Constraint Propagation for the Dial-a-Ride Problem with Split Loads

Samuel Deleplanque, Alain Quilliot

Constraint Propagation for the Dial-a-Ride Problem with Split Loads

Samuel Deleplanque[1], Alain Quilliot[1]

LIMOS, UMR CNRS 6158, Bt. ISIMA, BLAISE PASCAL university, France
{deleplan, quilliot}@isima.fr

Abstract. This paper deals with a new problem: the Dial and Ride Problem with Split Loads (DARPSL), while using randomized greedy insertion techniques together with constraint propagation techniques. Though it focuses here on the static versions of Dial and Ride, it takes into account the fact that practical DARP has to be handled according to a dynamical point of view, and even, in some case, in real time contexts. So, the kind of algorithmic solution which is proposed here, aim at making easier to bridge both points of view. First, we propose the general framework of the model and discuss the link with dynamical DARP, second, we describe the two algorithms (DARP and DARPSL), and lastly, show numerical experiments for both.

1 Introduction and Literature Review

Literature in the field of urban systems and geomatics hint a trend to a surge of new on demand flexible transportation systems (ODT): ad hoc shuttle fleets, vehicle sharing (AUTOLIB...), co-transportation (see for instance [17], [4]). This trend reflects from both environmental (climate change, overcrowded megalopolis) and economical concerns (surge of energy prices). It has also to be associated with technological advances: internet, mobile communication, geo-localization, which allow efficient monitoring of complex mobility system and large sets of heterogeneous requests.

An important Operations Research model for the management of flexible reactive transportation system is the DARP, which tries to optimize the way a given fleet of vehicles meet mobility demands emanating from people, or, in some cases from some combination of people and goods. DARP is a complex problem, which admits several formulation, most of them NP-Hard. It usually does not fit well the Integer Linear Programming framework [16] and one must try do handle it through heuristic techniques: Tabu search [7], genetic algorithms [18], partial branch/bound [16], Simulated Annealing [9], VNS techniques [20], [13], Dynamic Programming [16]-[17], Insertion techniques [6]-[12].

A basic features of DARP is that it usually derives from a dynamic context. So, algorithms for static DARP should be designed in order to take into account the fact that they will have to be adapted to dynamic and reactive context, which means synchronization mechanisms, interactions between the users and the vehicles, and uncertainty about fore coming demands.

In this paper, we use a generic DARP model with time windows to define the new DARP with split loads, and we propose algorithms for this model based upon randomized insertion techniques and constraint propagation. These algorithms can be easily adapted to dynamic contexts, where demand packages has to be inserted into (or eventually removed from) current vehicle schedules. This has to be done in a very short time while taking into account some probabilistic knowledge about fore coming demand packages.

The focus of the paper is to use constraint propagation on the time constraints and to allow split loads in the DARP. Little has been published on this problem, only [15]-[19]. The closest study is the Pickup and Delivery Problem with Splits Loads (PDPSL). The PDPSL solution of solved instances can give a rate up to 25% fewer vehicles used compared to the classic PDP [10]. The Tabu search is also used to solve the PDFSL [2]. For a recent review of this problem, refer to [3].

The paper is organized as follows: we first introduce the problem and discuss the link between static and dynamic formulations. Next, describe our formal DARP with split loads model, together with the performance criteria which we used. Then, we present the general insertion mechanism together with the constraint propagation techniques which we use in order to filter insertion parameters and to select the demands to be inserted. We conclude with experiments and comparison of the two resolutions.

2 The Standard Dial a Ride Problem

2.1 General Dial a Ride Problem

A Dial a Ride Problem instance is essentially defined by:

- a transit network $G = (V, E)$, which contains at least two nodes $Depot$, for the departure and the arrival, and whose arcs $e \in E$ are endowed with riding times equal to the Euclidean distance between two nodes of V, i and j, $DIST(i, j) \geq 0$, and, eventually, with other technical characteristics;
- a fleet VH of K vehicles k with a capacity CAP,
- a Demand set $D = (D_i, i \in I)$, any demand D_i being defined as a 6-uple $D_i = (o_i, d_i, D_i, F(o_i), F(d_i), Q_i)$, where:
  - $o_i \in V$ is the origin node of the demand D_i and the set of all the o_i is DE,
  - $d_i \in V$ is the destination node of the demand D_i and the set of all the d_i is AR,
  - $\Delta_i \geq 0$ is an upper bound (transit bound) on the duration of demand's processing (ride time),
  - $F(o_i)$ is a time window related to the time D_i starts being processed; $F.MIN_{o_i}$ and $F.MAX_{o_i}$ are the two bounds;
  - $F(d_i)$ is a time window related to the time D_i ends being processed,
  - Q_i is a description of the load related to D_i such that $q_{o_i} = Q_i = -q_{d_i}$,

– $\delta_j, j \in V$ is the non-negative service time necessary at the node j;
– t_i^k is the time at which the vehicle k begins service in i,
– Δ^k is the route maximum time imposed on the vehicle k.

Then, we consider in $G = (V, E)$ all the nodes corresponding to the $o_i \in V$ and $d_i \in V$ such that $V = DE \cup AR \cup \{0, 2\,|D| + 1\}$ with $\{0, 2\,|D| + 1\}$ the two depot nodes respectively for the departure and the arrival, $o_i \in \{1..\,|D|\}$, and $d_i \in \{(|D| + 1)..(2\,|D|)\}$. Moreover we denote by ζ_j^k the total load of the vehicle k leaving the node j, $j \in V$.

Dealing with such an instance means planning the handling demands of D, by the fleet VH, while taking into account the constraints which derive from the technical characteristics of the network G, of the vehicle fleet VH, and of the 6-uples $D_i = (o_i, d_i, D_i, F(o_i), F(d_i), Q_i)$, and while optimizing some performance criterion which is usually minimizing the total distance or a mix of an economical cost (point of view of the fleet manager) and of QoS criteria (point of view of the users).

