7th IFAC Conference on Manufacturing Modelling, Management, and Control
International Federation of Automatic Control
June 19-21, 2013. Saint Petersburg, Russia

Dial-a-Ride Problem with time windows, transshipments, and dynamic transfer points

Samuel Deleplanque*. Alain Quilliot*

*LIMOS, UMR CNRS 6158, Labex IMOBS3,
BLAISE PASCAL University, France (e-mail: {deleplan, quilliot}@isima.fr)

Abstract: Dial-a-ride problems deal with on demand transportation. Today, this type of shared transport is used for elderly and disabled people for short distances. People provide a request for transport from an origin to a destination, both specific for a particular demand. Time constraints deal with maximum user ride time, time windows, maximum route duration limits and precedence. In this paper we try to solve these dial-a-ride problems, including the possibility of one transshipment from a dynamic transfer point by request. We propose an algorithm based on insertion techniques and constraints propagation.

Keywords: Dial-a-Ride Problem, transshipments, transfer points, constraints propagation.

1. INTRODUCTION

In the Dial-a-Ride Problems (DARP) people can order a ride defining mobility demands, giving a pick-up location, a delivery location, two time windows, an upper bound on the duration of the demand, and the load related to the demand. The optimization consists of creating the route of a fleet of vehicles in order to satisfy all (or the most possible) demands. To determine these routes, one has to find a balance between two antinomic things: the quality of service and minimization of the total cost.

The majority of the Dial-a-Ride problem uses are related to the elderly and disabled people, but the latest research in transportation (connected cars, autonomous cars, etc.) could provide vehicles for the optimization problem.

This work integrates the possibility to make a transshipment between two vehicles in order to satisfy a request. This transshipment is done by a dynamic transfer point; it means this point is computed at the same time as the demand is included in a vehicle planning.

DARP can be modeled in different ways. Many integer linear programming exist (refer to Laporte et al. (2007)), but the problem complexity is too high to use, most of which are NP-Hard. So, the problem must be handled through heuristic techniques. Cordeau et al. (2003) is one of the most important works on the subject and uses the Tabu search to solve it. Others techniques work well like dynamic programming (e.g. Psaraftis (1983) and Chevrier et al. (2006)) or variable neighborhood searches (VNS) (e.g. Parragh et al. (2012) and Healy et al. (1995)). Psaraftis et al. (1986), and Madsen et al. (1995) 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 take into account in a short time.

The main contribution of the paper is to allow transfer in the DARP. These transshipments are made dynamically and can be located everywhere. Little has been published on this subject, only Masson et al. (2011, 2012) studied the problem. They express the problem by DARPT. The location of their transshipments points are known before the resolution. The authors use Tabu research, minimizing the total distance travelled by the fleet of vehicle. The closest problem is the Pickup and Delivery Problem with Transfers (PDPT).

There exist some exact methods to solve the PDPT, like Kerivin et al. (2008), where the transfer points are those shaped by origin and destination nodes. Cortes et al. (2010) and Nakao et al. (2012) used a Branch-and-cut algorithm. As stated previously, the exact methods are not a good solution for solving the problem because it can't be used in a reactive context. The approximate methods are able to solve the problem in time. Cortes et al. (2002) created several rules for selecting the vehicle, assuming a demand given. Other rules help to trace the routes. The VNS is also used for this problem, Masson et al. (2012) tested it on big instances (almost 200 requests). Shang et al. (1996) worked on a heuristic where each pick-up and delivery nodes could be a transfer point. Their solution is based on insertion techniques and they allow the transshipments if the demand could not be inserted; Thangiah et al. (2007) included the same algorithm in a dynamic context.

This paper is organized in the following manner: The next section will present a model of the classic DARP, and a heuristic solution based on insertion techniques using propagation constraints. The third section will continue with the introduction of the DARPT model, framework and our updated solution. In the last part of the paper, we will compare the two solutions on the same instances.

2. The DARP: model and framework

2.1 Notations

For any sequence (or list) Γ whose elements belong to some set Z, 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:
    - $z <<_\Gamma z'$ if z is located before z' in Γ;
    - $z <<_\Gamma^= z'$ if $z <<_\Gamma z'$ or z = z'.

In any algorithmic description, we use the symbol <- in order to denote the value assignment operator and the symbol = as a comparator.

