# Anticipation in the Dial-a-Ride Problem: an introduction to the robustness

Samuel Deleplanque
Blaise Pascal University
LIMOS CNRS Laboratory
LABEX IMOBS3
Clermont-Ferrand 63000, France
Email: deleplan@isima.fr

Jean-Pierre Derutin
Blaise Pascal University
Institut Pascal CNRS Laboratory
LABEX IMOBS3
Clermont-Ferrand 63000, France
Email: derutin@univ-bpclermont.fr

Alain Quilliot
Blaise Pascal University
LIMOS CNRS Laboratory
LABEX IMOBS3
Clermont-Ferrand 63000, France
Email: quilliot@isima.fr

***Abstract*—The Dial-a-Ride Problem (DARP) models an operation research problem related to the on demand transport. This paper introduces one of the fundamental features of this type of transport: the robustness. This paper solves the Dial-a-Ride Problem by integrating a measure of insertion capacity called Insertability. The technique used is a greedy insertion algorithm based on time constraint propagation (time windows, maximum ride time and maximum route time). In the present work, we integrate a new way to measure the impact of each insertion on the other not inserted demands. We propose its calculation, study its behavior, discuss the transition to dynamic context and present a way to make the system more robust.**

## I. Introduction

TODAY, the Dial-a-Ride Problems are used in transportation services for elderly or disabled people. Also, the recent evolution in the transport field such as connected cars, autonomous transportation, and the emergence of the shared service might need to use this type of problem at much larger scales. But this type of transport is expensive and the management of the vehicles requires as much efficiency as possible, however the number of requests included in the vehicles planning can vary depending on the resolution used.

In [1] we solve the DARP by using constraint propagation in a greedy insertion heuristic. This technique obtains good results, especially in a reactive context, and is easily adaptable to a dynamic context. But, each demand is inserted one after another and the process doesn't take into account the impact of each insertion on the other not inserted demands, and so, in a dynamic context, the future demands. In this work, we present a measure of an insertion capacity named *Insertability*. We introduce its calculation by integrating the impact of an insertion on the time constraints (time windows, maximum route time and maximum ride time).

This measure may be used in different ways: selection of the demand to insert, selection of the insertion parameters, and exclusion of a demand. These three uses may be related to static as well as dynamic contexts by anticipating the future demands. The goal is to insert the current demand in order to build flexible routes for the future ones.

This paper is organized in the following manner: after a literature review, the next section will propose a model of the classic DARP. Then, we will review how to handle

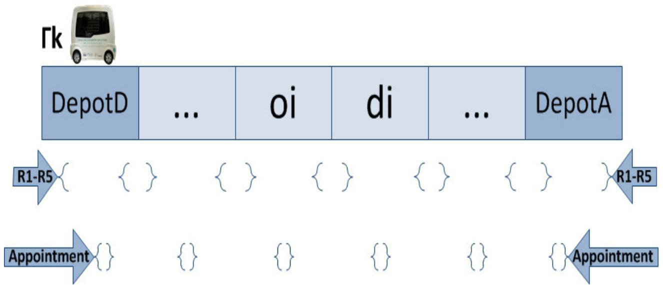

Fig. 1. Times windows' contraction

the temporal constraints with a heuristic solution based on insertion techniques using propagation constraints. We will continue by explaining the way to measure the *Insertability*, a calculation based on the evolution of the time windows after an insertion. Then, we will give some uses of this measure including making an appointment which minimize the time windows (cf. Figure 1). In the last part of the paper, the computational results will show the efficiency of our *Insertability*'s measure and we will report the evolution of the number of demands inserted in a resolution of some instances' sets.

## II. Literature review

The first works of the transportation optimization problem are related to the Traveling Salesman Problem ([2]). Since that time, other transportation problems have emerged as the vehicle routing and scheduling problems, and the Pick-up and Delivery Problem (PDP). The PDP is the ancestor of the problem of the Dial-a-ride problem which has been studied since the 1970's. DARP can be modeled in different ways. There are a number of integer linear programmings [3], but the problem complexity is too high to use it in a real context, most of which are NP-Hard because it also generalizes the Traveling Salesman Problem with Time Windows (TSPTW). Therefore, the problem must be handled through heuristic techniques. [4] is an important work on the subject and uses the Tabu search to solve it. Other techniques work well like dynamic programming (e.g. [5] and [6]) or variable neighborhood searches (VNS) (e.g. [7] and [8]). Moreover, a basic feature 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 contexts, which means synchronization mechanisms, interactions between the users and the vehicles, and uncertainty about for-coming demands. [9], and [10] later, developed the most used technique in dynamic context or in a real exploitation is heuristics based on insertion techniques. These techniques are a good solution when the people's requests have to be taken into account in a short period of time.

