International Journal of Transportation
Vol.2, No.1 (2014), pp.21-32
http://dx.doi.org/10.14257/ijt.2014.2.1.02

A GRASP-based Approach for Demand Responsive Transportation

Rui Gomes[1], Jorge Pinho de Sousa[2] and Teresa Galvão Dias[3]

[1]*Faculdade de Engenharia da Universidade do Porto*
[2,3]*Faculdade de Engenharia da Universidade do Porto, INESC TEC*
Rua Dr.Roberto Frias, s/n, 4200-465, Porto, Portugal
[1]*pdst11003@fe.up.pt,* [2]*jsousa@inescporto.pt,* [3]*tgalvao@fe.up.pt*

Abstract

Demand Responsive Transportation (DRT) systems try to provide quality public transportation in low, variable and unpredictable demand scenarios, with routes and frequencies that may vary according to observed demand, possibly in real-time. The design and operation of DRTs involve multiple criteria and have a combinatorial nature that prevents the use of traditional optimization methods. To obtain an approximation of the Pareto solution set, we have designed a heuristic approach involving the construction of a feasible route through a greedy randomized procedure, followed by a local search phase, latter embedded in a Decision Support System that also uses simulation. The goal is not only to minimize operating costs but also to maximize the quality of the service. Experiments with simple cases, inspired in real problems, have shown the potential of this approach for efficiently designing and managing DRT services.

Keywords: *multiple-objective optimization; heuristics; decision support systems; public transport; DRT.*

1. Introduction

The provision of quality public transportation can be extremely expensive in low, variable and unpredictable demand scenarios, as it is the case of disperse rural areas or some periods of the day in urban areas. Demand Responsive Transportation (DRT) services address these issues by providing a kind of hybrid approach between a taxi and a bus, with routes and frequencies that may vary according to the observed demand. Due to this added flexibility, the service provided by operators becomes more efficient, with routes planned shortly before their start, better occupancy rates and vehicles' characteristics better suited to users' mobility requirements. However, in terms of financial sustainability and quality of the service, the design of this type of services may be difficult.

The problems of designing and operating DRT services are closely related to the Vehicle Routing Problem [1], and in particular to the Dial-A-Ride Problem [2]. DRTs extend the "classical" Vehicle Routing Problems (VRP) in a number of ways, being, at least, as complex as the later [3]. It is clear that in the DRT context, vehicles have a limited capacity, demands should be served within a certain time window, each stop along the route can be both a pickup and a delivery point and there is the uncertainty and variability associated with the number of stops along the route. In the Dial-A-Ride Problem (DARP) one is interested not only in minimizing the operating costs or the distance travelled by the vehicles but also (and this is sometimes more important) in maximizing the quality of the service, based on indicators such as the average passenger waiting time or the on-board (ride) passenger time [4].

ISSN: 2287-7940 IJT

DRT services can operate in a static or in a dynamic mode. In static mode, all requests are known before-hand (*a priori*, or advanced, requests), whereas in dynamic mode transportation requests are gradually revealed along the service operating time, with routes and schedules having to be adjusted to meet the demand [5]. In practice, however, "pure" dynamic services are not common since some requests are usually known *a priori*. Besides being a multi-objective problem, DRT systems can also be strongly dynamic [6], requiring the adaptation of solutions in real-time. Given the complexity of these problems [7], optimization methods are highly time-consuming, ruling out their usefulness in practice. Moreover when we consider multiple criteria, the "optimal" solution is in general meaningless because it is impossible to satisfy all (usually conflicting) objectives simultaneously [8].

In this work, we present a heuristic approach for constructing a feasible set of routes through a parallel greedy random approach, followed by a local search phase, minimizing operating costs and maximizing the service quality. We are interested in finding a set of efficient solutions hopefully close to the Pareto front.

The paper is organized as follows. Section 1 presents a brief introduction. Then, section 2 describes the problem in a concise and formal way. Section 3 details the developed heuristic approach. Section 4 presents some implementation details and, particularly, the graphical user interface of a developed Decision Support Systems that embeds the presented heuristic approach. Section 5 presents some results. Finally, section 6 summarizes our findings.

2. Problem Description

In the Dynamic Vehicle Routing model for DRT we assume that passengers specify origins and destinations from a set of pre-defined possible route points, a pickup time window and a desired arrival time for their transportation needs, and that they are to be served by a fleet of vehicles of equal capacity (number of seats). Each possible route point, with the exception of the depot, can be a pickup-only point, a delivery-only point, or both. At a given pickup location, different passengers entering the vehicle can have different destinations and different time windows. Several users can be simultaneously transported in one vehicle, like a mini-bus. Vehicles start and end their trips at a single depot and transportation requests can be received at any time, from any origin. Since different users have different transportation needs, each stop along the route can have multiple (possibly disjoint) time-windows, which, in association with the real-time arrival of new requests, may require several visits to a given stop at different time periods. This is a major difference from all know variants of the VRP and DARP problems – and quite a fundamental one, thus requiring innovative approaches.