Let x_{ij}^k a boolean equals to 1 if the vehicle k travels from the node i to the node j. Then, based on [5], the mathematical formulation is the following mixed-integer program :

$$Min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} DIST(i,j) x_{ij}^k \tag{1}$$

subject to

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1, \forall i \in DE \tag{2}$$

$$\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{|D|+i,j}^k = 1, \forall i \in DE, k \in K \tag{3}$$

$$\sum_{j \in V} x_{0j}^k = 1, \forall k \in K \tag{4}$$

$$\sum_{j \in V} x_{ji}^k - \sum_{j \in V} x_{ij}^k = 1, \forall i \in DE \cup AR, k \in K \tag{5}$$

$$\sum_{i \in V} x_{i,2|D|+1}^k = 1, \forall k \in K \tag{6}$$

$$t_j^k \geq (t_i^k + \delta_i + DIST(i,j)) x_{ij}^k, \forall i \in V, j \in V, k \in K \tag{7}$$

$$\zeta_j^k \geq (\zeta_i^k + q_j) x_{ij}, \forall i \in V, j \in V, k \in K \tag{8}$$

$$DIST(i, |(D)| + i) \leq t_{|(D)|+i}^k - (t_i^k + \delta_i) \leq \Delta_i, i \in DE \tag{9}$$

$$F.MIN_i \leq t_i^k \leq F.MAX_i, \forall i \in V, k \in K \tag{10}$$

$$t_{2|D|+1}^k - t_0^k \leq \Delta^k, \forall k \in K \tag{11}$$

$$\zeta_i^k \leq CAP^k \tag{12}$$

$$x_{ij}^k \in \{0;1\}, t \in R^+ \tag{13}$$

The program above is a three index formulation (report to [5] for more explanations about the objective function (1) and the constraints (2)-(13)), it exists in literature several other mathematical formulations for the DARP, even some with two index formulation [8]. But, the complexity of all these linear programs doesn't allow finding an exact solution with a solver, the operation is too time consuming. In fact, it mixes a lot of booleans and plenty of fractional numbers.

All along this work, we are going to deal with homogeneous fleets and with nominal demands, and we shall limit ourselves to static points of view but our insertion process allows flexibility for using it in a dynamic context. Still, we shall pay special attention to cases when temporal constraints are tight.

Discussion: Dynamic versus Static DARP The problem is essentially a problem which arise in dynamic contexts, and the trend is about reactivity delays which become smaller and smaller [1]. Basically, one should consider a system which is identified by a vehicle set V, a user community C, and a supervision system S, which, because of advances in the field of geo-localization, mobile communications and remote monitoring, permanently disposes of a full knowledge about the current state of the vehicles (position, load, roadmap...) and maintains communication with both users and vehicles. All along the time, the system (centralized or decentralized) receives user request, which, in the simplest case, are characterized by a load, an origin and a destination node, and time windows related load and unload transactions, as well as about trip duration. At some instant t, supervisor S decides to launch a scheduling process P, which consider as its input the current state E of the vehicles of V, together with the currently waiting demand set D, and which, for any demand d in D, either rejects it or insert it into the current schedule of some vehicle k in V, without modifying in a significant way the way v is supposed to meet previous demands. Running P require a δ computing time, and, at time t + δ , propositions are transmitted to users and updated schedules are transmitted to the vehicles, which apply them until instant t', when the whole process takes place again. Meanwhile, it may occur that some demands are dropped or that vehicles register failure (delays or user fault) [11].

In any case, one see that, in case vehicles are moving inside a small area (a urban area) and deal with a large size set of demands, process P has to insert in a fast way a demand set D into a current schedule E, and that it has to do it in a way which keeps most features of E, and preserves the ability of the system to efficiently deal with fore coming demands, that means with demands which are likely to be formulated after the instant t when P is launched. This point is the key one which motivates the approach which is going to be described here. We want an algorithmic framework which is going to be naturally compatible with this context: the use of insertion techniques is clearly going to fit the input (E, D) of the dynamic context, and the use of constraint propagation techniques is going to make easier uncertainty about fore coming demands handling.

Also, one should notice that, under this prospect, the virtual complete network which is going to be the key input data for the static model (see next section), is, in practice, going to be a dynamic network.

2.2 The Framework

The Considered Network We treat here the general Dial a Ride Problem described above. It is known that we do not need to consider the whole transit network G = (V, E), and that we may restrict ourselves to the nodes which are either the origin or the destination of some demand, while considering that any vehicle which visits two such nodes in a consecutive way does it according to a shortest path strategy. This leads us to consider the node set $\{Depot, o_i, d_i, i \in I\}$ as made with pairwise distinct nodes, and provided with some distance function DIST, which to any pair x, y in $\{Depot, o_i, d_i, i \in I\}$, makes correspond the shortest path distance from x to y in the transit network G.

As a matter of fact, we also split the Depot node according to its arrival or departure status and to the various vehicles of the fleet VH, and we consider the input data of a Standard Dial a Ride Problem instance as defined by:

- the set {1..K = Card(VH)} of the vehicles of the homogenous fleet VH;
- the common capacity CAP of a vehicle in VH;
- the node set X = {*DepotD*(k), *DepotA*(k), k = 1..K} $\cup$ $\{o_i, d_i, i \in I\}$;
- the distance matrix DIST, whose meaning is that, for any x, y in X, DIST(x, y) is equal to the length, in the sense of the length function l, of a shortest path which connect x to y in the transit network G: we suppose that DIST, satisfies the triangle inequality.

Moreover the following characteristics, which, to any node x in X, make correspond:

- its status Status(x): Origin, Destination, *DepotA*, *DepotD*; we set *Depot* = *DepotD* $\cup$ *DepotA*;
- its load CH(x):
  - if Status(x) $\in$ *Depot* then CH(x) = 0;
  - if Status(x) = Origin then CH(x) = Q_i;
  - if Status(x) = Destination then CH(x) = $-Q_i$;
- its twin node Twin(x):
  - if x = *DepotA*(k) then Twin(x) = *DepotD*(k) and conversely;
  - if x = o_i then Twin(x) = d_i and conversely;
- its time window F(x): for any k = 1..K, F(*DepotA*(k)) = [0, $+\infty$ [= F(*DepotD*(k)). Also, we suppose that any F(x), $x \in X$, is an interval, which may be written F(x) = [F.min(x), F.max(x)];
- its transit bound $\Delta(x)$: if x = o_i or d_i, then $\Delta(x) = \Delta_i$, and $\Delta(x) = \Delta$ else, where Δ is an upper bound which is imposed on the duration of any vehicle tour.

According to this construction, we understand that the system works as follows: vehicle $k \in \{1..K\}$, starts its journey from *DepotD*(k) at some time t(*DepotD*(k)) and ends it into *DepotA*(k) at some time t(*DepotA*(k)), after having taken in charge some subset D(k) =$\{D_i, i \in I(k)\}$ of D: that means that for any i in I(k), vehicle k arrived in o_i at time $t(o_i) \in F(o_i)$, loaded the whole load Q_i, and kept it until it arrived in d_i at time $t(d_i) \in F(oi)$ and unloaded Q_i, in such a way that $t(d_i) - t(o_i) \leq D_i$. Clearly, solving the Standard Dial a Ride Problem instance related to those data (X, DIST, K, CAP) will mean computing the subsets D(k) = $\{D_i, i \in I(k)\}$, the routes followed by the vehicles and the time values t(x), $x \in X$, in such a way that both economical performance and quality of service be the highest possible.