2.2 A DARP instance

A DARP instance is essentially defined on a transit network G = (V, E) by:

- a fleet $\mathcal{VH}$ of K homogeneous vehicles with a common capacity $\mathcal{CAP}$ and provided by some specific node Depot,
- a *Demand* set $\mathcal{D} = (\mathcal{D}_i, i \in I)$, any demand $\mathcal{D}_i$ defined as a 6-uple $\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i)$, where:
    - $o_i \in V$ is the *origin* node of the demand $\mathcal{D}_i$;
    - $d_i \in V$ is the *destination* node of the demand $\mathcal{D}_i$;
    - $\Delta_i \geq 0$ is an upper bound (*transit bound*) on the duration of demand $\mathcal{D}_i$'s processing;
    - $\mathcal{F}(o_i)$ is a time window related to the time $\mathcal{D}_i$ starts being processed;
    - $\mathcal{F}(d_i)$ is a time window related to the time $\mathcal{D}_i$ ends being processed;
    - Q_i is a description of the load related to $\mathcal{D}_i$.

We noted by *DepotD* (resp. *DepotA*) the first node of a route (resp. the last node) and by DIST the distance between two nodes computed by the triangle inequality. A route is defined by a list of demand's nodes surrounded by *DepotD* and *DepotA*. Moreover, we set a status $\mathcal{Status}$(x): *Origin*, *Destination*, *DepotA*, *Depot D*; and *Depot* = *DepotD* $\cup$ *DepotA*. We define the function twin(x) such that if $x = o_i$ then $\mathcal{Twin}(x) = d_i$ and conversely. The two kinds of depot nodes are twin. Finally, we denote the node set X = {*DepotD(k), DepotA(k)*, k = 1..K} $\cup$ $\{o_i, d_i, i \in I\}$ and the operator INSERT such that 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).

2.3 Valid tours

The validity on a tour Γ is obtained when the load and time constraints are respected. The first is easy: if $\mathcal{C}(\Gamma, x)$ is the total load of the vehicle leaving the node x, the load validity is obtained if $\mathcal{C}(\Gamma, x) \leq \mathcal{CAP}$ for any x in Γ. The time validity is more complicated to check and may be performed through propagation of the following inference rules R_i, i = 1..5 according to a current time window set $\mathcal{FS} = \{\mathcal{FS}(x) = [\mathcal{FS}.\min(x), \mathcal{FS}.\max(x)], x \in \Gamma\}$:

- Rule R_1: y = Succ(Γ, x); $\mathcal{FS}.\min(x) + \mathrm{DIST}(x, y) > \mathcal{FS}.\min(y) \models \mathcal{FS}.\min(y) \leftarrow \mathcal{FS}.\min(x) + \mathrm{DIST}(x, y)$;
- Rule R_2: y = Succ(Γ, x); $\mathcal{FS}.\max(y) - \mathrm{DIST}(x, y) < \mathcal{FS}.\max(x) \models \mathcal{FS}.\max(x) \leftarrow \mathcal{FS}.\max(y) - \mathrm{DIST}(x, y)$;
- Rule R_3: y = Twin(x); $x <<_\Gamma y$; $\mathcal{FS}.\min(x) < \mathcal{FS}.\min(y) - \Delta_{i(x)} \models \mathcal{FS}.\min(x) \leftarrow \mathcal{FS}.\min(y) - \Delta_{i(x)}$;
- Rule R_4: y = Twin(x); $x <<_\Gamma y$; $\mathcal{FS}.\max(y) > \mathcal{FS}.\max(x) + \Delta_{i(x)} \models \mathcal{FS}.\max(y) \leftarrow \mathcal{FS}.\max(x) + \Delta_{i(x)}$;
- Rule R_5: $x \in \Gamma$; $\mathcal{FS}.\min(x) > \mathcal{FS}.\max(x) \models \mathcal{Fail}$.