3. DRT Service Operation

For transportation requests known *a priori* (static routing) their spatial distribution of transportation requests and their arrival rate to the system do not influence the routing algorithm, but the expected travel time does play an important role. For the dynamic routing, we followed a traditional approach, which is to solve static scenarios [5] when a new request arrives, so the above mentioned rational applies and each link on the service network is associated with the mean and standard deviation travel times as function of the period of the day, based on historical data (as in [9]). With the "solving static scenarios" approach (real-time optimization) the routing algorithm must be fast enough to (re)calculate a solution in case requests arrive in a quick sequence. Another possible algorithmic approach would have been to improve the solution found for the initial static scenario taking advantage of (probabilistic) knowledge of the future, determining one or more routes base on probabilistic information on future requests, travel times, and other parameters. However, the dynamic

environment of real-world instances of on-line DRTs may lead to high computational times using this approach.

In dynamic DRTs part of the necessary information becomes available only during the operation period. There is an initial route schedule that incorporates all data currently known and this route schedule is adjusted only when required using the most recent data. Route sections already traversed until the arrival of the new request are, obviously, unchangeable. Thus, the problem is re-optimize the remaining part of the initial solution after the insertion of the new request(s), taking in account that all the previous feasible request are already in the on-going routes. Once a passenger has been served, i.e. picked up and delivered, that passenger will be removed from the planned route thereafter. Passengers not yet picked up can be served by any vehicle but passengers already picked up must be delivered, naturally, by the same vehicle. With time windows constraints, the insertion of a new request in real-time is more complex: sometimes this new request has to be refused because it is not possible to neither include them in any routes, nor have another vehicle to start a new route. We considered pickup time-windows as hard constraints and delivery time-windows as soft constraints that can be violated by a certain amount of time defined by each operator as a quality of service indicator (but could also be defined as hard-constraints). These problems are NP-hard [7] and therefore optimal solutions cannot be reached in useful time.

We have designed a parallel heuristic approach that constructs a feasible route through a reactive greedy random approach followed by a local improvement phase that can be run when a new request arrives. Our main effort was devoted to build the highest quality possible solutions in the construction phase. In order to "involve" the experts in the planning process, a Decision Support System (DSS) embedding the heuristic approach and also integrating simulation was developed. The simulation integrates four models, covering the service area, the trip requests, the vehicles, and real time events. A detailed description of the developed simulation model is beyond the scope of this paper. How the optimization and simulation phases relate to each other can be seen from two perspectives. The first is to find an optimal solution to a specific case, and then simulate what effects this solution has on the performance, customer behavior, and other key performance indicators. The second perspective is to find a good overall design by the use of simulation, and then use optimization to find the best solution to a specific instance of the given design. The developed DSS adopts this second perspective.

3.1. Construction Phase: a Greedy Constructive Algorithm

Each feasible transportation request is formed by an origin, a destination and pickup and delivery times. Having a set of requests, the algorithm tries to find a set of trip sequences (routes) considering the objectives and respecting all problem constraints. The problem objectives are classified into two perspectives: a vehicle's perspective and a passengers' perspective. A Node Ranking Function (NRF) has been defined to find, at each iteration, the next "best" node to be inserted into the route under construction, taking into account the two aforementioned perspectives. In terms of the vehicle's perspective, the major factors for determining the next node to be selected are the distance to all other nodes from the current position and the number of passengers on those nodes. From the passengers' perspective, the major factors to be considered are the number of passengers on the bus having as destination a given node, and the time windows on the remaining nodes. For each of these factors a weight is assigned, to account for the different perspectives of the decision maker in a multi-criteria context. Let α_d be the weight of the distance factor, α_p the weight of the number of passengers' factor, α_v the weight of the delivery time window factor and, finally, α_t the

weight of the pickup time window factor. Let also W be the set of all nodes defining the problem and NW the subset of nodes not yet in the solution routes. The NRF is defined as:

$$\forall i \in NW, NRF[i] = \left(\alpha_d \times CRL[i] + \alpha_p \times NRL[i]\right) + (\alpha_v \times DRL[i] + \alpha_t TRL[i])$$

CRL (Cost Rank List) is an ordered list of the normalized travel costs to each node. *NRL* (Number of passengers Rank List) is the ordered list of the normalized number of passengers at each node. *DRL* (Delivery Time Rank List) is the ordered list of normalized delivery lower time limit at each node. *TRL* (Time Rank List) is the list of normalized pickup lower time limit at each node. The node with the highest NRF value is added to the route under construction at the end of each iteration. Figure 1 presents the NRF algorithm:

3.2. Improvement Phase

The improvement phase tries to improve the constructed solution by exploring solutions in some neighborhood of the current solution. But the problem is highly constrained and the combination of simultaneous pickup and delivery with the possibility of having multiple pickup and/or delivery time-windows at each stop makes the definition of neighborhood structures non-trivial and their implementation computationally very complex [10].