## III. The Dial-a-Ride Problem: model and insertion greedy algorithm

### A. The general notations

This section lets to set notations used throughout this document. For any sequence (or list) $\Gamma_k$ we set:

- for any z in $\Gamma_k$ :
  - Succ($\Gamma_k$, z) = Successor of z in $\Gamma_k$ ;
  - Pred($\Gamma_k$, z) = Predecessor of z in $\Gamma_k$ ;
- for any z, z' in $\Gamma_k$ :
  - $z \ll_k z'$ if z is located before z' in $\Gamma_k$ ;
  - $z \ll_k^= z'$ if $z \ll_k z'$ or z = z'.

### B. The model

A Dial a Ride Problem instance is defined by a *Demand* set $D = (D_i, i \in I)$, a fleet of *K* vehicles with a common capacity $CAP$, and a transit network $G = (V, E)$. *V* contains some specific node *Depot* and demands' nodes (DepotD for the departure and DepotA for the arrival). Each arc $e \in E$ is endowed with riding times give by a distance function $DIST(e)$. Each demand includes $o_i$ an *origin* node, $d_i$ a destination node, $F(o_i)$ and $F(d_i)$ two time windows, $\Delta_i$ a maximum ride time and $Q_i$ a description of the load such that $Q_i = q_{o_i} = -q_{d_i}$ with *q* the load related to a node. Finally, the total time of the *K* vehicles planning are limited by $\Delta^k, k \in K$.

Solving a DARP with such an instance means creating a scheduling for each vehicle handling demands of D. The routes are constructed while optimizing a performance, which could be a mix of costs (e.g. total distance) and QoS criteria (e.g. ride time).

### C. A greedy insertion algorithm: the insertion mechanism

In [1], we present an insertion greedy algorithm based on constraint propagation in order to contract time windows according to the time constraints. An insertion which does not imply constraint violation is said *valid* if $\Gamma = \cup_{k \in K} \Gamma_k$, the resultant collection of routes, if *load-valid* and *time-valid*. A route is *load-valid* if the capacity is not exceed, so, the *load-validity* is obtained if $ChT_k(x) \leq CAP$ with $ChT_k(x) = \sum_{y \ll_k^= x} q_y$, x and y nodes in the route *k*. The *time-validity* is obtained if there is no violation of the time constraints modeling by, for each demand *i*, $i \in D$, $\Delta_i$ the maximum ride time, $\Delta^k, k \in K$ the maximum route time and the constraints modeled by each time window $F(o_i) = [F.min(o_i), F.max(o_i)]$ and $F(d_i) = [F.min(d_i), F.max(d_i)]$. Checking the *load-validity* on $\Gamma = \cup_{k \in K} \Gamma_k$ is easy, and we show the efficiency of the constraint propagation in order to prove to *time-validity* after each planned insertion once the *load-validity* is proved. According to a current time window set *FP* = $\{$*FP(x)* = *[FP.min(x), FP.max(x)]*, $x \in \Gamma_k, k = 1..K\}$ the *time-validity* may be performed through propagation of the five following inference rules **Ri**, i = 1..5 in a given route $\Gamma_k$:

for each *(x,y)* pair of nodes such that y is the successor of x:

- **R1** : $FP.min(x) + DIST(x, y) > FP.min(y) \models (FP.min(y) \leftarrow FP.min(x) + DIST(x, y))$,
- **R2** : $FP.max(y) - DIST(x, y) < FP.max(x) \models (FP.max(x) \leftarrow FP.max(y) - DIST(x, y))$ ;

for each *(x,y)* pair of nodes such that both are related to the same demand, one is the *origin* so the other the *destination* :

- **R3** : $FP.min(x) < FP.min(y) - \Delta(x) \models (FP.min(x) \leftarrow FP.min(y) - \Delta(x))$,
- **R4** : $FP.max(y) > FP.max(x) + \Delta(x) \models (FP.max(y) \leftarrow FP.max(x) + \Delta(x))$ ;

and for each x, $x \in \Gamma_k, k = 1..K$ :