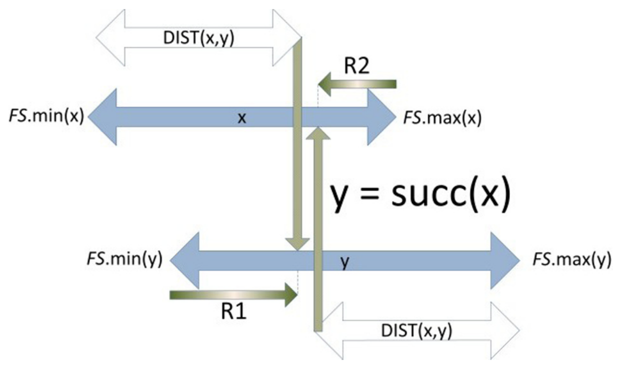

Fig. 1 The inference rules on two successive nodes

The propagation of these rules have to be performed for all the pairs of nodes (x,y). The figure 1 shows how R1 and R2 work on two successive nodes. If the propagation doesn't return $\mathcal{Fail}$, it means there aren't any time constraints violation, with every time set value *t* such that*:* for any $x \in \Gamma$, $t(x) \in \mathrm{FS}(x)$, refer to Deleplanque et al., 2012 for the proof.

A collection tour T = {T(k), k = 1..K} with K valid tours is a feasible solution for the DARP.

2.4 Evaluation of a route

It exists many different ways to evaluate the performance of a set of tours. It's often a balance between quality of service and a total cost. For any pair (Γ, t) defined by some time-valid tour Γ and by some valid related time value set t, we may set:

- $\mathcal{Glob}(\Gamma, t) = t(\mathrm{End}(\Gamma)) - t(\mathrm{First}(\Gamma))$: this quantity denotes the global duration of the tour Γ;
- $\mathcal{Ride}(\Gamma, t) = \Sigma_{x \backslash \mathcal{Status}(x) = Destination}\, t(x) - \Sigma_{x \backslash \mathcal{Status}(x) = Origin}\, t(x)$; 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;

If A and B are two multi-criterion coefficient, we may define the performance criterion $\mathrm{Cost}_{A, B}(\Gamma, t)$ as follows: $\mathrm{Cost}_{A, B}(\Gamma, t) = A.\mathcal{Glob}(\Gamma, t) + B.\mathcal{Ride}(\Gamma, t)$.

2.4 The insertion process

This process takes as input the demand set $\mathcal{D} = (\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i), i \in I)$, the 4-uple (X, DIST, K, $\mathcal{CAP}$), 2 multi-criterion coefficients A and B ≥ 0, and it works in a greedy way through successive insertions of the various demands $\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i)$ of the demand set $\mathcal{D}$. 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 $\mathcal{FP}$(T(k))(x), x $\in$ T(k), as well as the load values $\mathcal{C}$(T(k), x), x $\in$ T(k), and the values Cost(T(k),t);
- the knowledge, for any i in J = (I - I_1) of the set FREE(i) of all the 4-uple (k, x, y, v), k = 1..K, x, y $\in$ T(k), v $\in$ $\mathcal{Q}$, such that the origin (resp. destination) could be inserted after the node x (resp. after the node y). We denote by N-FREE(i) the cardinality of the set V-FREE(i) = {k = 1..K, such that there exists a 4-uple (k, x, y, v) in FREE(i)}: N-FREE(i) provides us with the number of vehicles which are still able to deal with demand $\mathcal{D}_i$.

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

1. 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) and Card(Free(i)) are small.
2. It picks up (k_0, x_0, y_0, v_0) in FREE(i_0) which corresponds to one of the smallest values v.
3. It inserts the demand $\mathcal{D}_{i0}$ 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);
4. It updates FREE(i) and N-FREE(i) consequently.

We denote by $\mathcal{Reject}$ the rejected demand set. The INSERTION algorithm may be used inside some MONTE-CARLO framework:

```
RANDOM-INSERTION(P: Integer) Scheme;
For p = 1..P do
    Apply the INSERTION() 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 $Cost_{A, B}$(T, t) value).
```

3. Dial-a-Ride problem with transfers

3.1 Model and framework

We are going to deal now with the case when transfers are allowed, that means then the load Q_i related to some demand $\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i)$, may be handled in several successive steps, each step involving some vehicle k $\in$ K, which make the charge Q_i go from some origin or relay node x to some relay or destination node y. Transfers means here that, at any time during the transportation, while the load Q_i is always handled as a whole, the route it follows may be split into several sub-route, all those sub-routes being taken in charge by distinct vehicles. As a matter of fact, we are going to restrict here ourselves to the case when no more than 2 vehicles are allowed to perform such a transportation task: though this restriction is not going to induce any true restriction on concepts and methods, it will help us in describing them; also, practical applications are such that they will hardly allow a same demand to be handled by more than 2 or 3 different vehicles.