For the improvement phase, we have used a combination of three mechanisms: the forward slack time, a "nearby-stops" analysis, and a simple 2-exchange procedure. After calculating the "dead" times available through the route (forward slack time), the "nearby-stop" analysis tries to find in-between stops that appear later in that route and can be served in the meantime. The last improvement is a simple 2-exchange procedure based on the k-interchange procedure by [11].

3.3. Orchestrating the Two Phases: a GRASP-type Metaheuristic

In order to produce better solutions, hopefully closest to the Pareto front, the presented NRF algorithm was embedded in a GRASP type [12] metaheuristic. One appealing characteristic of GRASP that we explored, mainly because our need of real time solutions, is that it can be trivially implemented in parallel, with each GRASP iteration being performed in parallel with only a single global variable required to store the best solution found over all processors. We have also implemented a reactive mechanism that reacts to solutions produced and tries to adjust the balance between greediness and randomness [13] of the process.

Step 1: inicialize $S = \{\}$
While $NW \neq \emptyset \wedge num_{vehicles} \neq 0$
Step 2: inicialize $R = \{\}$
Step 3: start at the depot $R = \{0\}$
Step 4: compute the NRF rank of all feasible nodes:
Step 4.1: build the *Cost Rank List* (CRL) – sort all nodes by increasing distance from the current one, and normalize the values obtained, such the closest node is assigned with the highest value and so on;
Step 4.2: build the *Number of Passengers Rank List* (NRL) – sort all nodes by decreasing order of the load and normalize.
Step 4.3: build the *Delivery time-window Rank List* (DRL) – sort all nodes with delivery requests by increasing order of the closest delivery time associated to the node plus the trip time to that node and, finally, normalize so that the "earliest" gets the highest score and so on.
Step 4.4: build the *Time-window Rank List* (TRL) – sort all nodes with pickup requests by increasing order of the closest pickup time associated to the node plus the trip time to that node and, finally, normalize so that the "earliest" gets the highest score and so on.
Step 5: compute NRF for each node, such that
$\forall i \in NW, NRF[i] = (\alpha_d \times CRL[i] + \alpha_p \times NRL[i]) + (\alpha_v \times DRL[i] + \alpha_t TRL[i])$
Step 6: select the node with highest NRF that does not violate the constraints (feasible node) and add it to the route – $R = R \cup max(NRF[i])$;
Step 7: update requests data, eventually removing the ones already satisfied (picked up and delivered), i.e., $NW = NW \backslash max(NRF[i])$, and moving the unfeasible ones to a temporary list U
Step 8: **if** $NW = \emptyset$ **then** add the depot node (0) to the end of the route R and add this route to the solution set S, i.e., $S = S \cup R$;
Step 9: **if** $U \neq \emptyset$ **then**
let $NW = U$ and $num_{vehicles} = num_{vehicles} - 1$
goto Step3;
else goto next step;
end-while
return solution S

Figure 1. Node-Ranking Function Algorithm

In the construction phase, a feasible solution (set of routes) is built by adding to the each initially empty route one element at a time. The evaluation of each element according to the criteria is made by the already mentioned NRF function. Each NRF algorithm iteration constructs a candidate list (CL) of the elements to be inserted in the current route. From this CL a number of its best elements are selected to form a restricted candidate list (RCL). The size of the RCL is defined by a parameter $\alpha \in [0,1]$ that sets either the numbers of elements or a threshold between the value of best element of the CL and the value of the last element to be included in the RCL. This last approach was considered the best for the problem at hand, because, being a very constrained problem and due to the adaptative nature of each iteration of the algorithm, the size of the CL varies and sometimes has very few elements. We implemented a Reactive GRASP that reacts to solutions produced using different values for the α parameter and tries to adjust it to give the GRASP the "best" balance between greediness and randomness. At each GRASP iteration, the α parameter is chosen from a discrete set of values $\{\alpha_1, .., \alpha_m\}$. The probability of selecting a given α_i is $p(\alpha_k), k = 1, .., m$ and these probabilities are adjusted to favor α values that produce good solutions. Initially, we set equal probabilities to each α_k, *i.e.*, $p(\alpha_k) = 1/m$, $k = 1, .., m$, then periodically, at every 200 GRASP iterations (for instance), the probabilities are updated according to $p(\alpha_k) = q_k / \sum_{j=1}^{m} q_j$, where $q_k = F(S^*)/A_k$, being $F(S^*)$ the cost of the best