- **R5** : $FP.min(x) > FP.max(x) \models REJET \leftarrow true.$

These 5 rules are propagated in a loop while there no time windows exists *FP* modified at the last iteration. The tour $\Gamma_k$, $k = 1..K$, is *time-valid* according to the input time window set *FP* if and only if the *REJET* Boolean value is equal to *false* as initialized at the beginning of the process. In such a case, any *valid- time* value set *t* related to $\Gamma_k$ and *FP* is such that: for any *x* in $\Gamma_k$, *t(x)* is the appointment's date in *FP(x)*.

The greedy insertion algorithm includes this propagation constraint technique in order to evaluate each possible insertion. Each iteration of the algorithm selects one demand according to the number of vehicle able to integrate it. Once a demand is selected, the process chooses the insertion's parameters that are the vehicle and the location of the *origin* and *destination* nodes.

## IV. *Insertability* optimization

### A. State of the system

In the above algorithm, each iteration selects a demand, and then, it finds the way to insert while optimizing the performance. This greedy algorithm doesn't take in account the impact of this actual insertion on the future demands integration, but only the effect on the demands already inserted. In this section, we introduce a *Insertability* calculation by integrating this impact of an insertion related to the time constraints (time windows, maximum ride time and maximum route time).

During the insertion process, the state of the system is given by:

- a set of demands $D - D1$ already integrated in the routes, and $D1$ is the set of demands not inserted,
- a collection $\Gamma = \cup_{k\in K}\Gamma_k$ of routes including a list of nodes related to the *Depot*, *origin* and *destination* nodes,
- a exhaustive list of insertion's parameters sets. Each set gathers 5 elements : *k* the vehicle, *i* the demand, $(x, y)$ the pair of insertion nodes (locating respectively $o_i$ between x and the successor of x, and $d_i$ between y and the successor of y), and *v* the evolution of the collection $\Gamma = \cup_{k\in K}\Gamma_k$ 's cost.

## *B. Insertion's parameters*

Given that the difficulty of the instances' problem is linked to the time constraints, we introduce an *Insertability* calculation related to the times windows contractions. During an insertion's assessment, these reductions appear once the inference rules are propagated. Here, we try to find a good triple $(k, x, y)$, the vehicle and the location of the *origin/destination* nodes, in order to give enough space to the future demands (which have to be integrated in $\Gamma = \cup_{k\in K}\Gamma_k$).

We set *INSER(i, Γ )* the *Insertability* measure of the demand *I*. The quantity $U^k_n(z)$ denotes the vehicle *k* time windows' amplitude of the node *n* once it has been inserted to the right of node z. *INSER* is calculated as follows:

- $INSER(i,\Gamma) = \sum_{k\in K} INSER1(i,\Gamma_k)$ ;
- $INSER1(i,\gamma) = Max_{(x,y)} INSER2(i,\gamma,x,y)$, $\gamma$ a tour of $\Gamma$ ;
- $INSER2(i,\gamma,x,y) = U^{\gamma}_{o_i}(x).U^{\gamma}_{d_i}(y)$.

*INSER1* gives us the maximum of the product of the time's windows amplitude at the origin $i$ and destination $i$ over the possible insertion positions $x$ and $y$ in the route $\gamma$. When *INSER1* is equal to 0, the new route $\gamma$ resulting to the new insertion isn't *time-valid.*

We set $Inserted(\Gamma, i_0, k, x, y)$ the updated collection of tours Γ with the insertion of the selected demand $i_0$ at the locations *x* and *y* in the vehicle k. The *INSER(i,* Γ) measure allows us to write the *Optimization Insertability Problem* which consists to find the best insertion parameters in order to keep the vehicles' scheduling more flexible:

***Optimization Insertability Problem.*** Find the optimal parameters *(k,x,y)* inserting $i_0$ and maximizing the value $Min_{i\in D1-i_0} INSER(i, Inserted(\Gamma, i_0, k, x, y))$ .