In order to put this in a formal way, we first need to extend X in order to make appear the relay nodes. Since we want to handle those relay node in a dynamic way, we suppose that X may be embedded into some (eventually infinite) implicit node set Z such that X $\subset$ Z. Additional nodes in Z – X are going to be used as relay nodes. Any such an active relay node will appear in 2 tours, once as an emitter node and once as a receiver node. Since we would like to keep on with the kind of model which we have been using for the standard version of the Dial-a-Ride problem, we also would like to make in such a way that all nodes which are going to appear in a tour family T be distinct. In order to do it, we define the implicit node set Z* as follows:

- $Z^* = X \cup \{(z, i, -1), (z, i, 1), i \in I, z \in Z\}$: the meaning of node (z, i, -1) is that if such a node becomes active, then it will appear as emitter node for load Q_i inside some tour T(k), which means that load Q_i is first going transported from o_i to z by vehicle k, and next from z to d_i by some other vehicle k', k $\neq$ k'. It comes that node (z, i, 1) will appear in a symmetric way in tour T(k');
- for any node z in Z, we set: $\mathcal{Node}$(z, i, -1) = $\mathcal{Node}$(z, i, +1) = z;
- for any node x in X we set: $\mathcal{Node}$(x) = x.

We extend the $\mathcal{Status}$, $\mathcal{Twin}$, Δ, and $\mathcal{F}$ functions by setting, for every z $\in$ Z, i $\in$ I:

- $\mathcal{Status}$(z, i, -1) = *Out-Reload*, $\mathcal{Status}$(z, i, +1) = *In-Reload*;
- $\mathcal{Twin}$(z, i, +1) = {(z, i, - 1), $\mathcal{Twin}$(z, i, -1) = {(z, i, + 1);
- Δ(z, i, -1) = Δ(z, i, +1) = + ∞;
- $\mathcal{F}$(z, i, -1) = $\mathcal{F}$(z, i, +1) = [0, + ∞[.

We are going to use two insertion operators:

- the INSERT operator defined in the section 2.2,
- the INSERT2 operator works by inserting demand $\mathcal{D}_i$ into two distinct tours: k, k' being two distinct vehicle indices, x, y being two nodes in T(k) such that $x <<^{=}_{T(k)} y$, x', y' being two nodes in T(k') such that $x' <<^{=}_{T(k')} y'$, z being some relay node in Z, INSERT2(i, k, k', x, y, x', y' , z) denotes a pair of

tours (INSERT2(i, k, k', x, y, x', y' , z).First, INSERT2(i, k, k', x, y, x', y' , z).Second) in such a way that:

- INSERT2(i, k, k', x, y, x', y' , z).First is the tour which is obtained through insertion of o_i between x and Succ(T(k), x) in T(k) and by insertion of (z, i, -1) between y and Succ(T(k), y) in T(k);
- INSERT2(i, k, k', x, y, x', y' , z).Second is the tour which is obtained through insertion of (z, i, 1) between x' and Succ(T(k'), x') in T(k') and by insertion of d_i between y' and Succ(T(k), y') in T(k').

3.2 Validity of collection tour with transfers

A collection tour T = {T(k), k = 1..K} is valid for the DARPT if:

- for every k = 1..K, the tour T(k) respects the load constraint (section 2.3),
- there exists some time value set t = {t(x), $x \in \cup_{k=1..K}$ T(k)} such that:
  - for any k = 1..K, the restriction of t to the nodes of T(k) defines a valid time value set related to T(k) (section 2.3)
  - for every *Out-Reload* node x, we have t(*Twin*(x)) ≥ t(x); (*Linking Constraints*)

  In case such a time value set t exists, we obtain a feasible solution of the DARPT.