Discussion: Durations and Waiting Times Many authors include what they call service durations in their models. That means that they suppose that loading and unloading processes related to the various nodes of X require some time amount $\delta(x)$, (service time) and, so, that they distinguish, for any node $x \in X$, time values t(x) (beginning of the service) and t(x) + $\delta(x)$ (end of the service). By the same way, some authors suppose that the vehicles are always running at their maximal speed, and make a difference between the time t*(x), $x \in X$, when some vehicle arrives in x, and the time t(x) when this vehicle starts servicing the related demand (loading or unloading process). We do not do it. Taking into account service times, which tends to augment the size of the variables of the model and to make it more complex it, has really sense only if we suppose that the service times $\delta(x)$ depend on the current state (its current load) of the vehicle at the time the loading or unloading process has to be launched. Making explicitly appear waiting times t(x) - t*(x) is really useful if we make appear the speed profile as a component of the performance criterion. In case none of the situation holds, the knowledge of the routes of the vehicles and of the time value t(x), $x \in X$, is enough to check the validity of a given solution and to evaluate its performance, and then it turns out that ensuring the compatibility of the model with data which involve service times and waiting times t(x) - t*(x), $x \in X$, is only a matter of adapting the times windows F(x), the transit bounds $\Delta(x)$, $x \in X$, and the distance matrix DIST.

Discussion: the Homogeneity of the Fleet The general case of the Dial a Ride Problem includes a homogeneous fleet of vehicles. The word "homogeneous" mean the vehicles come from (and come back to) the same depot, and have the same capacity. Our model can integrate different depots and capacities for each vehicle without changing in the framework. Moveover, *DepotD* and *DepotA* locations could be different because all these nodes are independent for a given route.

2.3 Modeling and Evaluation Techniques

The model described in this section needs some definitions, we set:

- First(Γ) = First element of Γ ; Last(Γ) = last element of Γ ;
- for any z in Γ:
  - Succ(Γ, z) = Successor of z in Γ ;
  - Pred(Γ, z) = Predecessor of z in Γ ;
- for any z, z' in Γ:
  - $z \ll_\Gamma z$' if z is located before z' in Γ ;
  - $z \ll_\Gamma^= z$' if $z \ll_\Gamma z$' or z = z';
  - Segment(Γ, z, z') = the subsequence defined by all z in Γ such that $z \ll_\Gamma^= z \ll_\Gamma^= z$'. If z = Nil, then Segment(G, Nil, z') denotes the subsequence defined by all z in Γ such that $z \ll_\Gamma^= z$'.

In any algorithmic description, we use the symbol $\leftarrow$ in order to denote the value assignment operator: x $\leftarrow \alpha$, means that the variable x receives the value α. Thus, we only use symbol = as a comparator.

In order to provide an accurate description of the output data of our standard Dial a Ride Problem instance (X, DIST, K, CAP), we need to talk about tours and related time value sets. A tour Γ is a sequence of nodes of X, which is such that: 3

- Status(First(Γ)) = *DepotD*; Status(End(Γ)) = *DepotA*;
- For any node x in Γ, x $\neq$ First(Γ), End(Γ), Status(x) $\notin$ *Depot*;
- No node $x \in X$ appears twice in Γ ;
- For any node x = o_i (resp. d_i) which appears in Γ, the node Twin(x) is also in Γ, and we have: $x \ll_\Gamma Twin(x)$ (resp. $Twin(x) \ll_\Gamma x$).

This tour Γ is said to be load-valid iff:

- for any x in Γ, $x \neq First(G)$, we have $\sum_{y, y \ll_\Gamma x} CH(y) \leq CAP$.

Moreover, this tour Γ is said to be time-valid iff it is possible to associate, with any node x in Γ , some time value t(x), in such a way that: (E1)

- for any x in Γ, $x \neq Last(\Gamma)$, $t(Succ(\Gamma, x)) \geq t(x) + DIST(x, Succ(\Gamma, x))$;
- for any x in Γ, $|t(twin(x)) - t(x)| <= \Delta(x)$;
- for any x in Γ, $t(x) \in F(x)$.

In case the tour Γ is time-valid, any time value set t = {t(x), $x \in X$}, which satisfies (E1) is said to be a valid related time value set.

The tour Γ is said to be valid if it is both time valid and load valid.

For any pair (Γ, t) defined by some time-valid tour Γ and by some valid related time value set t, we may set:

- *Glob*(Γ, t) = t(End(Γ)) - t(First(Γ)): this quantity denotes the global duration of the tour Γ ;
- *Ride*(Γ, t) = $\sum_{i \in \Gamma}(t(d_i) - t(o_i))$; this quantity may be viewed as a QoS criterion, and denotes the sum of the duration of the individual trips of the demanders which are taken in charge by tour Γ ;

– $Wait(\Gamma, t) = Glob(\Gamma, t)$ - $(\sum_{x, x \neq Last(G)} DIST(x, Succ(\Gamma, x)))$: this quantity denotes the waiting time of the vehicle involved in Γ, the waiting time related to some node x being the time the vehicle is supposed to wait before loading or unloading x in case it runs full speed on the route which connects Pred(Γ, x) to x.

If A, B, C are three multi-criterion coefficients, we may define the performance criterion $Cost_{A,B,C}$ (Γ, t) as follows: $Cost_{A,B,C}(\Gamma, t) = A.Glob(\Gamma, t) + B.Ride(\Gamma, t) + C.Wait(\Gamma, t)$.

In the experiments section we will use different criteria in order to compare with other techniques found in literature. Our insertion techniques allow some flexibility for this change.

So, let us suppose that we deduced from the data G = (V, E), VH = (K, CAP), $D = (D_i = (o_i, d_i, \Delta_i, F(o_i), F(d_i), Q_i), i \in I)$, a 4-uple (X, DIST, K, CAP), and that we are also provided with 3 multi-criterion coefficients A, B and $C \geq 0$. Then we see that solving the related Standard Dial a Ride Problem instance means computing:

– for any vehicle index k in 1..K, a valid tour T(k);
– a time value set $t = \{t(x), x \in X\}$;

in such a way that:

– the restriction of t to any T(k), k = 1..K, defines a valid time value set related to T(k);
– the tour set T = {T(k), k = 1..K} induces a partition of X;
– the quantity $Perf_{A,B,C}(T, t) = \sum_{k=1..K} Cost_{A,B,C}(T(k), t)$ is the smallest possible.