For instance, the value $Min_{i\in D1-i_0} INSER(i, Inserted(\Gamma, i_0, k, x, y))$ may be used if all the demands have to be inserted. Another optimization may be process as the maximization of the sum $\sum_{i\in D1-i_0} INSER(i, Inserted(\Gamma, i_0, k, x, y))$. The choice is made according to the homogeneity of the demands and if the problem requires to insert all the set *D*.

This problem only optimizes the variation of the *Insertability* values and doesn't include other performance criteria like the minimization of the ride times, waiting times or distances. The *Insertability* criterion can be integrated in a mix of economical cost (point of view of the fleet manager) and of QoS criteria (point of view of the users). Then, the process maximizes the function $Perf = \mu.\sum_{i\in D1-i_0} INSER(i, Inserted(\Gamma, i_0, k, x, y)) - v(Inserted(\Gamma, i_0, k, x, y))$ with $\mu$ a criterion coefficient and *v* the performance value function mixing the costs related to the both points of view.

## *C. Other uses of the Insertability measure*

So far, we select the demand $i_0$ according to the number of vehicles available (taking in account all the time and load constraints). The *Insertability* measure *INSER(*$i_0$, Γ*)* may be also used in order to select the next request $i_1$ to insert. This application could be used in a context where all the demands of $D$ have to be integrated. The selection is based on the smallest *Insertability* measure. Once a demand is selected, the problem may solve the *Optimization Insertability Problem.* Here, the two steps may be written in a non-deterministic way. The demand may be selected randomly through a set of N1 elements with the smallest *INSER* value. The same scheme may be applied on a set of a insertion parameters of N2 elements with a best $(k, x, y)$ elements maximizing the quantity $Min_{i\in D1-i_0} INSER(i, Inserted(\Gamma, i_0, k, x, y))$.

Also, *INSER(*$i_0$, Γ*)* may be useful for a larger set *D*. If the instance doesn't have any solution integrating all the set *D*, it is preferable to identify requests to exclude as soon as possible. The exclusion of a demand $i_0$ may be set up if its insertion results in Γ not enough flexible to include the other elements of *D1*. In other words, the demands excluded will be those that will have the most impact of future insertions. The difference $\sum_{i\in D1-i_0}(INSER(i,\Gamma) - INSER(i, Inserted(\Gamma, i_0, k, x, y)))$ of the inequality (1) takes in account the *Inserabilty* measure of $D1 - i_0$ before and after the insertion of $i_0$ in the routes of Γ. If this difference is larger than the threshold $\xi$, the demand is excluded. In the experimentation' section, we will discuss the fact this threshold should be dynamic and decreases over the execution.

$$\sum_{i\in D1-i_0} (INSER(i,\Gamma) - INSER(i, Inserted(\Gamma, i_0, k, x, y))) > \xi \qquad (1)$$

### D. The Insertability optimization suited to the greedy insertion algorithm

The calculation of *INSER(i*, Γ*)*, $i \in D$, begins to be time consuming starting from a medium size of *D* once the *INSER2* value is based on the time windows' amplitude obtained after the propagation of the time constraints. So, this is important to spot each step of the process where the *Insertability* measure doesn't have to be updated. When $i_0$ is selected, $INSER2(i, \Gamma_k, x, y)$, $INSER1(i, \Gamma_k)$ and $INSER(i, \Gamma)$ are known for all demand in $D1 - i_0$ and all $k = 1..K$. Once $i_0$ is about to be inserted, the process computed the value $H(i), i \in D1 - i_0$ (cf. formulation (2)). Then, the algorithm tries the insertion of each *i* from $D1 - i_0$ in $Inserted(\Gamma, i_0, k, x, y)$ and deduce the value *K(i)* given in formula (3) for all $i \in D1 - i_0$ and ultimately the quantity $Val(k, x, y) = Min_{i \in D1 - i_0}(K(i) + H(i))$.

$$H(i) = INSER(i, \Gamma) - INSER1(i, \Gamma_k) \tag{2}$$

$$\begin{aligned} K(i) &= INSER(i, Inserted(\Gamma, i_0, k, x, y)) \\ &= H(i) + INSER1(i, Inserted(\Gamma, i_0, k, x, y)_k) \end{aligned} \tag{3}$$

Other calculations may be avoided. We set $W_1$ such that $W_1 = Min_{i \in D1 i_0} INSER\ (i, \Gamma)$. If the quantity $INSER(i, \Gamma) - INSER1(i, \Gamma_k)$ is larger than $W_1$, there is no need to test the impact of the insertion of $i_0$ on *i*.