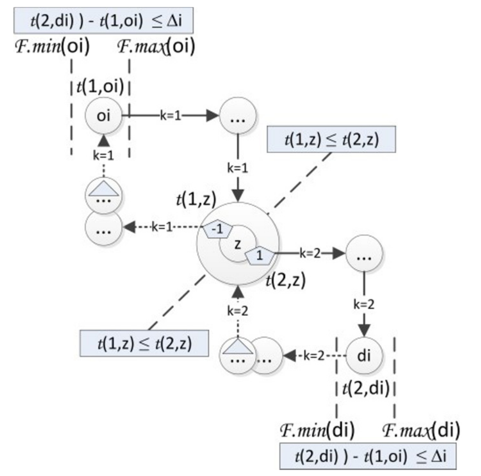

Fig. 2 Transshipments of the demand i between two vehicles (k1 & k2) - Linking Constraints

3.3 Handling the Relay Nodes, The ℒ-Candidate lists, and the FREE2 sets

For any i ∊ J, the set FREE2-o(i) will be made with the pair (k, z) such that:

- the active node z is in T(k), different from Last(T(z));
  - [a(z), b(z)] denotes the time window *FP*(T)(z);
  - z' denotes Succ(T(k), z);
  - [α, β] denotes the time window *F*(o_i);
- $(a(z) + DIST(z, o_i) \le \beta) \wedge (\alpha + DIST(o_i, z') \le b(z')) \wedge (a(z) + DIST(z, o_i) + DIST(o_i, z') \le b(z')) \wedge (\mathcal{C}(\Gamma, z) + Q_i \le \mathcal{CAP})$.

For any i ∊ J, the set FREE2-d(i) will be made with same process than FREE2-o(i). So, for any (x_1, k_1) in FREE2-o(i_0), and for any (y_2, k_2) in FREE2-d(i_0), $k_1 \neq k_2$, we compute a relay node z through the following process:

1. we are provided with a function *Midst*, which, from any pair of nodes z, z' in the node set Z, compute a new node z" = *Midst*(z, z') in Z, in such a way that *Dist*(z, z") and *Dist*(z", z) are no larger than some fraction λ.*Dist*(z, z'), with λ < 1;
2. for any node y in T(k_1), we denote by *Close*(y, k_2) the first (in the sense of the relation $<<_{T(k2)}$) element x in T(k_1) such that t(x) ≥ t(y).
3. then we apply the following *Exchange* function:

```
Exchange(x1, k1, y2, k2): (y : node in T(k1),
x : node in T(k2), z : relay node)
Compute y in T(k1), such that :
    $x_1 <<_{T(k)} z$ and Close(y, k2) $<<^{=}_{T(k2)} y_2$;
    DIST(y, Close(y, k2) is the smallest
    possible ;
If y is undefined then Exchange <- Undefined
Else Exchange <- (y, Pred(Close(y, k2)),
Midst(z, U(z, l)));
```

4. the result which is produced by ***Exchange***(x_1, k_1, y_2, k_2) provides us with the parameters y_1, x_2, z, which would eventually allow us to perform the (T(k_1), T(k_2)) <- INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2 , z) instruction.

So, the *Weak-ℒ-Candidate* list will be defined by those 7-uple (k_1, k_2, x_1, y_1, x_2, y_2, z) which we obtain this way, and the *ℒ-Candidate* list will defined by those among those 7-uple which are such the validity of the tour collection T will be preserved through application of the INSERT2 operator, ordered according to some auxiliary performance criterion.

3.4 The insertion scheme

We notice that the two operators which we described above have quite different impacts on the way a global insertion schema is going to work. While testing the feasibility of an application of the INSERT operator is a local task, which only involve dealing with the T(k) tour, testing the feasibility of the INSERT2 operator is likely to involve more than the T(k), T(k') tours. By the same way, handling the FREE(i) is going to become more complicated once we start introducing relay nodes and linking constraints.

For this reason, we decompose the resolution process into two steps:

- during the first step, we only use the INSERT operator, while proceeding as in the INSERTION procedure described in section 2.4. Since we would like to use transfers and the related INSERT2 operator in order to make the whole tour system more efficient, we perform this first step while using

stronger transit bounds Δ_i, $i \in I$, that means while making in such a way the riding times of the demanders get improved;
- this first step is likely to yield rejected demands, because of the stronger transit. So, the second step deals with those rejected demands while only using the INSERT2 operator.

We augment the collection of inference rules by for checking the time validity:

- Rule R_6: y = *Twin*(x); *Status*(x) = *Out-Reload*; *FS*.min(x) > *FS*.min(y) |= *FS*.min(y) <- *FS*.min(x);
- Rule R_7: y = *Twin*(x); *Status*(x) = *Out-Reload*; *FS*.max(x) > *FS*.max(y) |= *FS*.max(x) <- *FS*.max(y).

