i^-,j^-} + w_{j^+} \\ p^3_{i^-} = e_{i^+} + t_{i^+,j^+} + t_{j^+,j^-} + t_{j^-,i^-} + w_{j^+} \end{cases} \tag{21}$$

Thanks to the time windows computation procedure described in 3.2, there is only one possible waiting time, i.e. w_{j^+} in node j^+.

Then, the coefficient $\bar{p}_{i,j}$ can be defined as

$$\bar{p}_{i,j} = \frac{\sum_{r=1}^{3} p^r k_r}{\sum_{r=1}^{3} k_r} \tag{22}$$

where $k_r = 1$ if $p^r \neq \infty$, $k_r = 0$ otherwise. Of course, when $\sum_{r=1}^{3} k_r = 0$ the procedure sets $\bar{p}_{i,j} = \infty$: there are no feasible ways to go from customer i to customer j. The value $\bar{p}_{j,i}$ is defined in the same way and in general $\bar{p}_{i,j} \neq \bar{p}_{j,i}$.

The vehicles lower bound To make the solution of the assignment problem effective, it is necessary to enlarge the auxiliary graph with a minimum number of vehicles required to satisfy the requests. A *lower bound* of this number is computed using the concept of *incompatibility* between request i and request j. Customers i and j are incompatible if they cannot be loaded by the same vehicle without breaking any time constraint. Capacity constraints can be ignored since usually $q_i \ll Q$ for each $i \in U$. Formally two requests are not compatible if $\bar{p}_{i,j} = \bar{p}_{j,i} = \infty$.
Then, let $G = (U, I)$ denotate the incompatibility graph: U is the set of requests, while the set of arcs is defined as $I = \{(i, j) : i, j \in U, \bar{p}_{i,j} = \infty \wedge \bar{p}_{j,i} = \infty\}$. The vehicles lower bound m is given by the dimension of the maximal clique on this graph; if the lower bound is greater than the fleet size $|M|$, m is set equal to $|M|$.

The assignment problem Given the complete graph $G(\hat{U}, \hat{A})$, where the set of nodes is $\hat{U} = \{-m \dots 0\} \cup U$, while $\hat{A} = \left\{(i, j) : i, j \in \hat{U}, i \neq j\right\}$ is the set of arcs and each arc (i, j) has associated a cost function $\bar{p}_{i,j}$, it is possible to show that the assignment problem is a useful relaxation of the DAR problem. Indeed, the constraints involving time windows, namely constraints (7) and (8), can be relaxed, since they are used to calculate $\bar{p}_{i,j}$. Constraints (3) and (9) are useless in this graph, since it does not consider the origin and destination points of requests. Constraints (4), (5) and (6), ensuring the feasibility of loads, are discarded without taking care of them and the feasibility of the solution obtained is verified in a second step. In changing the constraints (1) and (2), the problem to solve becomes

$$\min \sum_{(i,j)\in\hat{A}} \bar{p}_{i,j} x_{i,j} \tag{23}$$

$$\text{s.t.} \sum_{j\in\hat{U}} x_{i,j} = 1 \qquad \forall i \in \hat{U} \tag{24a}$$

$$\sum_{j\in\hat{U}} x_{j,i} = 1 \qquad \forall i \in \hat{U} \tag{24b}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall\, (i, j) \in \hat{A} \tag{24c}$$

Usually the assignment problem is not an interesting relaxation of *routing* problems. The obtained solution consists of m sequences of customers containing the depot and other cycles composed only by requests. These cycles, called *subtours*, are not feasible for both *routing* and *routing and scheduling* problems. However in this case solutions contain few *subtours* thanks to the *subtour elimination constraints* (4) and (7) and the definition of equation (22), which considers these constraints.

Make feasible an unfeasible path In order to obtain a solution for the DAR problem, it is necessary to deal with the subtours and to make each one of the m paths (each of them connecting customers close together both temporally and spatially) a feasible route, i.e a sequence of pick-up and delivery nodes. The procedure adopted does not take care of subtours. Thus, it is sufficient to transform the solution of the assignment problem for each vehicle in a feasible route. Two procedures have been realized:

- *Trivial*: the requests are processed one by one in the same order as they are in the sequence; the procedure puts the couple of origin and destination nodes of each request immediately after the last node inserted into the route. Then it checks if any constraint is break and in that case the last inserted request is discarded.

- *Smart*: this procedure starts with a route equal to the sequence of pickup and delivery nodes produced by the assignment resolution procedure. The route cost is evaluated considering the total travel time and adding infinite each time a constraint is break. Then the procedure tries all the possible insertion positions of every destination node: if there isn't any feasible insertion position the whole request is discarded, instead if one or more are found the one leading to the minimum cost solution is selected.

After this process the initial solution generation is completed using the insertion procedure described in 4.2.1 to deal with the *subtours* and the requests discarded by one of the procedures just described. The whole fleet can be exploited during this process.

4.3 Granular Tabu Search

As said in 4.1, one of this work's main objectives was to choose a metaheuristic able to produce good results within short time. The details of the used technique are described as follows.

4.3.1 The solution space

The neighborhood $N(s)$ of the current solution s consists of all the solutions that can be obtained applying a single move to s. This space includes only feasible solutions; the possible moves are:

- removal of a request from a route and insertion into another route, choosing the best insertion position of both origin and destination points;
- insertion of a yet unserviced request into a route, choosing the best insertion position of both origin and destination points;
- removal of a request form a route; that request becomes unserviced.