solution found so far and A_k the average cost of the solutions found with α_k. Being such a constrained problem, it comes as no surprise that the best solutions are found with higher values for the α parameter. The next step is to randomly select one element from the RCL and insert it in the route being constructed. When a route cannot satisfy any more transportation requests, the route is finished. If there are any unsatisfied feasible requests left and other vehicles available, a new route for another vehicle is started and built in the same manner. The process is repeated until there are no more feasible transportation requests left to satisfy or no more available vehicles (it's useful to recall at this stage that, if one has enough vehicles at hand, every feasible request can be satisfied assigning a "private" vehicle to it). The final solution is the resulting set of routes and its cost is calculated. The found solution is then used in the local improvements phase until a better solution is found (first-best) or the available time for improvements runs out.

Figure 2 presents the high level pseudo-code of the Parallel Reactive-GRASP for the Dynamic Vehicle Routing for Demand Responsive Transportation (DVRDRT), where at every 200th iteration the probabilities of the GRASP α parameter that controls the "reactiveness" are updated.

```
Parameters:GRASP_max_iterations
while (num_iterations<GRASP_max_iterations)
        choose α_k parameter with probability p(α_k) = 1/m , k = 1,..,m
        initialize S = { }
        Construction phase:
                Calculate S using NRF and α_k for the RCL
        If mod(num_iterations,200)==0 then p(α_k) = q_k / Σ_{j=1}^{m} q_j , k = 1, ..., m
        Calculate the solution cost  F(S) = (Σ_{i=1}^{m} C(R_i) + Σ_{j=1}^{u} C(U_j))
        Improvement phase:
                using S do:
                        forward slack time
                        nearby stops analysis
                        simple 2-exchange procedure
                until F(S'') < F(S) or elapsed_time>allowed_running_time
                if F(S'') < F(S)  then S = S'
        update best solution found S*: if F(S) < F(S*) then S* = S
end-while
return best solution S*
```

Figure 2. Reactive-GRASP for DVRDRT Pseudo-code

4. Implementation Details

The developed system is composed of:

- a Decision Support System including the heuristic approach simulation;
- a desktop booking client prototype;
- a mobile booking client prototype;

The system was developed using C++ programming language, in a notebook with Intel Core2 Duo processor, 1,67 GHz, with 2GB DDR2-667 memory, running Windows 7 Professional 32bits. In terms of development tools, the business and data access tiers were developed using Microsoft Visual Studio C++ 2008, with Open Multi-Processing (OpenMP) [14] parallel programming API. The user interface of both the Decision Support System and the two clients and the networking and communications services were developed using Nokia QT 1.0. The mobile client was developed on a Nokia E71 mobile phone. Figure 3 shows the graphical user interface (GUI) of the developed DSS.

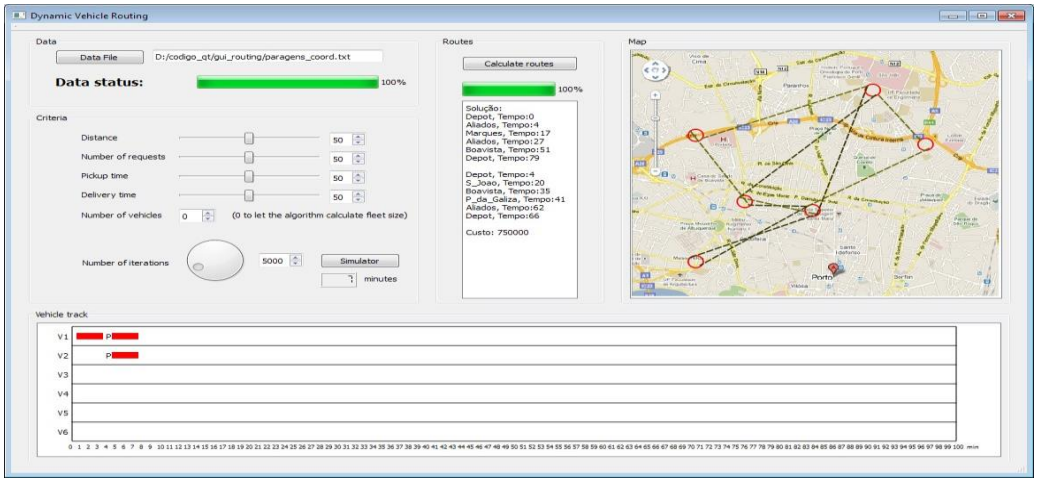


Figure 3. Decision Support System User Interface

In conjunction with its nature and structure, two other factors contribute to the high performance of the heuristic presented, namely the C++ Open Multi-Processing parallel programming API and the data pre-processing. A brief description of both factors and their integration on the heuristic approach proposed follows.