Finally, we're able to use *INSER(i*, Γ*)* once we integrate the future demands presented in the next section. In a dynamic context, the *Insertability* measure helps the routes to be enough flexible for the next insertion process. Moreover, the appointments have to be set with the same purpose and *INSER(i*, Γ*)* is able to help to do it.

## V. Introduction to the robustness in the DARP: Anticipation of the future demands and *Insertability* measure integration

The problem may have to be handled according to a *dynamic* context and the greedy insertion algorithm is easily adaptable to this context. Once the *Insertability* measure is included in the performance criteria, the system may increase its robustness. In order to accomplish this, we need to exploit knowledge about future demands. In our case, this knowledge is related to the type of on demand transportation service. In this paper, we will use a simple extrapolation of this probable demand based on the demand already broadcasted.

We won't take into account the way the system supervises its various communication components with the users. In reality, there are eventual divergences between the data which were used during the planning phases and the situation of the system.

We set $D-V$ the virtual demands, $D-R$ the real demands, and $D - Rejet$ the set of the ones excluded from the insertion algorithm such that $D - Rejet = DV - Rejet \cup DR - Rejet$. The $D-V$ formulation is given in (4). $p_i$ gives us the number of times the demand $D_i$, $i \in D$, will appear for each period of each discrete planning horizon.

$$D - V = \sum_{i \in D} D_i . p_i \tag{4}$$

Then, we're able to update the formula (5) giving the performance function *Perf.*

$$\begin{aligned} Perf = \alpha . \sum_i p_i INSER(i, Inserted(\Gamma, i_0, k, x, y)) \\ + \mu . \sum_{i \in D1 - i_0} INSER(i, Inserted(\Gamma, i_0, k, x, y)) \\ - v(Inserted(\Gamma, i_0, k, x, y)) \end{aligned} \tag{5}$$

As in the previous sections, the process may exclude some demands taking in account the future requests. We updated the inequality (1) by the (6). $\alpha$ is a coefficient based on the importance of the future demands.

$$\begin{aligned} \alpha . \sum_i p_i . (INSER(i, \Gamma) \\ - INSER(i, Inserted(\Gamma, i_0, k, x, y))) \\ + \sum_{i \in D1 - i_0} (INSER(i, \Gamma) \\ - INSER(i, Inserted(\Gamma, i_0, k, x, y))) > \xi \end{aligned} \tag{6}$$

## VI. Discussion about the appointments and the dynamic context

Most works on vehicle scheduling problems including time window studies how to integrate a set of demands in the vehicle planning. Making an appointment anticipating the future is especially rare. Previous sections explained how to select and integrate user's request while keeping enough space for the next set of demands.

Once routes are built and integrated a first set *D*, the users expect the date when the vehicle selected will pick them up. In the lists forming the *K* routes, each node has a time window. After the appointment's date is set, each time window becomes tight with zero amplitude or equals a very small delay. How the appointments' dates are made is very important for the next insertion's process. For instance, we consider a fleet of 2 vehicles with two plannings including 5 demands while the distances are minimized (cf. Figure VI). The time windows are relatively wide so, while the distance traveled is minimized, the difference of each appointment's time between two nodes is the exact time to join them. The vehicle *k=2* from the Figure VI may integrated the node $o_7$ between its depot node and $o_5$ even if its time windows have a zero amplitude (the vehicle will only have to leave the depot earlier). On the other hand, if the difference on the appointment' times given to the users related to the nodes $d_5$ and $o_3$ equals to $DIST(d_5, o_3)$, the insertion of $d_7$ will be forbidden. In the same way, there will be a violation of some constraint once nodes $o_6$ and $d_6$ will be inserted in the vehicle $k = 1$.