More complex transformations can be achieved through sequences of the described simple moves.

During each iteration, the tabu search procedure selects the best solution contained in $N(s)$; both improving and worsening[7] solutions are allowed.

4.4 The granular neighborhood structure

Considering the moves just defined, the neighborhood can be described using a directed graph $G(U, \tilde{A})$: U is the set of requests, while $\tilde{A}$ is the set of arcs connecting two requests serviced by different vehicles or a serviced request and an unserviced request. Figure 12 shows an example of this graph, with requests 1, 2 and 3 serviced by a vehicle and requests 4, 5 and 6 serviced by another one.

Each arc of the graph defines two moves: the first one requires the arc tail node request to be moved, the second one the arc head node request. Consider for example the arc $(4, 2)$ and suppose that requests 4 and 2 are serviced by different vehicles: Figure 13 shows how the tail node request 4 is moved to the route request 2 belongs to. The insertion positions for node 4^+ depend on the arc direction: 4^+ can be inserted only before 2^+. Figure 14 instead shows how the head node request 2 is moved to the route of request 4. In this case the arc direction states that 2^+ has to be inserted after 4^+. Figures 15 and 16 show two examples of the remaining moves, when request 2 is unserviced.

[7] In comparison with the objective function values of the current solution and of the best solution found so far.

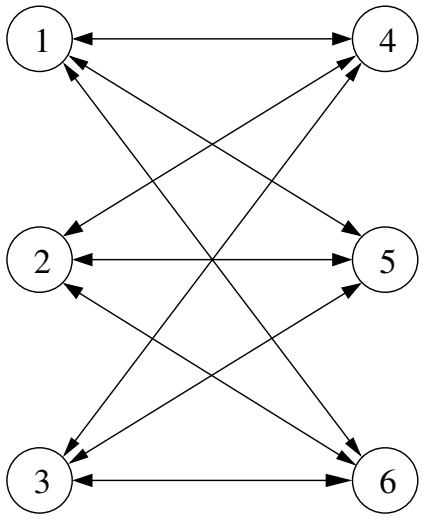

Figure 12: An example of graph $G(U, \tilde{A})$

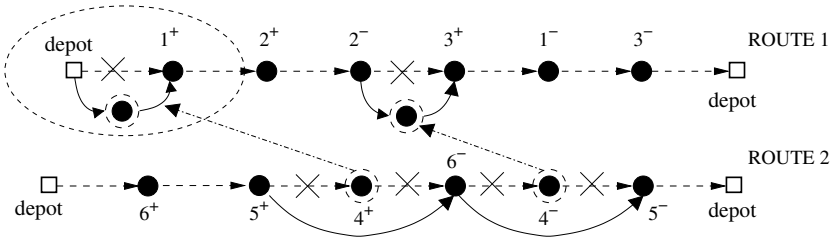

Figure 13: The tail node request 4 of arc $(4, 2)$ is moved to the route of the head node request 2

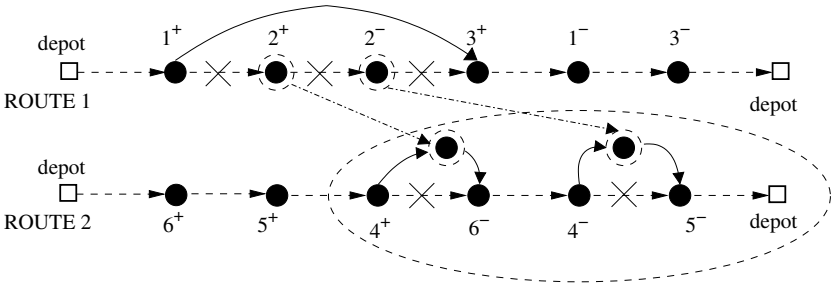

Figure 14: The head node request 2 of arc $(4, 2)$ is moved to the route of the tail node request 4

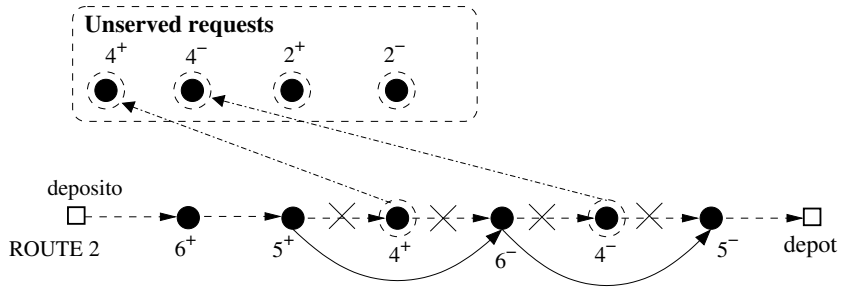

Figure 15: The tail node request 4 of arc $(4, 2)$ becomes unserviced

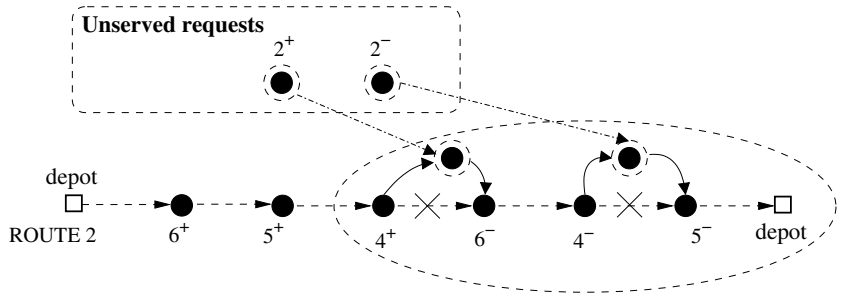

Figure 16: The head node request 2 of arc $(4, 2)$ is moved to the route of the tail node request 4 and thus becomes serviced