4.1. Open Multi-Processing

Open Multi-Processing (OpenMP) [14] implements multithreading parallelization [15], leveraging the development of parallel applications. Both task parallelism and data parallelism can be achieved using OpenMP.

As already mentioned, one appealing characteristic of a GRASP implementation that we wanted to explore was that it can be trivially implemented in parallel, with each GRASP iteration being performed in parallel with only a single global variable required to store the best solution found over all processors. OpenMP was used to achieve parallelization: each thread does one iteration of the Reactive-GRASP type heuristic. Ideally, each thread (iteration) would run on its own processor and the total run time of any number of iterations would be constant and equal to the time one processor takes to do one iteration. But having that many processors is not cost-effective (nor feasible). So what we did was to dynamically fork the threads according to the number of cores on the platform hardware, taking care that the "thread splitting" process introduces overhead times and resources consumption. The same rationale applies to the "thread joining" at the end. There needs to be a balance between the number of threads executing concurrently in a single machine and the resources available in the machine. After some testing, we came to the conclusion that, in the case of the proposed heuristic approach, the advisable number of threads on a single machine would be 4 times the number of processor cores and this "parameter" is set automatically - for instance, if the algorithm is run on a dual-core machine it will have 8 threads running the algorithm in parallel; if the algorithm is run on a quad-core machine it will have 16 threads running in parallel. All tests and results presented were obtained in a dual-core machine. It is expect to see performance gains running the algorithm in platforms with more cores.

4.2. Data Pre-processing

One of the criteria is the minimization of the travel cost and to define the next node to construct a feasible route at each iteration it is necessary to calculate the distance from the

present position to all other nodes – as is very often the case in many transportation problems. A possible approach was to implement the Dijkstra's algorithm [16]. But, besides being necessary to run the algorithm many times during a route construction (in the worst case, as many times as the number of nodes), the procedure was repeated for the other routes in the (possible) route solution set and, even worse, all this was repeated as many times as iterations on the outer GRASP loop. So, the approach was to do this as a setup procedure before running the algorithm. But as the Dijkstra's algorithm has a $O(n^2)$ complexity order, even this pro-processing might be prohibitive in terms of computer time for real-life size problems, with thousands of stops (nodes). It was therefore important to have a highly efficient method for obtaining these shortest paths.

The Floyd-Warshall Algorithm [17] is an efficient dynamic programming algorithm to find all-pairs shortest paths on a graph. A single execution of the algorithm will find the lengths of the shortest paths between all pairs of vertices. But this algorithm has a drawback when compared to the Dijkstra's algorithm: it does not return details of the paths themselves. As we needed to be able to re-construct the path, we implement a small tweak in the algorithm for maintaining a record of the shortest paths [18]. The Floyd-Warshall Algorithm has a $O(n^3)$ complexity order, which is worse than the Dijkstra's algorithm complexity order, but as it returns all all-pairs shortest paths on a graph and the respective path in a single algorithm run and this information is (highly) unlike to change in the course of the day, we run the Floyd-Warshall Algorithm only at the starting of the system and the data will be always available to the proposed heuristic. So, in fact, we took the burden from the heuristic and placed it at the start of the system. Every time the algorithm needs the shortest-path information it only takes the linear time necessary to access a matrix position with that information, instead of running the Dijkstra's algorithm every time.

5. Experiments

Being a "new" problem, there are no "off-the-shelf" benchmark data bases to test the algorithm for DRTs and to compare it with other published approaches. To the best of our knowledge, the most similar instances in the literature are the ones for the Capacitated VRP with Time Windows [19], the Capacitated VRP with Pick-up and Deliveries and Time Windows [20] and the Dial-A-Ride-Problem [21]. But, the adaptation of the benchmark database itself to convert it to a DRT instance is needed and this raises a series of questions, being the most important to know to what extent the results eventually obtained compare with other author approaches. For example, in [19] each costumer as a demand of X units, a delivery time window and a service time. In order to use this instance in DRT problems, one has to convert the X units demand into X different transportation requests (i.e., X different passengers) with the same delivery time window, assume that they all have the as origin depot – which is not "real" as per DRT definition -, or some random stop, and, finally, ignore the service time. Each of the other benchmark datasets poses similar problems, so, our decision was to use randomly generated instances for the city of Porto, Portugal, with different number of stops, different number of requests and with different degrees of dynamism (DOD). Being a proof of concept, we chose Porto because the needed data was readily available from several sources.