3 Constraint Propagation into an Insertion Algorithm

3.1 Handling Constraints

Let Γ a tour. The algorithm which we are going to describe in this section will essentially be based upon the use of insertion techniques. Thus, we must be able to check in a fast way, whether the insertion of some demand D_i inside Γ will maintain the validity of Γ, and to get an evaluation of the quality of this insertion. Since we want to pay a special attention to the case when temporal constraints are tight, we are first going to provide ourselves with a package of constraint handling tools for testing the valid tours.

First, checking the load validity of Γ is easy. In order to be able to test the impact of the insertion of some demand into the tour Γ on the charge-validity of this tour, we associate, with any such a tour, the quantities $\zeta(\Gamma, x)$, defined by:

– for any x in Γ, $\zeta(\Gamma, x) \quad = \sum_{y, y \ll_{\Gamma}^{=} x} CH(y)$.

Then it comes that Γ is load-valid iff for any x in Γ, $\zeta(\Gamma, x) \leq CAP$.

Second, checking the time validity of Γ, according to a current time window set FS = {FS(x) = [FS.min(x), FS.max(x)], $x \in \Gamma$} may be performed through propagation of the following inference rules R_i, i = 1..5. We denote by NFact a list of nodes related to time constraints non propagated. The five inferences rules are:

Rule R1 (if (y = Succ(Γ, x))):

$$\begin{array}{c} \text{FS.min(x) + DIST(x, y)} > \text{FS.min(y)} \\ \models \text{FS.min(y)} \leftarrow \text{FS.min(x) + DIST(x, y); NFact} \leftarrow \text{y;} \end{array}$$

Rule R2 (if (y = Succ(Γ, x))):

$$\begin{array}{c} \text{FS.max(y) - DIST(x, y)} < \text{FS.max(x)} \\ \models \text{FS.max(x)} \leftarrow \text{FS.max(y) - DIST(x, y); NFact} \leftarrow \text{x;} \end{array}$$

Rule R3 (if (y = Twin(x)) and ($x \ll_\Gamma y$)):

$$\begin{array}{c} \text{FS.min(x)} < \text{FS.min(y) - } \Delta\text{(x,y)} \\ \models \text{FS.min(x)} \leftarrow \text{FS.min(y) - } \Delta\text{(x,y); NFact} \leftarrow \text{x;} \end{array}$$

Rule R4 (if (y = Twin(x)) and ($x \ll_\Gamma y$))

$$\begin{array}{c} \text{FS.max(y)} > \text{FS.max(x)} + \Delta\text{(x,y)} \\ \models \text{FS.max(y)} \leftarrow \text{FS.max(x)} + \Delta\text{(x,y) ; NFact} \leftarrow \text{y;} \end{array}$$

Rule R5 (if x $\in \Gamma$):

$$\begin{array}{c} \text{FS.min(x)} > \text{FS.max(x)} \\ \models \text{Fail.} \end{array}$$

Propagating these rules may be performed as follows:

Procedure Propagate
Input: (Γ : Tour, L: List of nodes, FS: Time windows set related to the node set of Γ);
Output: (Res: Boolean, FR: Time windows set related to node set of Γ);
Not Stop;
While L $\neq$ Nil and Not Stop do
  z $\leftarrow$ First(L); L $\leftarrow$ Tail(L);
  For i = 1..5 do Compute all the pairs (x, y) which make possible
  an application of the rule R_i and which are such that x = z or y = z;
    For any such pair (x, y) do
      Apply the rule Ri;
      If NFact is not in L then Insert NFact in L;
      If Fail then Stop;
Propagate $\leftarrow$ (Not Stop, FS);

Proposition 1

The tour Γ is time-valid according to the input time window set FS if and only if the Res component of the result of a call Propagate(FS, Γ) is equal to 1. In such a case, any valid time value set t related to Γ and FS is such that: for any x in Γ, t(x) $\in$ FS(x).

Proof

The part (only if) of the above equivalence is trivial, as well as the second part of the statement. As for the part (if), we only need to check that if we set, for any x in Γ:

- FS(x) = [FS.min(x), FS.max(x)];
- t(x) = FS.min(x);

then we get a time value set t =$\{$t(x), x $\in$ X(Γ)$\}$ which is compatible with Γ and FS.

End-Proof.

We denote by FP(Γ) the time window set which result from a call Propagate(Γ, L, F). FP(Γ) may be considered as the largest (in the inclusion sense) time window set which is included into F and which is stable under the rules R_i, i = 1..5, and is called the window reduction of F through Γ.

3.2 Evaluating a Tour

Let us consider now the tour Γ, provided with the window reduction set FP(Γ). We want to get some fast estimation of the best possible value $Cost_{A,B,C}(\Gamma, t) = A.Glob(\Gamma, t) + B.Ride(\Gamma, t) + C.Wait(\Gamma, t), t \in Valid(\Gamma)$. We already noticed that it could be done through linear programming or through general shortest path and circuit cancelling techniques. Still, since we want to perform this evaluation process in a fast way, we design two ad hoc procedures EVAL1 and EVAL2:

- the EVAL1 procedure works in a greedy way, by first assigning to the node First(Γ) its largest possible time value, and by next performing a Bellman process in order to assign to every node x in Γ its smallest possible time value.
- the EVAL2 procedure starts from a solution produced by EVAL1, and improves it by performing a sequence of local moves, each move involving a single value t(x), $x \in \Gamma$.

Γ being some valid tour, we denote by VAL1(Γ) and VAL2(Γ) the values respectively produced by the application of EVAL1 and EVAL2 to Γ.

3.3 The Insertion Mechanism

The insertion process works in a very natural way. Let Γ be some valid tour, let D_i=(o_i, d_i, Δ_i, $F(o_i)$, $F(d_i)$, Q_i) be some demand whose origin and destination nodes are not in Γ, and let x, y be two nodes in Γ, such that $x \ll\!\!=_{\Gamma} y$. Then we denote by INSERT(Γ, x, y, i) the tour which is obtained by:

- locating o_i between x and Succ(Γ, x);
- locating d_i between y and Succ(Γ, y).

We say that the tour **INSERT(Γ, x, y, i)** results from the insertion of demand D_i into the tour Γ according to the insertion nodes x and y. The tour INSERT(Γ, x, y, i) may not be valid. So, before anything else, we must detail the way the validity of this tour is likely to be tested.