Each iteration of the tabu search optimization process requires the evaluation of the moves deriving from all the arcs of the neighborhood graph $G(U, \tilde{A})$. The *granular tabu search* method is based on using a drastically restricted neighborhood: the information required to obtain a sparse neighborhood graph removing some moves comes from the assignment heuristic described in 4.2.2.

4.4.1 Reduced costs

The neighborhood graph $G(U, \tilde{A})$ has been simplified considering the arc reduced costs.

Let $a = n + m$ be the size of the assignment problem, where n is the number of requests and m is the vehicles *lower bound*. Given the solution of the assignment problem, the reduced cost of each arc (i, j) is equal to $\bar{c}_{i,j} = \bar{p}_{i,j} - u_i - v_j$ where u_i and v_j are the dual variables of the assignment dual problem. The solution basis[8] consists of $2a - 1$ variables since the number of constraints of the assignment problem (shown in 4.2.2) is equal to $2a$ and it

[8]A variable belongs to the basis if its reduced cost is equal to 0.

can be proved that they are not all linear independent [50].

This means that three kinds of arc can be determined:

- a arcs with reduced cost equal to 0 and whose variables define the solution ($x_{i,j} = 1$);
- $a - 1$ arcs with reduced cost equal to 0 and whose variables do not belong to the solution ($x_{i,j} = 0$);
- the remaining arcs with $\bar{c}_{i,j} \geq 0$; their reduced costs depend on the dual variables u_i and v_j and vary between 0 and almost infinite.

The second group arcs have a good cost $\bar{p}_{i,j}$ but they do not belong to the assignment problem solution since their introduction would require the insertion of positive cost arcs. However we are not interested only in the assignment problem resolution. In fact arcs with low reduced cost values connect requests that are close together both temporally and spatially, since $\bar{c}_{i,j}$ depends on $\bar{p}_{i,j}$; thus it would be a good idea to serve these requests with the same vehicle in the DAR problem.

For this reason, the weight of each arc (i, j) of graph $G(U, \tilde{A})$ is equal to its reduced cost $\bar{c}_{i,j}$; then all the arcs having weight greater than a *granularity threshold* are discarded. The threshold starting value is 0, but it can be modified in order to obtain the diversification and intensification strategies described in 4.6.

This technique reduces the neighborhood size and thus speeds up the research process since less moves need to be evaluated during each iteration. Results reported in Section 5 show that the neighborhood reduction process does not affect in a significant way the solutions quality.

4.5 Tabu list and aspiration criteria

The *tabu search* heuristic uses a short term memory [51][50] to avoid cycling and the stopping of the research process in local minimums. At each iteration the move opposite to the one being executed is declared *tabu*. That move will be forbidden for a given number of iterations (*tabu tenure*) [34] so that the executed solution transformation can't be immediately undone.

In this work moves have been saved using the arcs they involve: in Figure 17 the arcs being saved after the removal of request 2 from its route are marked with bold arrows. The arcs directly connecting nodes to depot or the origin and destination points of the same request are not considered as they do not indicate the move in an univocal way.

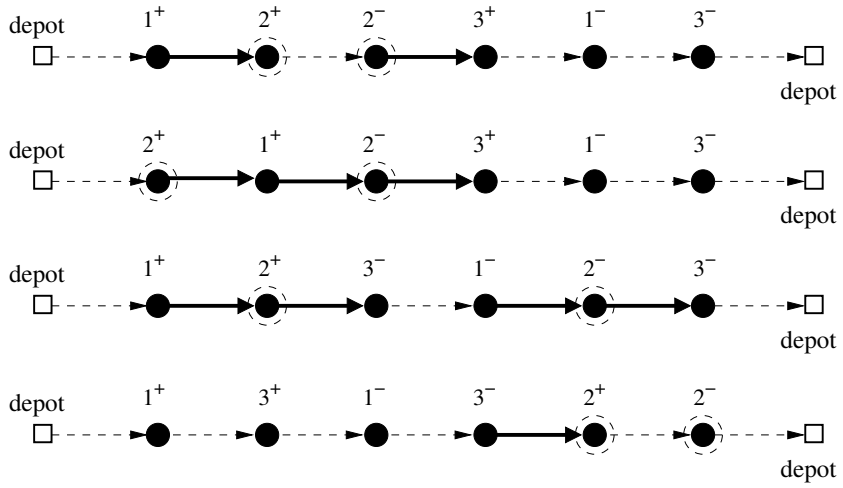

Figure 17: Arcs saved into the *tabu list*

The moves involving only *tabu* arcs at the current iteration can't be executed except if they lead to a solution whose objective function value is better than the best one found so far: this is called *aspiration criterion*.