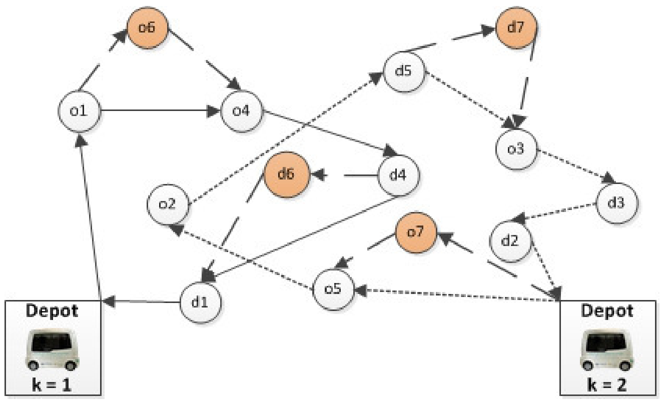

Fig. 2. New insertions after the first set of appointments

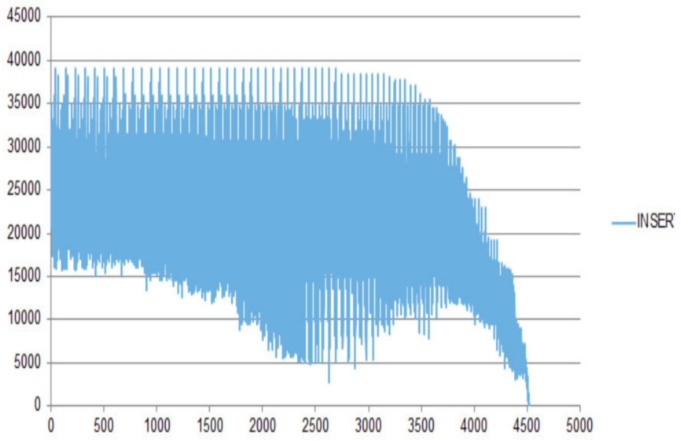

Fig. 3. INSER values on the not inserted demands

One more time, the *INSER(i, Γ)* values may be used in order to set the appointment dates without to have the problem above. The appointment's dates may be calculated once the process have inserted the virtual demands $D-V$ and the real demands $D-R$.

The previous section shows the way to anticipate the future demands $D-V$. These demands are related to a dynamic context. Note again that our greedy algorithm is easily adaptable to this context. More specifically, the technique doesn't change unlike the state of each route. The first node isn't a depot node anymore but a dynamic node related to the vehicle's location. The entire constraint propagation process is applied on these new routes. A simulation will be necessary to evaluate the anticipation of the future demands including in the dynamic context.

## VII. Computational Experiments

In this section, we study the behavior of our *Insertability* measure used in the resolution of Dial-a-Ride instances. The algorithms were implemented in C++ and compiled with GCC 4.2. In [1], we solve the [4]'s instances by our greedy insertion algorithm based on constraint propagation. We obtained good results in the majority of instances, but, only 1% of the replications gave us a feasible solution on the tenth instance (*R10a*). The CPU time was smallest or equal to the best times in the literature; we don't work on this feature for this experiment.

### A. First experimentation: the optimization of the selection of the demand to insert

*1) INSER used in the selection of a demand:* We note by $R^{DARP}$ the rate of 100 replications which give us a feasible solution obtained by using the solution of [1]. Here, the selection of the demand is based on the lowest number of cars which are able to accept it. $R^{DARP}_{Rob}$ is the rate obtained with the same process except that each demand is selected at each iteration by the lowest *Insertability* value *INSER*.

The *Insertability* measure is already efficient once it's used in the selection of the demands to insert. The rate obtained for the pr08, pr09, pr10 and pr19 are clearly more interesting as shown in table I (e.g. for the instance pr08, the rate increases by 56% to 91% of success).

| Inst. | $R^{DARP}$ | $R^{DARP}_{Rob}$ |
|---|---|---|
| pr01 | 99 | 100 |
| pr02 | 100 | 100 |
| pr03 | 97 | 100 |
| pr04 | 100 | 100 |
| pr05 | 100 | 100 |
| pr06 | 100 | 100 |
| pr07 | 90 | 96 |
| pr08 | 56 | **91** |
| pr09 | 18 | 21 |
| pr10 | 1 | **7** |
| pr11 | 100 | 100 |
| pr12 | 100 | 100 |
| pr13 | 99 | 100 |
| pr14 | 100 | 100 |
| pr15 | 100 | 100 |
| pr16 | 100 | 100 |
| pr17 | 98 | 100 |
| pr18 | 99 | 100 |
| pr19 | 64 | **99** |
| pr20 | 43 | 56 |
| Av. | 83,2 | 88.5 |

TABLE I
$R^{DARP}$ Vs $R^{DARPRob}$

*2) INSER behaviour:* Each time a replication can't integrate all the request, the *INSER* value of the demands not inserted has to be null. In Figure VII-A2, while the resolution process applied to the R10a instance, we note the evolution of more than 4500 *INSER*'s demands not inserted. The technique used is the second approach selecting the demand by the smallest *Insertability*. The values noted are from a failed replication.