Testing the Load-Admissibility of INSERT(Γ, x, y, i)

We only need to check, that for any z in Segment(Γ, x, y) = {z such that $x \ll\!\!=_{\Gamma} z \ll\!\!=_{\Gamma} y$} we have, $\zeta(\Gamma, x) + Q_i \leq$ CAP. It comes that we may set:

Procedure Test-Load(Γ, x, y, i):
Test-Load $\leftarrow$ {For any z in Segment(Γ,x, y), $\zeta(\Gamma, x) + Q_i \leq$ CAP};

Testing the Time-Admissibility of INSERT(Γ, x, y, i)

It should be sufficient perform a call Propagate(Γ, {o_i, d_i}, FP(Γ)), while using the list {o_i, d_i} as a starting list. Still, such a call is likely to be time consuming. So, in order to make the testing process go faster, we introduce several intermediary tests, which aim at interrupting the testing process in case non-feasibility can be easily noticed:

- the first test Test-Node aims at checking the feasibility of the insertion of a node u, related to some load Q, between two consecutive node z and z' of a given tour Γ. It only provides us with a necessary condition for the feasibility of this insertion.
- the second test Test-Node1 aims at checking the feasibility of the insertion of an origin/destination node u, v, related to some load Q, between two consecutive node z and z' of a given tour Γ (e.g. into an empty tour). Again, it only provides us with a necessary condition for the feasibility of this insertion.

So, testing the admissibility of a tour INSERT(Γ, x, y, i) may be performed through the following procedure:

Procedure Test-Insert(Γ, x, y, i): (Test: Boolean, Val: Number);
If $x \neq y$ then
  Test $\leftarrow$ Test-Node(Γ, x, Succ(Γ, x), o_i, Q_i) $\wedge$ Test-Node(Γ, y, Succ(Γ, y), d_i, Q_i);
Else Test $\leftarrow$ Test-Node1(Γ, x, Succ(Γ, x), o_i, d_i, Q_i);
If Test = 1 then Test $\leftarrow$ Test-Load(Γ, x, y, i);

If Test = 1 then (Test, F1) ← Propagate(Γ, {o_i, d_i }, FP(Γ);
If Test = 1 then Val ← EVAL1(INSERT(Γ, x, y, i), F1).Val;
Else Val ← Undefined;
Test-Insert ← (Test, Val - Val1(Γ));

3.4 The Insertion Process

So, this process takes as input the demand set D = (D_i=(o_i, d_i, Δ_i, $F(o_i)$, $F(d_i)$, Q_i)), i ∈ I), the 4-uple (X, DIST, K, CAP), and 3 multi-criteria coefficients A, B and C $\geq$ 0, and it works in a greedy way through successive insertions of the various demands $D_i = (o_i, d_i, \Delta_i, F(o_i), F(d_i), Q_i)$ of the demand set D. The basic point is that, since we are concerned with tightly constrained time windows and transit bounds, we use, while designing the INSERTION algorithm, several constraint propagations tricks. Namely, we make in such a way that, at any time we enter the main loop of this algorithm, we are provided with:

- the set $I_1 \subset I$ of the demands which have already been inserted into some tour T(k), k = 1..K;
- current tours T(k), k = 1..K: for any such a tour T(k), we know the related time windows FP(T(k))(x), $x \in T(k)$, as well as the load values $\zeta(T(k), x), x \in T(k)$, and the values VAL1(T(k)) and VAL2(T(k));
- the knowledge, for any i in J = (I - I1) of the set U_{free}(i) of all the 4-uple (k, x, y, v), k = 1..K, $x, y \in T(k)$, $v \in Q$, such that a call Test-Insert(T(k), x, y, i) yields a result (1, v). We denote by N_{free}(i) the cardinality of the set K_{free} = {k = 1..K, such that there exists a 4-uple (k, x, y, v) in U_{free}(i)}: N_{free}(i) provides us with the number of vehicles which are still able to deal with demand D_i.

Then, the INSERTION algorithm works according to the following scheme:

- First, it picks up some demand i_0 in J, among those demands which are the most constrained, that means which are such that $N_{free}(i_0)$ is small: more specifically, if there exists i such that N_{free}(i) = 1, then i_0 is chosen in a random way among those demand indices i in J which are such that N_{free}(i) = 1; else we select randomly in a set of demands j with N_{free}(j) inside {2, $N_{freeMAX}$ }. $N_{freeMAX}$ becomes a parameter of the INSERTION. (E2)
- Next, it picks up (k_0, x_0, y_0, v_0) in $U_{free}(i_0)$ which simultaneously corresponds to one of the smallest values v, and to one of the smallest values EVAL2(INSERT(T(k), x, y, i_0)).Val - VAL2(T(k)): more specifically it first builds the list L-Candidate of the N_1 (up to five) 4-uples (k, x, y, v) in U_{free}(i_0) with best (smallest value v). For any such a 4-uple, it computes the value w = EVAL2(INSERT(T(k), x, y, i_0)).Val - VAL2(T(k)), and it orders L-Candidate according to increasing values w. Then it randomly chooses (k_0, x_0, y_0, v_0) among those N2 $\leq$ N1 first 4-uples in L-Candidate. N_1 and N_2 become two parameters of the INSERTION procedure. (E3)
- Next it inserts the demand i_0 into T(k_0) according to the insertion nodes x_0, y_0, which means that it replaces T(k_0) by INSERT(T(k_0), x_0, y_0, i_0);

– Next it defines, for any $i \in J$, the set $\Lambda(i)$ as being the set of all pairs (x, y) such that there exists some 4-uple (k_0, x', y', v) in U_{free}(i), which satisfies:
  - $(x' = x)$ or $((x' = x_0)$ and $x' = \text{Pred}(T(k_0), x))$ or $((x' = x_0 = y_0)$ and $(x' = \text{Pred}(\text{Pred}(T(k_0),x))))$;
  - $(y' = y)$ or $((y' = y_0)$ and $y' = \text{Pred}(T(k_0), y))$ or $((y' = x_0 = y_0)$ and $(y' = \text{Pred}(\text{Pred}(T(k_0),y))))$; (E4)
– Finally, it performs, for any pair (x, y) in $\Lambda(i)$, a call Test-Insert(T(k_0), x, y, i), and it updates U_{free}(i) and N_{free}(i) consequently.