4.6 Diversification and intensification

Intensification and *diversification* techniques are used to improve the effectiveness of the TS method [34]. *Intensification* tries to focus the research on promising portions of the solution space, while *diversification* moves it to other ones trying to make up for the local research drawbacks. Three techniques have been used in this work:

Tabu tenure dynamic variation The *tabu tenure* value can be constant or can vary according to several strategies [34][49]. We based this variation on the objective function evolution. If improving moves have been executed for a consecutive pre-defined number of iterations, the research process is probably analyzing an interesting portion of the solution space, thus intensification is required and the *tabu tenure* value is reduced. On the contrary, if the objective function value has not been improved for some iterations, may be the research process has reached a local minimum, thus diversification is required and the *tabu tenure* value is increased.

Frequency-based penalization This technique uses a long term memory to save the frequency of addition of arcs to the current solution. For example

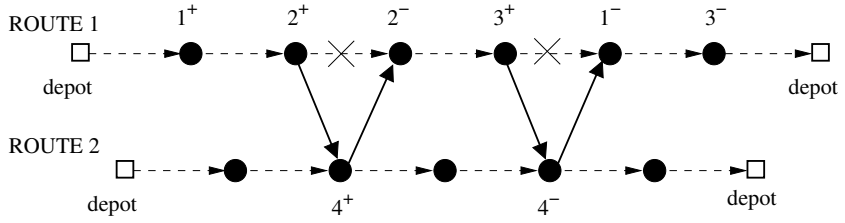

Figure 18: Arcs to be saved when a move is executed.

consider this move: the request 4 is taken out of route 2 and inserted into route 1; Figure 18 shows which arcs are saved, e.g. $(2^+, 4^+)$, $(4^+, 2^-)$, $(3^+, 4^-)$ e $(4^-, 1^-)$.

Let s be the current solution; each solution $\bar{s} \in N(s)$ such that $f(\bar{s}) > f(s)$ is penalized by a factor $p(s) = \lambda\rho\sqrt{nm}f(\bar{s})$, where ρ is the mean value of the number of times each considered arc has been added to the current solution, λ is a parameter used to control the intensity of the diversification and $\sqrt{nm}$ is a scaling factor required to adjust the penalties with respect to the problem size[9]. This strategy has been proposed by Taillard [52] and successfully used in many other *tabu search* applied to the *vehicle routing*, for example in [37].

Granularity threshold variation The granularization process naturally produces a research intensification since few moves are evaluated during each iteration. However three diversification strategies can also be defined varying the granularity threshold and thus dynamically changing the structure of the sparse graph associated to the granular neighborhood.

- At each iteration, if the algorithm can not find any feasible solution, the granular graph is enlarged considering arcs with a positive, but still small, reduced cost (the granularity threshold is increased). Whenever the algorithm improves the best solution found so far, the sparse graph becomes small again.

- If the algorithm is unable to improve the best solution found so far for a fixed number of iterations, the granular graph is enlarged considering arcs with a positive, but still small, reduced cost (the granularity threshold is increased). Whenever the algorithm improves the current solution, the sparse graph becomes small again.

- If the algorithm is unable to improve the best solution found so far for a fixed number of iterations, the granular graph is enlarged consider-

[9] n is the number of requests to be served and m is the fleet size.

ing arcs with a positive, but still small, reduced cost (the granularity threshold is increased). Whenever the algorithm improves the best solution found so far, the sparse graph becomes small again.

These techniques allowed us to define three versions of the algorithm: GTSF uses only the first technique, GTSVC exploits the first one and the second one, finally GTSVO uses the first one and the last one. These algorithm's versions produce different quality solutions within different times, as shown in Section 5.

4.7 Stopping criteria

Two stopping criteria have been defined for the *off-line* algorithm: the optimization process is stopped after a fixed number of iterations or after a fixed number of iterations without any improvement of the best solution's objective function.

The stopping criterion for the *on-line* algorithm is obtained as the minimal value between a fixed CPU time and the distance among two consecutive request arrivals.

4.8 Conclusions

As said in 4.1, the described optimization system can be applied to both static and dynamic DAR problems. In the first case, the whole structure of the algorithm is the one shown in Figure 19.

The structure of the *on-line* algorithm is shown in Figure 20: the optimization process has to be stopped and then started again every time a new request arrival occurs. Moreover some devices have been used to deal with the real-time context:

- During each iteration the optimization process considers only the unblocked requests. A request is *blocked* if its earliest pick-up time is prior to the current time plus a time interval used as margin of safety and necessary to transmit paths to the vehicle drivers.
- The optimization process runs only if a fixed number of unblocked requests is reached; as a matter of fact during the earliest and latest hours of the service time there are few unblocked requests to be processed, so no improvements to the current solution are usually possible.