5.1. Test Instances

The service area for each test instance is a graph defined by a set of nodes, corresponding to the available stops, and links, corresponding to the roads connecting the stops. For the nodes, we used the stops of "Sociedade de Transportes Colectivos do Porto, S.A." (STCP) -

bus operator for the municipality of Porto – network, corresponding, roughly, to an area of 41,66 km². The assumed commercial speed of the vehicles is 16 km/h. For each stop, we use its real geographic coordinates and calculate the links lengths based on straight line distances between the stops. The road network is, thus, a graph defined by a set of nodes, representing the available stops, and a set of links, representing the roads connecting the stops. When a vehicle enters a given link, the travel time is randomly selected from a lognormal distribution with mean and standard deviation as functions of the time the vehicle entered the link – as in [9]. In future developments, this information may be fed directly from detailed microscopic network conditions. The depot is situated at "Francos" (a real STCP depot location).

For each test instance, the total number of generated requests is the sum of both advanced and real time requests. Advanced transportation requests have a number of attributes: desired pick-up time, pick-up location, desired delivery time and delivery location. Real-time requests have an additional attribute: request time. We used 15 minutes as the shortest time limit to place a request - as this was the shortest value found in the European DRT survey we made - and randomly select request times with uniform probability between 15 to 60 minutes. The request arrival time to the service is modeled as a Poisson process, as this seems to be a general assumption for transportation related works [18], with parameter $\lambda = 0{,}3$. Adding the request time limit to the request arrival time to the service, we have the user desired pickup time. For the users' "expected" travel time (i.e., to generate the desired delivery time), we use the normal distribution, with a mean of 35 minutes for Porto (according to Instituto Nacional de Estatística (INE) in 2001) and a standard deviation of 17 minutes [22].

We generate origin and destination locations of the requests assuming that all nodes in the network have the same probability of being departure or destination points, i.e., assume that requests occur uniformly in any place of the service area.

Table 1 shows the dimension of the generated test instances, in terms of number of stops and requests. For each of these dimensions, as already mentioned, we tested also with different DODs: from 0% to 90% (we considered that all services had at least some requests in advance), with 10% increments.

Table 1. Dimension of Generated Test Instances

Number of stops	10	25	50	100	250	500	750	1000
Number of requests	5	10	20	50	75	100	250	500

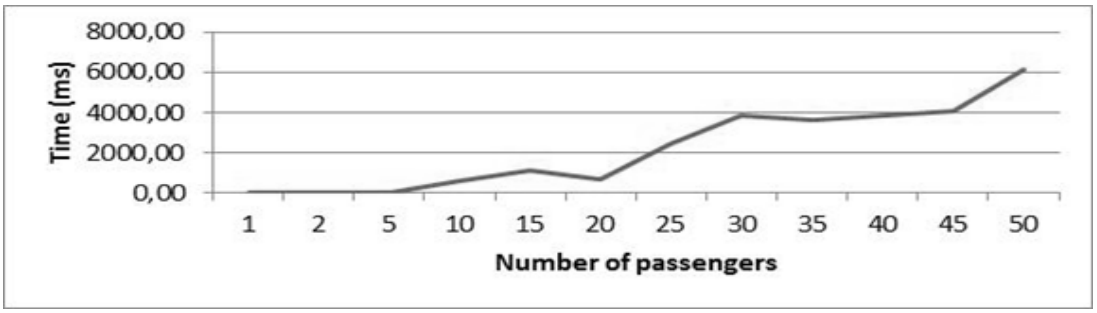


Figure 4. Number of Passengers' Effect on the Algorithm

5.2. Results

Computational tests were done using an Intel Core Duo running at 1,67GHz, 2GB RAM memory, and the adjustment of the parameter that controls greediness/randomness level at every 100th algorithm iteration. The number of parallel threads running the algorithm is dynamically set to 8.

Computational results on the defined instances look very promising, both in terms of cost savings and in terms of computational efficiency. These results seem to highlight that the major factor affecting the algorithm running time is the number of passengers. The Figure 4, obtained using 50 stops and 1000 algorithm iterations, shows the effect of increasing the number of passengers (requests).

Moreover, if the number of passengers is fixed, adding possible stops does not increase the algorithm running time. Another observation is the linear increase in running time with the number of iterations, the running time for each iteration being constant – this is in line with literature results for GRASP-based algorithms.