We are now able to summarize the whole resolution process of the *Dial-a-Ride* problem with transfers. It consists in a "for" loop, during which the parameter λ progressively decreases from an initial value Λ until 1: the length P of this loop is a parameter of the main procedure**.** Any iteration inside this loop works as described second section:

- any Δ_i values are updated by the loop:
  - For i in I do {If $\Delta_i > \lambda$. DIST(o_i, d_i) then Δ^*_i <- λ. DIST(o_i, d_i) else Δ^*_i <- Δ^*_i};
- a first step involves a call INSERTION and yields some *Reject1* rejected demand index set, together with some pair (T, t), where T is a valid covering collection for I – *Reject1*, and t is a related time value set;
- in case *Reject1* is not empty, a second step is performed, which involves a call to a procedure INSERTION2, which works while trying to insert the rejected demands through applications of the INSERT2 operator;

The INSERTION2 procedure takes as input the 3-uple (T, t, *Reject1*) which was computed through the first step, and proceeds, according to a "while" loop, in order to insert demands of *Reject1* through the INSERT2 operator. Any time it enters this main "while" loop, it is provided with:

- a subset J of the *Reject1* set, which contains the demands which remain to be inserted and a current *Reject* set;
- a pair (T, t), related time windows *FP*(T)(x), $x \in$ ACT(T), and a related current active set ACT(T);
- sets FREE2-o(i) and FREE2-d(i), $i \in J$. We denote by N-FREE2-o(i) (N-FREE2-d(i)) the number of vehicle which appear in FREE2-o(i) (FREE2-d(i)); and it works as follows:

1. it picks up some demand i_0 in J: if there exists i such that N-FREE2-o(i) = 1 or N-FREE2-d(i) = 1, then i_0 is chosen in a random way among those demands in J which such that N-FREE2-o(i) = 1 or N-FREE2-d(i) = 1; else it is chosen in a random way among the demands in J which minimize the quantity N-FREE2-o(i) = 1 + N-FREE2-d(i);
2. it builds the *Weak-L-Candidate* list with those 7-uple (k_1, k_2, x_1, y_1, x_2, y_2, z) which are such that (k_1, x_1) $\in$ FREE2-o(i_0), (k_2, x_2) $\in$ FREE2-d(i_0), and (y_1, x_2, z) = *Exchange*(x_1, k_1, y_2, k_2);
3. for any 7-uple (k_1, k_2, x_1, y_1, x_2, y_2, z) in *Weak-L-Candidate*, the procedure checks if the tours are valid after the insertion, and, in case, the result is true, it inserts (k_1, k_2, x_1, y_1, x_2, y_2, z) into a *L-Candidate* list, ordered according to increasing to the related *Val* component values;
4. In case *L-Candidate* is empty, then i_0 is inserted into *Reject*, else:
   1. It picks up (k_1, k_2, x_1, y_1, x_2, y_2, z) in *L-Candidate*: it proceeds through a random choice among the up to N_7 elements of *L-Candidate.* N_7 becomes a parameter of INSERTION2;
   2. It creates two new active nodes related to (z, i_0, -1) and (z, i_0,1);
   3. It effectively performs the insertion: (T(k_1), T(k_2)) <- INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2, z);
   4. It updates t, together with the time windows *FP*(x), $x \in$ ACT(T), through application of the *Propagate2* and *Evaluate2* procedures;
   5. It updates the sets FREE2(i), $i \in J$;

Finally, the process keeps the best result (T, t, *Reject*, $Cost_{A,B}$(T, t)) which was ever obtained during this process.

4. Computational results

We apply our solution on a set of randomly generated instances. Each instance is different by the size of the windows, the number of demand, and the number of cars. Like in Cordeau et al. (2003), we randomly generated the coordinates of pick-up and drop-off nodes in the square of side 20. We split the square in 4 parts and the fleet *VH* in 4 sub-fleets *VH1*, *VH2*, *VH3*, and *VH* 4 related to the sub-squares *EP1*, *EP2*, *EP3*, and *EP4* (Fig. 3). *D* is classified in two sets: the transverse demands which are its origin node in a different sub-square than the destination and the local demands. For each instance studied here, 50% of the demands are local and uniformly set to the 4 sub-squares.