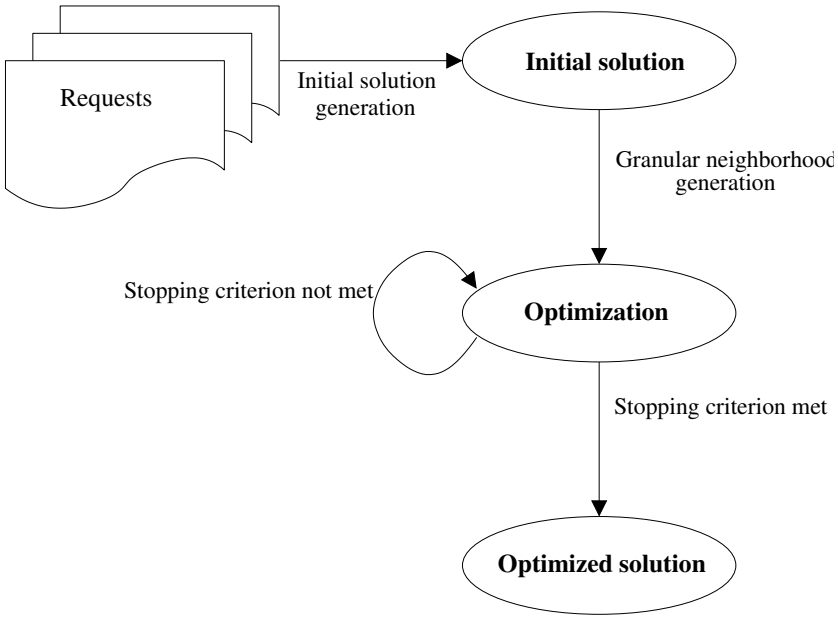

Figure 19: Structure of the *off-line* algorithm

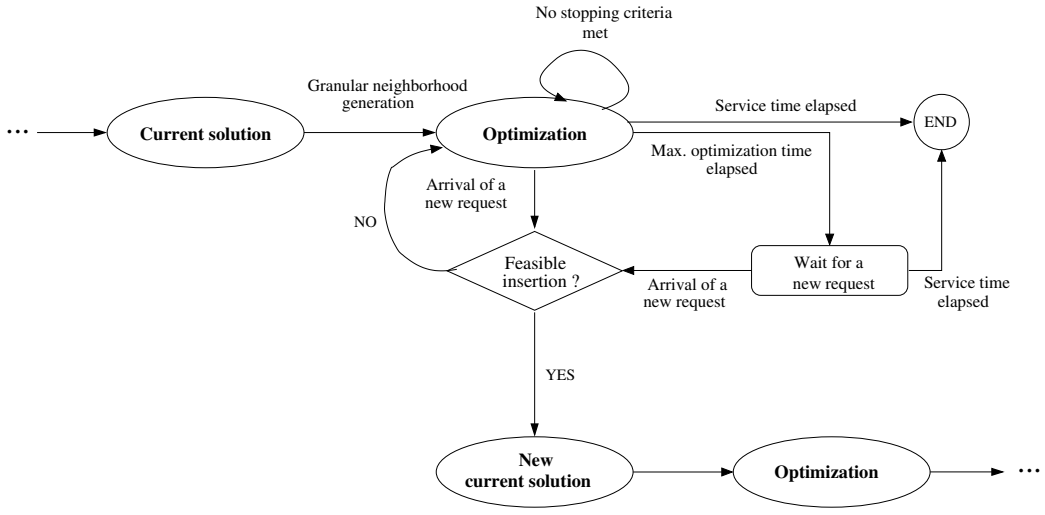

Figure 20: Structure of the *on-line* algorithm

- Every solution must serve all the customers in the *on-line* case; this means that the only admitted transformations between solutions are those that move a request from a route to another one.

Both *off-line* and *on-line* algorithms can also be used in time-dependent conditions making use of the minimum travel time paths algorithms described in 3.3.3.

5 Computational results

This section explains how the described algorithm has been tested and analyzes the obtained results.

Both *off-line* and *on-line* algorithms have been tested, considering *non time-dependent* and *time-dependent* networks. Moreover the use of different diversification strategies based on the granularity threshold variation allowed us to test 3 versions of the algorithm (GTSF, GTSVC and GTSVO, see 4.6 for further details) and to compare them with a TS algorithm.

5.1 Data generation

To the authors knowledge, there are no benchmark instances for the DAR problem. The data set has been created using the complete network of Milan (about 1700 arcs and 7000 nodes); the set of random instances has been generated extracting casually a pair of nodes for each customer. Each customer requires either the arrival time or the departure time and that value is generated randomly in a time interval of 10 hours. Time windows are constructed as described in 3.2, considering the maxLOS function shown in Figure 21 and given by:

$$\text{maxLOS}(t) = \begin{cases} \frac{\text{DV}}{\text{VC}} + (1+\alpha) & \text{if } t < \text{VC} \\ \frac{\text{DV}}{t} + (1+\alpha) & \text{if } \text{VC} \leq t < \text{VL} \\ \frac{\text{DV}}{\text{VL}} + (1+\alpha) & \text{if } t \geq \text{VL} \end{cases} \tag{25}$$

where t is the direct travel time from the pick-up to the delivery point.

Two minimum levels of service have been considered: a low-quality level of service, obtained using the parameters shown in Table 1, and a high-quality level of service, obtained using the parameters shown in Table 2. A MWT equal to 1000 seconds has been used for every test.