This can be summarized as follows:

Procedure INSERTION(N_1 and N_2 : Integer): (T: tour set, t: time value set, Perf: induced $Perf_{A,B,C}$(T, t) value, Reject: rejected demand set);
For any k = 1..K do
T(k) ← {*DepotD*(k), *DepotA*(k)};
t(*DepotD*(k)) = t(*DepotA*(k)) ← 0;
I1 ← Nil ; J ← I ; Reject ← Nil;
For any $i \in J$ do
N_{free}(i) ← K;
U_{free}(i) ← all the possible 4-uple (k, x, y, v), $k \in K$, x, y $\in$ {*DepotD*(k), *DepotA*(k)}, $x \ll_{T(k)} y$, v = **EVAL2**({*DepotD*(k), o_i, d_i, *DepotA*(k)}).Val;
While $J \neq Nil$ do
Pick up some demand i_0 in J as in (E2); Remove i_0 from J;
If $U_{free}(i_0)$ = Nil then $Reject \leftarrow Reject \cup \{i0\}$;
Else
Derive from $U_{free}(i_0)$ the L-Candidate list and pick up (k_0, x_0, y_0, v_0) in L-Candidate as in (E3);
$T(k_0) \leftarrow$ **INSERT**$(T(k_0), x_0, y_0, i_0)$;
$\delta \leftarrow$ **EVAL2**$(T(k_0)).\delta$; Insert i_0 into I_1 ;
For any x in T(k_0) do $t(x) \leftarrow \delta(x)$;
For any $i \in J$ do
$\Lambda(i) \leftarrow$ {all pairs (x, y) such that there exists some 4-uple (k_0, x', y', v) in U_{free}(i), which satisfies (E4);
For any pair (x, y) in $\Lambda(i)$ do
(Test, Val) ← **Test-Insert**(T(k_0), x, y, i);
Remove (k_0, x, y, v) from U_{free}(i) in case such a 4-uple exists and update N_{free}(i) consequently;
If Test = 1 then insert (k_0, x, y, Val) into U_{free}(i) and update N_{free}(i) consequently;
Perf ← $Perf_{A,B,C}$ (T, t);
INSERTION ← (T, t, Perf, Reject);

Since the above (I1) and (I2) instruction may be written in a non deterministic way, the whole INSERTION algorithm becomes non deterministic and may be used inside some MONTE-CARLO framework:

RANDOM-INSERTION(N_1, N_2, P: Integer) Scheme;
For p = 1..P do
Apply the INSERTION(N_1, N_2) procedure;
Keep the best result (the pair (T, t) such that |Reject| is the smallest possible, and which is such that, among those pairs which minimize |Reject|, it yields the best $Perf_{A,B,C}$ (T, t) value).

4 Dial-a-ride Problem with Split Loads

4.1 Model and Framework updated

The Dial-a-ride problems with split loads means we allow related to some demand to be split in several pieces and to transported separately. Such a situation may occur in the case of good transportation (large scale load management) as well as in the case of people transportation (group management). Difficulties start with modeling, since the way loads Q_i are divided into load-pieces $Q_{i,j}$, $j = 1..n(i)$, is part of the problem.

We based on the general Dial-a-ride Problem defined above and we update :

- the set X which gives rise to a infinite set Z = Z(X), which derives from X by replacing every node x such that Status(x) = {Origin, Destination}, by nodes (x, s), $s \in N$. This splitting process will allow us to distinguish the nodes of X which are related to some demand D_i according to the load-pieces $Q_{i,j}$, j = 1..n(i): the meaning of node (o_i, s) is that if a tour T(k) contains this node (o_i, s), then it will also contain the node (d_i, s), and vehicle k will ensure the transportation of some load-piece $Q_{i,j}$ from o_i to d_i.
- the DIST matrix which may be considered as extended in a natural way as a DIST function which is defined on Z.Z;
- the Twin function : for any node z = (x, s) = (o_i, s) ((d_i, s)) which appears in Γ, the node Twin(z) = (d_i, s) ((o_i, s)) is also in Γ, and we have: z << Twin(z) ((Twin(z) << z));
- the ride is computed by the duration between the first time's origin node and the last time's destination node for a given demand;
- the maximum ride time could be considered in two different ways (for a given demand) :
  - it bounds the duration given by the first time's origin node and the last time's destination node,
  - each load-pieces is independent and bounded by the same maximum ride time (like in our experiments).

So, the Dial-a-Ride Problem with split loads may be put in a formal way as follows:

The dial-a-ride problem with split loads
Input: the demand set D = ($D_i = (o_i, d_i, \Delta_i, F(o_i), F(d_i), Q_i)$, $i \in I$), the 4-uple

(X, DIST, K, CAP) which we defined above, and 3 multi-criteria coefficients A, B and $C \geq 0$;
Output: a triple (T, t, Q) where T = {T(k), $k \in K$} is a time valid tour family, Q = {Q(k), $k \in K$} is a family of related valid load value sets Q(k) = {Q(k)(z), $z \in T(k)$}, and t = {t(k), $k \in K$} is a family of related valid time value sets t(k) = {t(k)(z), $z \in T(k)$} such that:

- for every $i \in I$, we have:
  $\sum_{k=1..K} \sum_{(i,s) \text{ if } o_i \in T(k)} Q(k)(o_i, s) = - \sum_{k=1..K} \sum_{(i,s) \text{ if } d_i \in T(k)} Q(k)(d_i, s)$
- The quantity $Perf_{A,B,C}(T, t, Q) = \sum_{k=1..K} Cost_{A,B,C}(T(k), Q(k), t(k))$ is the smallest possible.

Active set related to a feasible triple (T, t, Q): it is the set of the active nodes (x, s) in Z = Z(X), which means the nodes which belong to some tour T(k), k = 1..K. The general algorithmic scheme **INSERTION-SPLIT-LOADS** will come as follows:

Initialize (T, t, Q); Initialize the sets U_{free}(i), $i \in I$;
Initialize the active Z-ACT: Z-ACT <- Nil;
J <- I; For any i in J, set $Q\text{-}Aux_i$ <- Q_i; Reject <- Nil;
While $J \neq Nil$ do
  Picks up some demand i_0 in J; Remove i_0 from J;
  If $U_{free}(i_0)$ = Nil then Reject <- Reject $\cup \{i_0, Q\text{-}Aux_{i_0}\}$;
  Else
    Compute s_0; Create two new active nodes (o_{i_0}, s_0) and (d_{i_0}, s_0) and insert them into Z-ACT;
    Derive from $U_{free}(i_0)$ a L-Candidate list;
    Pick up $(k_0, x_0, y_0, v_0, Q(k_0)(o_{i_0}, s_0))$ in L-Candidate; (E5)
    $T(k_0)$ <- INSERT($T(k_0)$, x_0, y_0, i_0);
    Update t and Q;
    Update the sets U_{free}(i), i ∈ J;
    If $Q(k_0)(o_{i_0}, s_0) = Q\text{-}Aux_{i_0}$ then Remove i_0 from J else replace $Q\text{-}Aux_{i_0}$ by $Q\text{-}Aux_{i_0}$ - $Q(k_0)(o_{i_0}, s_0)$;

