Large Neighborhood Search for Dial-a-Ride Problems

Siddhartha Jain and Pascal Van Hentenryck

Brown University, Department of Computer Science,
Box 1910, Providence, RI 02912, U.S.A.
{sj10,pvh}@cs.brown.edu

Abstract. Dial-a-Ride problems (DARPs) arise in many urban transportation applications. The core of a DARP is a pick and delivery routing with multiple vehicles in which customers have ride-time constraints and routes have a maximum duration. This paper considers DARPs for which the objective is to minimize the routing cost, a complex optimization problem which has been studied extensively in the past. State-of-the-art approaches include sophisticated tabu search and variable neighborhood search. This paper presented a simple constraint-based large neighborhood search, which uses constraint programming repeatedly to find good reinsertions for randomly selected sets of customers. Experimental evidence shows that the approach is competitive in finding best-known solutions and reaches high-quality solutions significantly faster than the state of the art.

1 Introduction

The Dial-a-Ride Problem (DARP) is a variant of the Pickup and Delivery Problem (PDP), frequently arising in door-to-door transportation services for elderly and disabled people or in services for patients. In recent years, dial-a-ride services have been steadily increasing in response to popular demand. [8]. A DARP consists of n customers who want to be transported from an origin to a destination. Requests can be classified as *outbound* (say from home to the hospital) or *inbound* (from hospital back to the home). DARPs can be rather diverse and there is no standard formulation in literature. Various formulations try to balance the cost of the route and user inconvenience via soft and hard constraints. One formulation minimizes the weighted sum of total routing cost, time-window violations, and the number of vehicles used [1]. Another has multiple depots, a heterogeneous fleet, service times, time windows, and maximum customer ride times [13]. Yet another minimizes the weighted sum of the customer transportation times, the excess customer ride time with respect to direct and maximum ride time, time-window violations, customer waiting time and excess work time [10]. A survey of various DARP models and the algorithms used to solve them is given in [8].

This paper studies the formulation of Cordeau et al. [7] defined in terms of a fixed number m of vehicles, which makes sense in practice. There is only one depot. There are time-window constraints on the pickup or delivery vertex depending on whether the request is inbound or outbound. We also have service times, maximum ride time, and maximum route duration constraints. The objective is to minimize the total routing cost,

J. Lee (Ed.): CP 2011, LNCS 6876, pp. 400–413, 2011.

i.e., the travel distance. A tabu-search procedure to solve the *static* version of the problem where the requests are known in advance was presented by Cordeau et al [7] while a Variable Neighborhood Search procedure was proposed recently by Parragh et al. [11]. A procedure for testing the *satisfiability* of an instance was given by Berbeglia et al. [5]. A procedure for testing the satisfiability for the *dynamic* version of the problem, where only a subset of requests is known in advance, was presented in [4]. In general, one is interested in finding high-quality solutions to DARPs since, as the name indicates, customers call for a service.

This paper presents a large neighborhood search for DARPs and makes the following contributions:

1. It proposes a large neighborhood search LNS-FFPA (FFPA will be defined in Section 6) which significantly outperforms the traditional LNS algorithm used in vehicle routing (e.g., [12,3,2]).
2. It shows that LNS-FFPA significantly improves the quality of the routings found under tight time constraints compared to the state-of-the-art variable neighborhood search and tabu-search algorithms.
3. It shows that LNS-FFPA compares very well with the state-of-the-art constraint-programming approach to find feasible solutions to DARPs.

From a technical standpoint, LNS-FFPA features two novelties. First, it does not impose that the neighborhood search must find an improving solution. Second, LNS-FFPA terminates the neighborhood search after finding a feasible solution. This solution is accepted using a Probabilistic criterion, which allows worse solutions to be accepted for subsequent iterations. As mentioned earlier, the diversification resulting from these two design decisions is key in finding high-quality solutions under time constraints. It is also important to emphasize that LNS-FFPA, which is a very generic search technique, improves solution quality under very tight constraints over highly dedicated local search implementations. This makes it ideal for the highly dynamic environments in which DARPs arise.

The rest of the paper is organized as follows. We first present the problem formulation and give an overview of the state-of-the-art. This review should give readers a sense of the sophistication of the existing approaches. We then present our large neighborhood algorithm LNS-FFPA, report the experimental results, and conclude the paper.