Almost all the tests has been performed using instances consisting of 100, 250 and 500 requests. A fleet size suitable to each size of instances has been

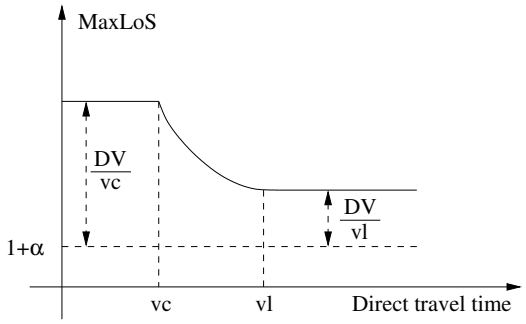

Figure 21: MaxLoS function

Parameter	*Value*
DV	1000
alpha	0,5
VC (s)	600
VL (s)	2400

Table 1: MaxLOS parameters for a low-quality level of service.

Parameter	*Value*
DV	900
alpha	0,0
VC (s)	600
VL (s)	2400

Table 2: MaxLOS parameters for a high-quality level of service.

Start time	*End time*	*Speed (km/h)*
7.00	10.00	20
10.00	12.00	25
12.00	17.00	28

Table 3: Time intervals

Procedure	*Requests*	*Unserved req.*	*Average* LOS	*Vehicles*	*Time (s)*
TRIVIAL	100	5,0%	0,31671	6,0	0,097
SMART	100	3,4%	0,36769	6,0	0,100
INSERTION	100	5,0%	0,43142	6,0	0,147
TRIVIAL	250	6,6%	0,43450	12,0	0,825
SMART	250	4,0%	0,45249	12,0	0,922
INSERTION	250	6,5%	0,48148	12,0	0,897
TRIVIAL	500	13,5%	0,49094	20,0	4,628
SMART	500	11,9%	0,49976	20,0	4,741
INSERTION	500	13,4%	0,50547	20,0	4,890

Table 4: Initial solutions analysis: results with a non time-dependent network.

determined through some preliminary tests: the algorithm was allowed to use as many vehicles as it needed to serve all the customers. In this way the required fleet size has been estimated and we decided to use 6, 12 and 20 vehicles to serve 100, 250 and 500 requests respectively.

The speed over the *non time-dependent* network used in all the tests is 25 km/h. *Time-dependent* network, instead, has been described considering 3 time intervals whose data are shown in Table 3.

The machine used to solve the problems is an AMD ATHLON XP 2100+ with 512MB of RAM.

5.2 Initial solutions analysis

The aim of the first test phase was to decide which of the procedures described in 4.2 gives the best initial solution in terms of quality (number of served requests and offered level of service) and required computing time.

Tables 4 and 5 present the obtained results in *non time-dependent* and *time-dependent* conditions respectively. *Smart* (*Trivial*) refers to the assignment heuristic followed by the *smart* (*trivial*) procedure to obtain feasible paths.

The *smart*-assignment procedure proves to be the best technique since it always allows the algorithm to serve more customers than the other techniques

Procedure	*Requests*	*Unserved req.*	*Average* LOS	*Vehicles*	*Time (s)*
TRIVIAL	100	3,8%	0,31708	6,0	4,609
SMART	100	2,6%	0,38346	6,0	4,581
INSERTION	100	5,0%	0,42595	6,0	5,922
TRIVIAL	250	6,7%	0,42050	12,0	25,503
SMART	250	4,9%	0,44249	12,0	25,753
INSERTION	250	7,4%	0,48108	12,0	32,372
TRIVIAL	500	14,2%	0,46808	20,0	123,759
SMART	500	11,0%	0,48898	20,0	116,840
INSERTION	500	13,8%	0,51562	20,0	128,000

Table 5: Initial solutions analysis: results with a time-dependent network.

do. Moreover the computing time required by the assignment procedure is less than the one the insertion procedure needs, especially in the *time-dependent* case. As expected, the insertion procedure produces solutions with a bad (high) average LOS value, since no optimization is performed and each request is inserted into a route at the first feasible position.

Finally time-dependent problems require more time to be solved than non time-dependent ones due to the high complexity of the shortest path algorithms frequently used in time-dependent conditions.

5.3 Optimization algorithms analysis

The aim of the second test phase was to evaluate the performance and quality of results produced by the different versions of the optimization algorithm (GTSF, GTSVC and GTSVO).

Table 6 shows the results obtained after 2500 tabu search iterations with a non time-dependent network, while Table 7 shows the results obtained after 1000 iterations in time-dependent conditions.

The optimization process considerably improved the solution quality. Every optimization algorithm succeeded in serving much more requests than the initial solutions do, while providing a better level of service to the customers. The number of vehicles did not change as there was always at least one unserviced request so the whole fleet was always required.

TS and GTSVO produce better solutions but are very slow. GTSF and GTSVC, instead, generate very good solutions (whose quality is almost equal to that of the solution produced by more complex algorithms) within modest computing times. In particular the GTSVC version of the algorithm is the best compromise between efficiency and effectiveness.

Algorithm	*Requests*	*Unserved req.*	*Average* LOS	*Vehicles*	*Time (s)*
INITIAL SOL.	100	3,4%	0,36769	6	0,097
GTSF	100	0,4%	0,19034	6	40,616
GTSVC	100	0,2%	0,16928	6	63,956
GTSVO	100	0,2%	0,16875	6	652,762
TS	100	0,2%	0,16862	6	1092,071
INITIAL SOL.	250	4,0%	0,45250	12	0,922
GTSF	250	1,0%	0,23460	12	107,678
GTSVC	250	0,4%	0,19336	12	218,585
GTSVO	250	0,3%	0,16494	12	2648,395
TS	250	0,4%	0,18535	12	6469,919

Table 6: Optimization algorithms analysis: results with a non time-dependent network.