One common way to evaluate algorithmic approaches for Dynamic Vehicle Routing problems is to use the competitive analysis framework [6], but, typically, only very simple versions of the problem, that do not contemplate side constraints such as time-windows, can be analyzed with such framework. Our approach was to increase of the degree of dynamism [23] - ratio between the real-time requests over the total number of requests -, to analyze how the overall solution cost increases compared to having all information in advance (static scenario) and, as such, provide an empirical estimate of the competitive ratio of the algorithm. For the same problem, the solution cost of a 90% degree of dynamism scenario, with requests distributed evenly throughout the planning horizon, is around 45% higher than the static scenario with all information known a priori (0% degree of dynamism). Figure 5 shows the increase cost structure found.

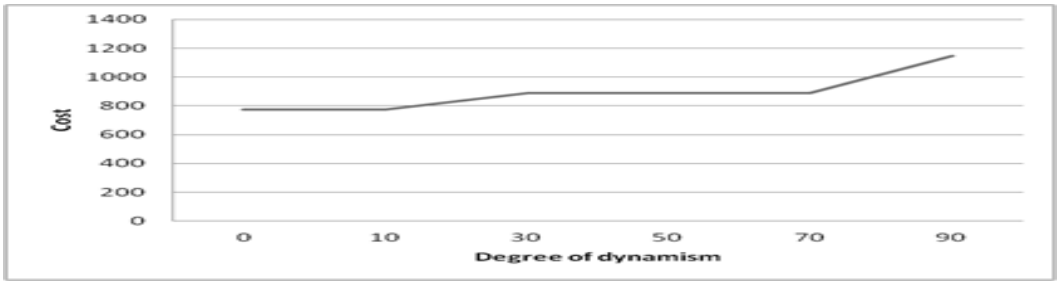


Figure 5. Increase Cost Structure

6. Conclusions

Providing quality public transportation is extremely expensive when demand is low, variable and unpredictable. DRT services try to address this problem by providing a kind of hybrid approach between a taxi and a bus, with routes and frequencies that may vary according to the actual observed demand. The problems of designing and operating DRT services are closely related to the Vehicle Routing Problem (VRP), and in particular the Dial-A-Ride models. Given the complexity of these problems, optimal solutions can take an enormous amount of time to be found, ruling out their usefulness in the context at hand. Besides, in a multiple criteria decision analysis the "optimal" solution is in general meaningless because it is impossible to satisfy all (usually contradictory) objectives

simultaneously. So we are interested in finding a set of efficient solutions hopefully close to the Pareto front.

Computational results on randomly generated instances look very promising, both in terms of cost savings and in terms of computational efficiency. The running time for each iteration is constant and the solution cost of a 90% degree of dynamism scenario seems only around 45% higher than the static scenario with all information known a priori.

The parallel reactive GRASP based constructive heuristic algorithm developed allows for the multiple perspectives of the different stakeholders to be taken into account, improving decision-making process, and has real-time performance to cope with the degree-of-dynamism of the problem. Planners can use the developed Decision Support System to design DRT services that meet envisaged cost levels and quality of service.

Acknowledgements

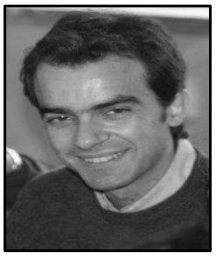
This work was supported by the Portuguese National Science Foundation (FCT) under Grant SFRH/BD/42974/2008. The author gratefully acknowledges this generous support. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the FCT.