2 Formulation

The input to DARP consists of the number m of vehicles, n requests, the maximum ride time L for customers, the maximum route duration D, and the planning time horizon T, i.e., the hours between which the vehicles can operate. A DARP is defined on a complete graph $G = (V, E)$ where $V = \{v_0, v_1, \ldots, v_{2n}\}$ is the set of vertices and $E = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the set of edges. Vertex v_0 denotes the depot. Each request i ($1 \leq i \leq n$) consists of a pair of vertices (v_i, v_{i+n}). With each vertex v_j is associated a service duration $d_j \geq 0$, a load q_j and a time window $[e_j, l_j]$. We also have $d_0 = 0, q_0 = 0$, and $e_0 = 0, l_0 = T$. Service duration is the time needed to service a vertex. Requests are either *inbound* or *outbound*. If i is an outbound request, then

the pickup vertex v_i has the time window $[0, T]$ and is called *non-critical*, whereas the delivery vertex is called *critical*. If i is an inbound request, then the delivery vertex v_{i+n} has time window $[0, T]$ and is non-critical whereas the pickup vertex is critical. The matrix $t_{i,j}$ also denotes the distance between vertices i and j. Given these definitions, a DARP consists in finding a route for each of the m vehicles such that (1) the route begins and ends at the depot; (2) The load of a vehicle k never exceeds its capacity Q_k; (3) The total duration (i.e., the difference between the end time and the start time) never exceeds a preset bound T_k; (4) For each request i, v_i and v_{i+n} are serviced by the same vehicle and v_{i+n} is visited after v_i; (5) The ride time of any customer (i.e., the difference between the serving time at the delivery vertex and the departure time at the pickup vertex) does not exceed L; (6) For each vertex v_i, the starting time of its service lies between $[e_i, l_i]$; and (7) The total routing cost of all vehicles is minimized. In our formulation as in [7], the total routing cost is equal to the total distance traveled by the vehicles.

3 A Constraint Programming Approach

Berbeglia et al [5] use a Constraint Programming (CP) approach for the DARP formulation in [7], except they do not model the route duration constraint and do not attempt to minimize the routing cost. Their focus is to check the satisfiability of DARP instances.

The Model. Each vertex v_i has a successor variable $s[i]$ and there is an *AllDifferent* constraint on all successor variables. The precedence, time window, ride time, and maximum vehicle capacity constraints are modeled via auxiliary variables representing the load, serving vehicle, and serving time for each vertex. The routes are constructed by branching on the successor variables.

Variable Selection. Let S be the set of all successor variables with the smallest domains. For every value v in the domain of some variable in S, the CP algorithm computes $v^{\#}$, the number of times that value appears in the domain of some variable in S. Denote by S' the set of all variables in S for which the sum $\sum_{v \in domain(S_i)} v^{\#}$ is maximized. The CP algorithm randomly select a variable from S'.

Value Selection. Let s be the chosen variable. The partial route of s is defined as the sequence of vertices $v_i, v_{i+1}, \ldots, v_j$ such that the successor $s[v_k]$ of v_k is v_{k+1} and $v_j = s$. The value-selection heuristic considers the following vertices in sequence:

1. a delivery vertex whose corresponding pickup vertex is in the partial route of s;
2. a pickup vertex randomly selected from the domain of s;
3. a delivery or a depot vertex.

Filtering Algorithms. Berbeglia et al. [5] developed two dedicated filtering algorithms for DARPs. The first filtering algorithm is based on solving exactly the Pickup and Delivery Problem with Fixed Partial Routes (PDP-FPR), a relaxed version of the DARP. The PDP-FPR takes into account the precedence and the capacity constraints and is strongly NP-complete [6]. The authors proposed a dynamic-programming algorithm to solve it exactly and use that to develop a filtering algorithm for PDP-FPR.

The second filtering algorithm is a partial filtering algorithm for the basic DARP with Ride Time Constraint problem, also a relaxation of the original problem with only the ride time constraint which is NP-complete [6]. For every unassigned successor variable s, the algorithm examines every pickup vertex p in the partial route of s and calculates a lower bound on the minimum time needed to get from p to the corresponding delivery vertex d. If this bound exceeds the maximum ride time, then some values from the domain of s can be removed. A similar procedure is executed for the delivery vertices in the partial routes of the vertices in the domain of s. The filtering algorithms are too complex to be described given space constraints but readers can consult [5] for the full details.