Algorithm	*Requests*	*Unserved req.*	*Average* LOS	*Vehicles*	*Time (s)*
INITIAL SOL.	100	2,6%	0,38346	6,0	4,328
GTSF	100	0,6%	0,18165	6,0	197,341
GTSVC	100	0,4%	0,17711	6,0	1209,111
GTSVO	100	0,4%	0,16084	6,0	10986,608
INITIAL SOL.	250	4,9%	0,44249	12,0	22,450
GTSF	250	1,6%	0,24035	12,0	366,703
GTSVC	250	1,5%	0,20855	12,0	4798,447
GTSVO	250	1,6%	0,18429	12,0	22522,980

Table 7: Optimization algorithms analysis: results with a time-dependent network.

Algorithm	*Requests*	*Unserved req.*	*Average* LOS	*Vehicles*
ON-LINE	99,6	15,3%	0,34567	6
OFF-LINE (GTSVC)	99,6	10,7%	0,29464	6
ON-LINE	236,0	19,9%	0,34814	12
OFF-LINE (GTSVC)	236,0	18,7%	0,30638	12
ON-LINE	479,2	23,8%	0,40306	20
OFF-LINE (GTSVC)	479,2	18,1%	0,33596	20

Table 8: Comparison between results obtained by the *on-line* and *off-line* algorithms.

As said in 5.2, time-dependent problems are more complex and require more time to be solved.

5.4 On-line algorithm analysis

The aim of the last test phase was to evaluate the performance of the *on-line* system. Real-time context requires fast optimization procedures, so the optimization algorithm used was the GTSVC since it proved to be both effective and efficient.

The set of instances has been generated through a Poisson process[53] considering three arrival frequencies:

- 1 request every 5,4 minutes (100 requests during the whole service time);
- 1 request every 2,2 minutes (250 requests during the whole service time);
- 1 request every 1,1 minutes (500 requests during the whole service time).

The *on-line* algorithm has been evaluated comparing its results with the ones produced by the *off-line* version running on the same data set. Table 8 shows the collected results: the *off-line* algorithm generates better solutions. This can be explained considering that the *on-line* optimization process is stopped every time a new request arrival occurs; moreover the customers demand is not completely known during the process and some request are blocked, thus the *on-line* problem is much more difficult to solve than the *off-line* one. Nevertheless the difference between the two algorithms' results in terms of unserved requests is always less than 6%.

Finally we evaluated the system response time that is the time required to tell each customer if his request has been accepted or rejected. Considering *non time-dependent* conditions and the maximum arrival frequency described above, the average response time was less than 5 seconds, thus the system can be used in a real-time context.

5.5 Conclusions

In this work some algorithms to solve both the *off-line* and *on-line* versions of the *Dial a Ride* have been proposed. The use of a *granular tabu search* metaheuristic applied to the DAR problem for the first time allowed us to design an algorithm both effective and efficient. The computational results proved the algorithm good performances and that it will be possible to use the algorithm for practical applications in a real context.

References

[1] R. Wolfler Calvo. *Un sistema di supporto alle decisioni per il problema del Dial-a-Ride*. PhD thesis, Politecnico di Milano, 1994-1996.

[2] M. W. P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation research*, 29:17–29, 1995.

[3] G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 151:1–11, 2003.

[4] M. Desrochers, J. K. Lenstra, and M. W. P. Savelsbergh. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational research*, 46:322–332, 1990.

[5] H. N. Psaraftis. Dynamic vehicle routing: Status and prospects. *Annals of Operational Research*, 61:143–164, 1995.

[6] L. Bianchi. Notes on dynamic vehicle routing - The State of the Art -. Technical report, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), 2000.

[7] S. Ichoua, M. Gendreau, and J. Y. Potvin. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144:379–396, 2003.

[8] C. Malandraki and M. S. Daskin. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science*, 26(3):185–200, Agosto 1992.

[9] A. V. Hill and W. C. Benton. Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. *Journal of the Operational Research Society*, 43(4):343–351, 1992.

[10] H. N. Psaraftis. An exact algorithm for the single-vehicle many-to-many dial a ride problem with time windows. *Transportation Science*, 17:351–357, 1983.

[11] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22, 1991.

[12] M. W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985/6.

[13] G. Laporte, M. Gendreau, J. Y. Potvin, and F. Semet. Classical and modern heuristic for the vehicle routing problem. Technical report, 1999.

[14] O. Bräysy and M. Gendreau. Route construction and local search algorithm for the vehicle routing problem with time windows. Technical report, SINTEF Applied Mathematics, Research Council of Norway, 2001.

[15] S. Mitrović-Minić. Pickup and delivery problem with time windows: A survey. Technical report, Simon Fraser University SFU, 1998.

[16] M. M. Solomon. Algorithms for the vehicle routing problems with time windows constraints. *Operations Research*, 35:254–265, 1987.