References

[1] G.Dantzig and J. amser, "The Truck Dispatching Problem", Management Science, vol. 6, no. 1, **(1959)**, pp. 80-91.
[2] J. F. Cordeau and G. Laporte, "The dial-a-ride problem: models and algorithms", Annals of Operations Research, vol. 153, no. 1, **(2007)**, pp. 29-46.
[3] J. F. Cordeau, G. Laporte, M. Savelsbergh, D. Vigo and C. Barnhart, Chapter 6 Vehicle Routing, in Handbooks in Operations Research and Management Science, Elsevier, **(2007)**, pp. 367-428.
[4] J. Paquette, J. F. Cordeau and G. Laporte, "Quality of service in dial-a-ride operations", Computers & Industrial Engineering, Press, Corrected Proof, **(2010)**.
[5] H. Psaraftis, "Dynamic vehicle routing: Status and prospects", Annals of Operations Research, vol. 61, no. 1, **(1995)**, pp. 143-164.
[6] A. Larsen, O. Madsen and M. Solomon, "Classification Of Dynamic Vehicle Routing Systems", Dynamic Fleet Management, V. Zeimpekis, et al., Editors, Springer US, **(2007)**, pp. 19-40.
[7] J. Lenstra and A. Kan, "Complexity of vehicle routing and scheduling problems", Networks, vol. 11, no. 2, **(1981)**, pp. 221-227.
[8] J. Branke, K. Deb, K. Miettinen and R. Słowiński, "Multiobjective Optimization: Interactive and Evolutionary Approaches", Springer-Verlag, **(2008)**.
[9] E. Taniguchi, "City logistics: network modelling and intelligent transport systems", Amsterdam; New York: Pergamon, **(2001)**.
[10] G. Kindervater and M. Savelsbergh, "Vehicle routing: handling edge exchanges", Local Search in Combinatorial Optimization, E. Aarts and J.K. Lenstra, Editors, John Wiley & Sons: Chichester, **(1997)**, pp. 311-336.
[11] H. Psaraftis, "k-Interchange procedures for local search in a precedence-constrained routing problem", European Journal of Operational Research, vol. 13, no. 4, **(1983)**, pp. 391-402.
[12] T. Feo and M. Resende, "A probabilistic heuristic for a computationally difficult set covering problem", Operations Research Letters, vol. 8, no. 2, **(1989)**, pp. 67-71.
[13] M. Resende and C. Ribeiro, "Greedy Randomized Adaptive Search Procedures", Handbook of Metaheuristics, F. G.a.G.A. Kochenberger, Editor, Springer New York, **(2003)**, pp. 219-249.
[14] OpenMP, "The OpenMP API specification for parallel programming", [cited 2010 01-08-2010]; The official OpenMP web site]. Available from: http://openmp.org (2010), **(2010)**.
[15] B. Chapman, G. Jost and R. v. d. Pas, "Using OpenMP: Portable Shared Memory Parallel Programming", Scientific and Engineering Computation, MIT Press, vol. 300, **(2007)**.
[16] E. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, vol. 1, no. 1, **(1959)**, pp. 269-271.
[17] R. Floyd, "Algorithm 97: Shortest path. Commun", ACM, vol. 5, no. 6, **(1962)**, pp. 345.
[18] R. Larson and A. Odoni, "Urban Operations Research", Englewood Cliffs, New Jersey: Prentice Hall, **(1981)**.

[19] M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints", Operations Research, vol. 35, no. 2, **(1987)**, pp. 254-265.
[20] L. Haibing and A. Lim, "A Metaheuristic for the Pickup and Delivery Problem with Time Windows", **(2001)**.
[21] G. Laporte and J. F. Cordeau, "VRP data", [cited 2010 04]; VRP data web page. Available from: http://neumann.hec.ca/chairedistributique/data/, **(2010)**.
[22] C. M. d. P.e. Melo, "Deslocações Pendulares da População Empregada Residente na Área Metropolitana do Porto", Faculdade de Engenharia da Universidade do Porto, Universidade do Porto, Porto, **(2002)**, pp. 177.
[23] K. Lund, O. Madsen and J. Rygaard, "Vehicle Routing Problems with Varying Degrees of Dynamism: Technical University of Denmark", Institute of Mathematical Modelling, **(1996)**.

Authors



Rui Gomes, holds a MSc in Informatics Engineering and a PhD in Transportation Systems from the MIT-Portugal Program (Faculty of Engineering of the University of Porto - FEUP). He is Post-Doctoral Researcher at Faculty of Sciences and Technology of the University of Coimbra. He was a Visiting Graduate Student at Massachusetts Institute of Technology (MIT) in 2011 and Invited Assistant Professor at FEUP between 2008 and 2012. His main research interests are combinatorial optimization problems, metaheuristics and transportations systems.

Jorge Pinho de Sousa, has a PhD in Operations Research (Université Catholique de Louvain, Belgium). He is Associate Professor in the University of Porto, and director of the Doctoral Program in Transportation Systems. As a senior researcher at INESC Porto (Instituto de Engenharia de Sistemas do Porto), he coordinates the Manufacturing Systems Engineering Unit. He has actively participated in several large national and European R&D projects in areas such as decision support systems, operations management, cooperative networks, transportation systems and mobility. Regular publication of papers in international scientific journals. From 2005 to 2009, was President of the Portuguese Operations Research Society (APDIO).

Teresa Galvão Dias, has a background in Mathematics and holds a PhD in Sciences of Engineering (University of Porto, Portugal). She is Assistant Professor in the Faculty of Engineering of University of Porto. She has actively participated in several large national and European R&D projects in areas such as decision support systems, transportation systems and mobility. Her main research interests are combinatorial optimization problems, metaheuristics, human-computer interaction and transportations systems and she regular publishes papers in international scientific journals.