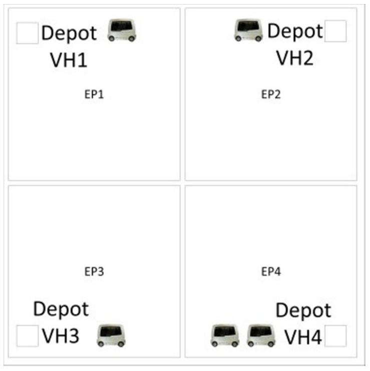

Fig. 3. Generation of 4 different depots

We generated a different maximum user ride time which equal to the product of 20 and the distance between the origin node and the destination node. The capacity *CAP* equals 6 for each vehicle. Each demand has a large time windows (all the day, from 0 to 1440 minutes) and another tight (15 or 30 minutes), their *Status* is grant randomly.

Our heuristic was implemented in C++ and each replication was run on the same thread of an Intel Q8300 (2.5 GHz). We performed 100 replications of 8 instances generated by the parameters written above. Table 1 gives the results. We provided $R1_c$ which is the rate of the demand inserted in the routes when the transfers are forbidden $R1_t$ is the same rate when transfers are allowed. We computed Gap such as Gap = $(R1_t- R1_c)/(R1_c/100)$. All the results are average of the 100 replications.

Table 1. DARP classic Vs DARP with transfers

Inst.	|D|	K	Window Size	RI_c	RI_t	Gap (%)
1	32	4	15	54.59	62.76	14.95
2	32	4	30	61.84	68.32	10.47
3	32	5	15	70.28	86.02	22.40
4	32	5	30	77.00	89.07	15.67
5	96	4	15	22.43	24.16	7.74
6	96	4	30	25.92	27.02	4.24
7	96	5	15	28.64	32.62	13.93
8	96	5	30	33.26	35.35	6.30

When comparing rates an average obtained by each resolution, about 11.5% of demand can be inserted if the transfers are allowed. $R1_t$ and $R1_c$ are obviously better when the fleet has more cars (K=5), but if the gap is more important it means there are more possibility to do a transshipment. The first fourth instances have 3 times less demands inserted than the second set (with the same fleet). The R1t and R1c for the first set are a little less than 3 times less than the second set. That is explain by the fact the second set are a bigger choice to included his demands.

5. Acknowledgments

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

REFERENCES

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

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

Cortes, C. E., Matamala, M., Contardo, C. (2010). The pickup and delivery problem with transfers, Formulation and a branch-and-cut solution method, European Journal of Operational Research 200 p711-724 .

Cortes, C. E., Jayakrishnan, R. (2002). Design and operational concepts of high-coverage point-to-point transit system, Transportation Research Record 1783 p178-187.

Deleplanque, S., Quilliot, A. (2012). Insertion techniques and constraint propagation for the DARP. *Computer Science and Information Systems* (FedCSIS). 393–400. IEEE conference publications.

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

Kerivin, H., Lacroix, M., Mahjoub, A. R., Quilliot, A. (2008). The splittable pickup and delivery problem with reloads, European Journal of Industrial Engineering 2 (2) p112-133.

Laporte, G., Cordeau, J.F. (2007). The dial-a-ride problem: models and algorithms. Annals of Operations Research.

Madsen, O., Ravn, H., Rygaard, J. (1995). 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.

Masson, R., Lehuédé, F., Péton, O. (2011). A tabu search algorithm for the Dial-a-Ride Problem with Transfers, Proceedings of the International Conference on Industrial Engineering and Systems Management.

Masson, R., Lehuédé, F., Péton, O. (2012). Simple Temporal Problems in Route Scheduling for the Dial–a–Ride Problem with Transfers, 2012, Lecture Notes in Computer Science, Volume 7298/2012, 275-291, DOI: 10.1007/978-3-642-29828-8_18.

Nakao, Y., Nagamochi, H. (2012). Worst case analysis for pickup and delivery problems with transfer, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E91-A (9) p2328-2334 .

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

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

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

Masson, R., Lehuede, F., Peton, O. (2012). An adaptive large neighborhood search for the pickup and delivery problem with transfers, Transportation Science in press.

Shang, J. S., Cu, C. K. (1996). Multicriteria pickup and delivery problem with transfer opportunity. Computers & Industrial Engineering.

Thangiah, S., Fergany, A., Awam, S. (2007). Real-time split-delivery pickup and delivery time window problems with transfers, Central European Journal of Operations Research - 15 329-349.