[17] J. J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. M. Wilson. A heuristic algorithm for the multi-vehicle many-to-many advance request dial-a-ride problem. Technical Report MIT-UMT-82-3, M.I.T Cambridge, Massachussetts 02139, 1982.

[18] R. Wolfler Calvo and A. Colorni. An approximation algorithm for the dial-a-ride problem. Technical report, 2003.

[19] M. Gendreau, G. Laporte, and J. Y. Potvin. Metaheuristics for the vehicle routing problem. Technical report, Canada Research Chair in Distribution Management and GERAD École des Hautes Études Commerciales, Agosto 1999.

[20] G. Laporte I. H. Osman. Metaheuristics: A bibliography. *Annals of Operation Research*, 63:513–623, 1996.

[21] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceding of* ECAL91 - *European Conference on Artificial Life, Paris, France*, pages 134–142. Elsevier Publishing, 1992.

[22] M. Dorigo, V. Maniezzo, and A.Colorni. The ant system: Optimization by a colony of cooperating agents. IEEE *Transactions on Systems, Men, and Cybernetics – Part B*, 26(1):1–13, 1996.

[23] H. Kawamura, M. Yamamoto, T. Mitamura, K. Suzuki, and Ohuchi A. Cooperativa search on pheromone communication for vehicle routing problems. IEEE *Transactions on Fundamentales*, E81-A:1089–1096, 1998.

[24] B. Bullnheimer, R. F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 109–120, 1998.

[25] B. Bullnheimer, R. F. Hartl, and C. Strauss. An improved ant system for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999.

[26] L. M. Gambardella, É. Taillard, and G. Agazzi. MACS-VRPTW: *A Multiple Ant Colony System for vehicle routing problems with time windows.* Technical report, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), 1999.

[27] J. Y. Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63:339–370, 1996.

[28] D. E. Goldberg and R. Lingle. Alleles, loci and the traveling salesman problem. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159, 1985.

[29] J. Y. Potvin and S. Bengio. The vehicle routing problem with time windows part ii: Genetic search. INFORMS *Journal on Computing*, 8(2):165–172, 1995.

[30] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms. *Annals of Operation Research*, 41:421–451, 1993.

[31] W. C. Chiang and R. A. Russell. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63:3–27, 1996.

[32] F. Glover. Tabu search. *Part I* - ORSA *Journal on Computing 1*, pages 190–206, 1990.

[33] F. Glover. Tabu search. *Part II* - ORSA *Journal on Computing 2*, pages 4–32, 1990.

[34] M. Gendreau. An introduction to tabu search. Technical report, Centre de recherche sur les transport and Dpartement et de recherche oprationnelle, Universit de Montral, Luglio 2002.

[35] J. F. Cordeau and G. Laporte. Tabu search heuristic for the vehicle routing problem. Technical Report G-2002-15, Canada Research Chair in Distribution Management and GERAD École des Hautes Études Commerciales, Marzo 2002.

[36] Y. Rochat and É. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.

[37] J. F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B*, 37:579–594, 2003.

[38] P. Toth and D. Vigo. The granular tabu search (and its application to the vehicle routing problem). Technical report, Dipartimento di Elettronica, Informatica e Sistemistica, Universit di Bologna, 1998.

[39] M. Gendreau and J. Y. Potvin. Dynamic vehicle routing and dispatching.

[40] F. Maffioli. *Elementi di programmazione matematica*, volume 2. Masson, 1990.

[41] M. Gendreau, F. Guertin. J. Y. Potvin, and R. Sguin. Neighborhood search heuristics for a dnamic vehicle dispatching problem with pick-ups and deliveries. Technical Report CRT-98-10, Centre de Recherche sur les Transports, Universit de Montral, 1998.

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

[43] D. Bertsimas and G. J. van Ryzin. A stochastic and dynamic vehicle routing problem in the euclidean plane. *Operations Research*, 39:601–615, 1991.

[44] K. Sung, M. G. H. Bell, M. Seong, and S. Park. Shortest path in a network with time-dependent flow speeds. *European Journal of Operational Research*, 121:32–29, 2000.

[45] C. E. Miller, A. W. Tucker, and A. Zemlin. Integer programming formulations and traveling salesman problem. *Journal of the* ACM, 7(4):326–329, Ottobre 1960.

[46] F. B. Zhan and C. E. Noon. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32(1), Febbraio 1998.

[47] D. O'Connor. Lecture 8: Shortest path spanning trees. Corso "'Algorithms and Data Structures"' presso University College Dublin, 2000.

[48] R. Cordone and R. Wolfler Calvo. Note on time window constraints in routing problems. Technical report, Politecnico di Milano - Dipartimento di Elettronica e Informazione, Maggio 2000.

[49] A. Cenni. Un algorimto tabu search granulare per il vehicle routing problem. Master's thesis, Universit degli studi di Bologna, Facolt di Ingegneria, Corso di Laurea in Ingegneria Informatica, 1996–97.

[50] A. Colorni. *Ricerca Operativa*. Citt Studi, 1984.

[51] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publisher, Norwell, MA, 1997.

[52] É. D. Taillard. Parallel iterative search methods for vehicle routing problem. *Networks*, 23:661–673, 1993.

[53] A. M. Mood, F. A. Graybill, and D. C. Boes. *Introduzione alla statistica*. McGraw-Hill, 1988.