One can observe big gaps between the different *INSER* until the 4000 first values. After that, for the remaining requests, the *Insertability* values decrease strongly because the routes begin to be not flexible. Between the $2500^{th}$ and the $3500^{th}$, for some demands, the values are very low at the beginning just before increasing strongly. This is explained by the fact the process inserts the demand with the lowest

*INSER* but their insertion don't make a big impact on the other demands not inserted. This impact is related to the *Optimization Insertability Problem* studied below.

## B. Second experimentation: the optimization of the insertion parameters

In a second experimentation, we compare the [1]'s approach and another algorithm based on the optimization of the parameters *(x,y,k)*. The selection of the request to insert is the same for both solutions. For the second one, once a demand $i_0$ is selected, we maximize the sum $\sum_{i \in D1-i_0} \mathit{INSER}(i, \mathit{Inserted}(\Gamma, i_0, k, x, y))$ in order the find the best parameter *(x,y,k)* which will integrate $i_0$ in the route $k$. We don't optimize $\mathit{Min}_{i \in D1-i_0} \mathit{INSER}(i, \mathit{Inserted}(\Gamma, i_0, k, x, y))$ because we create instances especially with a set *D* too large for inserting all the requests. So, the demand with the smallest value *INSER* for a given parameters *(x,y,k)* could never be integrated into the routes.

The two algorithms were applied to five sets of 5 randomly generated instances. All the instances have their time constraints related to the interval [0;400] and all the load was unit. We set by $e_{F(o)}$ and $e_{F(d)}$ the amplitude of the time windows at the *origin* and the *destination* given by the users, respectively. The other parameters are given in table II.

| K | $e_{F(o)}$ | $e_{F(d)}$ | $\Delta$ | $CAP$ |
|---|---|---|---|---|
| 10 | 35 | 10 | $\infty$ | 10 |

TABLE II
PARAMETERS' INSTANCES

We generate 5 different sets of 5 instances with a variation of the number of demands $|D|$. We set by $T_{Insert}$ and by $T_{Insert_{Rob}}$ the demand inserted's rate the first resolution and the second technique, respectively. Finally, $Gap_{Insert}$ is the gap in percentage between each rate. Its calculation is given by $Gap_{Insert} = 100.(T_{Insert_{Rob}} - T_{Insert})/T_{Insert}$. We launched 100 replications of each technique on the 5 sets. The results are provided by the table III.

| $\lvert D \rvert$ | 50 | 75 | 100 | 150 | 200 |
|---|---|---|---|---|---|
| $T_{Insert}$ | 100 | 93.2 | 78.9 | 64.2 | 52.6 |
| $T_{Insert_{Rob}}$ | 100 | 96.8 | 85.3 | 66.4 | 54.1 |
| $Gap_{Insert}$ | 0 | 3.86 | 8.11 | 3.43 | 2.81 |

TABLE III
GAP BETWEEN THE *INSERT* RATES

In future experiments, we need to optimize the value $\mathit{Perf} = \mu.\sum_{i \in D1-i_0} \mathit{INSER}\ (i, \mathit{Inserted}(\Gamma, i_0, k, x, y)) - v(\mathit{Inserted}(\Gamma, i_0, k, x, y))$ to calculate each best insertion parameters. Here, we're just taken into account the *INSER* values in order to integrate the most requests possible. The results show us that the larger of $|D|$ defines if the system needs to optimize the *Insertability* measure. For $|D| = 50$, all the requests are able to be inserted easily, so, the *INSER* values doesn't have any interest. When the set is composed of 100 demands, we obtained a $Gap_{Insert}$ of 8,11% meaning there are more than 8% more requests inserted by the second approach.