4.2 Trade-off between load and speed: the Load-Distribute Problem.

As a matter of fact, performing Instruction (E5) above, which means conveniently the parameters k_0, x_0, y_0 and $Q(k_0)(o_{i_0}, s_0)$ of the insertion process, also means defining some trade-off between the value $Q(k_0)(o_{i_0}, s_0)$, which we would like to be the larger possible, and the quality of the insertion in relation to the criterion measure $Perf_{A,B,C}(T, t)$ and to the values which are returned by EVAL1 and EVAL2. In order to define this trade-off, we do not exactly follow the above algorithmic scheme: instead, we proceed in a specific way, which consists in handling the whole demand D_{i_0} inside a same iteration, while eventually splitting into several blocks and simultaneously distributing those blocks in an

ad hoc way between the different tours. In order to put it in a more precise way, let us suppose that we are dealing as above with a demand index i_0 in J, in such a way that $Q\text{-}Aux_{i_0} = Q_{i_0}$ and with a L-Candidate list which we derived from the set $U_{free}(i_0)$. The elements of L-Candidate are 5-uple (k, x, y, v, q), which express the feasibility of the transportation by vehicle k of a load q from o_{i_0} to d_{i_0}, respectively inserted into the tour T(k) between x and Succ(T(k), x) and between y and Succ(T(k), y), the number v providing us with the EVAL2 value of this insertion. Then we try to solve the following problem:

Load-Distribute Problem

{Select a collection $\Lambda = \{(k_1, x_1, y_1, v_1, q_1),.., (k_s, x_s, y_s, v_s, q_s)\}$ of 5-uples of L-Candidate, in such a way that:

- the k_j, j = 1..s, are pairwise distincts;
- $\sum_{j=1..s} \ q_j \geq \ Q_{i_0}$;
- $\sum_{j=1..s} \ v_j$ is the smallest possible}

While this problem may be easily solved in an exact way through a bipartite graph matching procedure, we deal with it in a fast way through a simple heuristic Load-Distribute procedure. In case we don't find any feasible solution to this problem, then we reject the whole demand i_0. Else, we create the active nodes (o_{i_0}, j), (d_{i_0}, j), j = 1..s , we add them to Z, and, for every index value j = 1..s, we replace the tour T(k_j) by the tour INSERT(T(k_j), x_j, y_j, i_0), and we consequently update the time value set t(k_i) and the load value set Q(k_i).

Remark 1. The failure of the Load-Distribute test does not completely mean that the insertion of demand i_0 cannot be performed: theoretically, one might build instances which would make a distributed insertion possible, under the condition that a same vehicle is going to support several distinct nodes (o_{i_0}, j). In such a case, we should perform the insertion of a first part of demand i_0, and next try again with the remaining part, while eventually using the same vehicle as for the first part. Still, practically, such a configuration is likely to occur very scarcely, and, so, we decide not to take it into account.

5 Computational Experiments

5.1 Experiment on the classic Dial a Ride Problem

This first experiment deals with the two sets of instances defined in [5]. We integrated the same mono criterion objective function given by Cordeau: the minimization of the total distance. The instances have between 16 and 48 requests which have to be supported by a fleet of 2 to 4 vehicles, and have been divided into subsets *a* and *b*. In the first set, $CAP = 3$, the loads are all equal to 1, and the maximum riding time is 30min. For the second set, $CAP = 6$, the load q is randomly chosen according to a uniform distribution such as $q = 1..CAP$,

Inst.	Lb	$Opti$	Ub	$c1(s)$	TI	Gap	$Wait$	$Ride$	$c2(s)$
a2-16	294.25	294.25	294.25	1.1	294.25	0.00	387.32	344.54	0.0
a2-20	344.83	344.83	344.83	2.6	344.83	0.00	605.44	455.32	0.1
a2-24	431.12	431.12	431.12	8.5	431.12	0.00	536.79	603.31	0.3
a3-18	300.48	300.48	300.48	4.6	300.81	0.11	196.65	419.35	0.7
a3-24	344.83	344.83	347.42	7.6	344.83	0.00	642.72	628.86	1.5
a3-30	494.85	494.85	494.85	9.8	495.26	0.08	721.21	732.86	16.3
a3-36	583.19	583.19	584.44	105.1	589.86	1.14	868.83	903.77	13.8
a4-16	282.68	282.68	282.68	5.6	283.10	0.15	100.72	307.00	0.3
a4-24	375.02	375.02	378.13	5.6	376.21	0.32	527.81	581.60	94.0
a4-32	485.5	485.5	487.81	30.7	487.10	0.33	593.75	796.45	29.4
a4-40	557.69	557.69	582.26	8328.5	565.95	1.42	1112.33	824.32	63.3
a4-48	668.82	NA	709.47	14542.6	700.30	NA	966.85	1132.92	30.8

Table 1. Resolution of the set a [5]

and the maximum riding time is 40min. All the demands are randomly chosen in the square [-10,10].[-10,10] according to a uniform distribution, and all the routing costs between two nodes are equal to the Euclidean distance. The heuristics proposed in this paper was implemented in C++ and compiled with GNU GCC. Each replication was run on the same thread of an Intel Q8300 (2.5 GHz).

Table 1 shows the results obtained for the a first set of instances, and table 2 gives the results for the b second set. Lb, Ub, $Opti$, and TI are the best lower bound, the best upper bound, the known optimal value ([5]-[14]), and the result obtained with our insertion techniques respectively. The cpu times are in seconds for the first table and in minutes for the second table. $c1$ is the literature best cpu time and $c2$ is the cpu time obtained in our experiment. *gap* is the gap in percentage between the optimal distance and our result.

Almost each time, our heuristic found the optimal solution known in the literature and the worst gap obtained was 2,38%. We obtained these results quickly, the cpu times are low compared to previous studies, and a good solution is obtained in little time. Also, we show that our solution can be used with other objective functions, proving one the flexibility aspects of our solution.

5.2 Experiment on the Dial a Ride Problem with split loads

In this section we present our instances generated in the same square [-10,10].[-10,10] as above in order to test our heuristic that solves the DARPSL. In all demands, the *origin* time window is tight (15 minutes) and the *destination* time window is large. Moreover, their *origin* location are randomly located in the rectangle [-10,-9].[-10,10], and their *destination* location is generated in the square [9,10].[-0.5,0.5]. The depot point is located on the center of the main square. We generated four sets of 10 instances (20 to 65 demands managed by 4 to 10 vehicles). All the random generation has been computed by a uniform distribution.

The optimization uses a mono criterion : the minimization of the total distance. Our two algorithms proposed in this paper were applied to the four sets,

Inst.	*Lb*	*Opti*	*Ub*	$c1(m)$	TI	*Gap*	*Wait*	*Ride*	$c2(m)$
b2-16	309.41	309.41	309.61	0.2	309.41	0.00	386.45	448.66	0.7
b2-20	332.64	332.64	334.93	0.0	332.64	0.00	458.17	465.23	0.6
b2-24	444.71	444.71	445.11	0.1	444.71	0.00	475.88	674.12	2.6
b3-18	301.64	301.64	301.8	0.7	301.65	0.00	278.70	479.38	0.7
b3-24	394.51	394.51	394.57	3.6	397.47	0.75	609.62	572.61	3.5
b3-30	531.44	531.44	536.04	6.8	534.23	0.52	785.71	857.28	3.2
b3-36	603.79	603.79	611.79	62.1	603.79	0.00	919.58	942.81	0.9
b4-16	296.96	296.96	299.07	0.8	296.96	0.00	218.97	402.16	2.8
b4-24	371.41	371.41	380.27	5.9	371.41	0.00	490.06	567.75	0.1
b4-32	494.82	494.82	500.92	176.8	506.60	2.38	921.55	749.85	1.9
b4-40	591.76	656.6	662.91	240.0	662.74	0.94	1013.20	1021.47	3.5
b4-48	586.91	673.8	685.46	240.0	684.83	1.64	1458.76	1262.59	5.2

Table 2. Resolution of the set *b* [5]

K	$\|D\|$	*Glob*	*Ride*	*Dist*	R_{Succ}	R_{Insert}	$cpu(s)$
4	20	836.3	953.8	416.9	71.2	98.1	0.3
6	35	1366.7	1590.7	714.2	58.0	98.0	0.7
8	50	1831.0	2203.1	1038.4	28.4	96.3	1.4
10	65	2349.7	2758.5	1347.2	25.2	95.8	2.4
Av.		1595.9	1876.5	879.2	45.7	97.0	1.2

Table 3. Instances solved by the DARP's heuristic

each instance has been solved with 100 runs. Table 3 and table 4 report the results obtained with the classic problem and the problem with split loads respectively. *Glob* and *Ride* are the times reported from the best run, *Dist* is the best total distance obtained, R_{Succ} (%) is the insertion average rate (over the runs) of all the demands, R_{Insert} (%) the average rate of insertion for each demand, and $cpu(s)$ is the time in seconds for the 100 runs.

K	$\|D\|$	*Glob*	*Ride*	*Dist*	R_{Part}	R_{Succ}	R_{Insert}	$cpu(s)$
4	20	856.3	1113.1	356.8	1.205	93.2	99.6	0.5
6	35	1269.2	1787.2	605.5	1.216	78.4	98.8	1.3
8	50	1689.7	2542.1	850.2	1.216	53.6	97.8	2.6
10	65	2024.7	2981.0	1100.3	1.220	47.6	97.1	4.2
Av.		1460.0	2105.9	728.2	1.214	68.2	98.3	2.1

Table 4. Instances solved by the DARPSL's heuristic

We report the average number of divisions per demand (R_{Part}). So, for the four sets, this rate is 1,214 (each demand is divided by 1,214 on average).

We observed that the split loads gives us better solutions, we obtained distances 20% less than the other problem. *Glob* also decreased (the average number of vehicles leaving the depot is lower than the classic problem). *Ride* increased

because it is computed on the difference between the last date of the pickups at the destination point and the first date of the pickups at the origin point.

6 Conclusion

The static multi-vehicle DARP with Time Windows requires approximate solutions in order to be solved in a reasonable time. We have described an implementation of some insertion techniques using constraint propagation. This solution makes it possible to obtain good results in little time. In addition, we formulate an objective function which optimizes the combination of QoS and cost's minimization. But, in order to compare with tests found in literature, we prove the flexibility of our framework by changing the objective function without modification of the framework itself. Despite this change, we show that our solution is effective.

We also propose a new problem: the Dial a Ride Problem with split loads. This problem gives us better solving method compared to the classic DARP, we get 20% shorter routes with the resolution of the DARPSL with our instances.

In a future work, we could solve instances in real context, and improve our solution by integrating *inserability* demand calculator.

7 Acknowledgments

We wish to thank you the Conseil Regional d'Auvergne and the FEDER of the European Union.

References

[1] G. Ghiani G. Laporte A. Attanasio, J.F. Cordeau. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing. Volume 30, Issue 3, Page 377-387*, 2004.

[2] A. Hertz C. Archetti, M.G. Speranza. A tabu search algorithm for the split delivery vehicle routing problem. *Transportation Science, 40(1) :64–73*, 2006.

[3] M.G. Speranza C. Archetti. The vehicle routing problem : Latest advances and new challenges, chapter the split delivery vehicle routing problem : A survey. *pages 103–122. Springer US*, 2008.

[4] R. Chevrier. Optimisation de transport à la demande dans des territoires polarisés. *PhD. Thesis. Université d'Avignon et des Pays de Vaucluse, 242p*, 2008.

[5] J-F Cordeau. A branch-and-cut algorithm for the dial-a-ride. *Operation Research. Vol, 54, n°3, p 573-586*, 2006.

[6] J. Jaw A. Odoni H. Psaraftis, N. Wilson. A heuristic algorithm for the multi-vehicle many-to-many advance request dial-a-ride problem. *Transportation Research B 20B, 243-257*, 1986.

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

[8] G. Laporte J.F. Cordeau. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 2007.

[9] J.R. Stone J.W. Baugh Jr., D.K.R. Kakivaya. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization, 30(2): 91-124*, 1998.

[10] P. Trudeau M. Dror. Savings by split delivery routing. *Transportation Science, 23(2) :141–145*, 1989.

[11] A Schrijver M. Grötschel, L. Lovász. Geometric algorithms and combinatorial optimization. *Springer-Verlag*, 1988.

[12] J. Rygaard O. Madsen, H. Ravn. A heuristic algorithm for the a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research 60, 193–208*, 1995.

[13] R. Moll P. Healy. A new extension of local search applied to the dial-a-ride problem. *European Journal of Operational Research 83, 83–104*, 1995.

[14] S. Parragh. Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Research Part C : Emerging Technologies. Volume 19, Issue 5, August 2011, Pages 912–930*, 2011.

[15] S. N. Parragh. Solving the dial-a-ride problem with split requests and profits. *CO 2012*, 2012.

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

[17] P. Chatonnay D. Josselin R. Chevrier, P. Canalda. Comparison of three algorithms for solving the convergent demand responsive transportation problem. *ITSC'2006, 9th Int. IEEE Conf. on IntelligentTransportation Systems, Toronto, Canada, 1096–1101*, 2006.

[18] K.B. Bergvinsdottir R.M. Jorgensen, J. Larsen. Solving the dial-a-ride problem using genetic algorithms. *Journal of the Operational Research Society, 58(10):1321-1331*, 2007.

[19] A. Quilliot S. Deleplanque. Dial a ride problem avec transbordement et division du chargement. *14e conférence ROADEF. 14-15-15 Février 2013 (résumé accepté)*, 2013.

[20] R.F. Hartl S.N. Parragh, K.F. Doerner. Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research, 37 p1129–1138*, 2010.