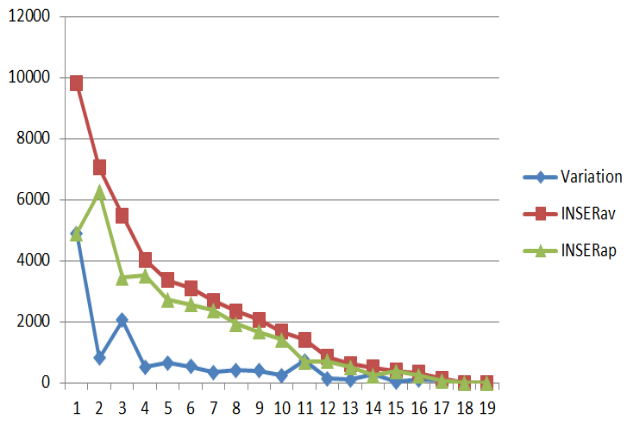

Fig. 4. Variation of the *Insertability* values between each insertion

For this set of instance, we also tried to integrate a new feature in our algorithm: we've added the ability to exclude a request if the impact of one insertion involving a significant drop of the general *Insertability's* demands from $D1 - i_0$. Before that, we study the threshold which limits the variation of *Insertability*.

We exclude a demand selected $i_0$ if $\sum_{i \in D1-i_0}(\mathit{INSER}(i, \Gamma) - \mathit{INSER}(i, \mathit{Inserted}(\Gamma, i_0, k, x, y))) > \xi$ is true with $\xi$ a threshold. The calculation of the threshold is a difficult problem. In the figure VII-B, we report the $\sum_{i \in D1-i_0}(\mathit{INSER}(i, \Gamma) - \mathit{INSER}(i, \mathit{Inserted}(\Gamma, i_0, k, x, y)))$ *Variation* with *INSERav* and *INSERap* the values $\sum_{i \in D1-i_0} \mathit{INSER}(i, \Gamma)$ and $\sum_{i \in D1-i_0} \mathit{INSER}(i, \mathit{Inserted}(\Gamma, i_0, k, x, y))$, respectively. This figure shows us that the threshold $\xi$ have to be calculated dynamically according to the average of *INSER*.

We used this type of dynamic threshold for the third set of instances with 100 demands. We exclude an request if the current $\xi$ is exceeded, and only this feature is added in the second approach. We obtained a gain of 1,3% in average (from 85,3% to 86,6%) meaning approximately one more demand is able to be inserted.

## VIII. CONCLUSION

The Dial-a-Ride Problem is one of the transport problems with the highest number of hard constraints like time windows. The insertion techniques are able to obtain a good solution in a reasonable time. Their adaptability to a dynamic context is easy but a lack of robustness could appear once the goal is to integrate requests as much as possible.

We have introduced a way to measure the impact of each insertion on the other demands not inserted. This *Insertability* measure could be used in order to exclude a demand, to select a demand to insert and also to calculate the best insertion parameters. This value, named *INSER*, leads to a large amount of work opportunities. We have introduced a simple way to make the model of the future demands, and how to adapt our greedy insertion algorithm based on the constraint propagation to the dynamic context. In future work, we will develop a simulation which is necessary to show the efficiency of the demands anticipation. The final goal will be to develop the most robust algorithm possible in order to adapt it to a real context.

## Acknowledgment

This work was founded by the French National Research Agency, the European Commission (Feder funds) and the Region Auvergne in the Framework of the LabEx IMobS3.

## References

[1] S. Deleplanque, A. Quilliot, Constraint Propagation for the Dial-a-Ride Problem with Split Loads, 2013, Recent Advances in Computational Optimization. Studies in Computational Intelligence, Vol. 470. ISBN 978-3-319-00409-9, Volume 470, 2013, Springer, 31-50.

[2] K. Menger, Das botenproblem, 1932, Ergebnisse eines mathematischenkolloquiums 2, 11-12.

[3] J.F. Cordeau, G. Laporte, The dial-a-ride problem: models and algorithms, 2007, Annals of Operations Research,153(1):29-46.

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

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

[6] R. Chevrier, P. Canalda, P. Chatonnay, D. Josselin, Comparison of three algorithms for solving the convergent demand responsive transportation problem, 2006, Intelligent Transportation Systems, Toronto, Canada, 1096-1101.

[7] S.N. Parragh, K.F. Doerner, R.F. Hartl, Variable neighborhood search for the dial-a-ride problem, 2010, Computers & Operations Research, 37, 1129-1138.

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

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

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