4 A Tabu Search Approach

A Tabu-Search approach was developed by Cordeau et al. [7]. The algorithm starts with a random initial solution s_0 and, at every iteration t, moves from solution s_t to a solution in its neighborhood. To prevent cycling, certain attributes of previous solutions are declared tabu unless those attributes form part of a new best solution. A diversification mechanism is in place to reduce the likelihood of being trapped in a local minimum. In addition, every κ iterations, every request is sequentially removed from its current route and inserted in the best possible location. Some important aspects of the algorithm are briefly described 1below.

Relaxation Mechanism. One of the key features of the tabu-search algorithm is that it allows the exploration of infeasible solutions. The time window, ride time, capacity, and route duration constraints are relaxed and their violation is penalized in the objective. The objective is defined as $f(s) = c(s) + \alpha q(s) + \beta d(s) + \gamma w(s) + \tau t(s)$ where $\alpha, \beta, \gamma, \tau$ are self-adjusting positive parameters, $c(s)$ is the routing cost, $q(s)$ the load violation, $d(s)$ the route duration violation, $w(s)$ the time window violation, and $t(s)$ the ride-time violation. The search tries to minimize the routing cost and the violations simultaneously to get good solutions that satisfy all the constraints.

Neighborhood. The neighborhood of a solution consists of moving a request i from a route r to a route r'. In such a case, the attribute (i, r) is put in the tabu list. If an attribute (i, r') is in the tabu list, then the request i cannot be moved to route r'. As a form of aspiration, if moving request i to route r' would result in a smaller cost than the best known solution *which has request i in route r'*, then the tabu status of the attribute (i, r') is revoked.

Penalty Adjustment. The penalties for the violations are adjusted dynamically through the course of the search. At every iteration, if a constraint is being violated in the current solution, the penalty for that constraint is multiplied by a factor $(1 + \delta)$ $(\delta > 0)$. If on the other hand, the constraint is not violated, the penalty is divided by the same factor. If a penalty reaches a fixed upper bound, then it is reset to 1.[1]

[1] This particular aspect is not mentioned in [7] but was learned through personal communication.

Neighborhood Evaluation. Cordeau et al. [7] uses three different schemes for choosing where to insert a request on a route. The simplest one only minimizes the time-window violations. The second does the same and also minimizes the route duration violations without increasing the ride-time violations. Both are linear time algorithms. The third evaluation procedure minimizes first the time-window violations, then the route duration violations and then the ride-time violations without increasing the time window or route duration violations. It is a quadratic time procedure. To reduce the size of the neighborhood, the algorithm first looks for the best insertion place for the critical vertex (ride-time violations are ignored in this step) and then the best insertion place for the non-critical vertex, while keeping the critical vertex in its best insertion place. In particular, different insertion places for the critical vertex are not considered.

5 A Variable Neighborhood Approach

A Variable Neighborhood Search (VNS) procedure for the DARP was proposed by [11]. The search starts with an initial solution s_0 generated by taking into account the spatial and temporal closeness of vertices. Then, at every iteration t with solution s_t, a random solution s' is generated in the neighborhood $N_k(s_t)$ in a step called *shaking*. Here k indicates which neighborhood is being used. The heuristic uses three different types of neighborhoods with multiple neighborhood sizes for a total of 13 different neighborhoods. Following that, a local search step is applied to s' to get solution s''. A simulated annealing type criterion is used to decide whether s'' replaces s_t and become the new incumbent solution. If s_t is not replaced, the next (larger) neighborhood is tried. Otherwise, s'' replaces s_t and the search begins with the first neighborhood, i.e., k is reset to 1. If k reaches 13, the maximum number of neighborhoods, it is also reset to 1. Infeasible solutions are also permitted in this framework and they are incorporated into the objective as in [7]. The neighborhood evaluation is also the same. Their results are competitive with the results obtained by [7]. A few other important aspects of the solver are highlighted below.

Neighborhood Structure. Three different types of neighborhoods are employed. In the *swap* neighborhood, two sequences of requests are chosen from two randomly selected routes. Those requests are then ejected from their current route and inserted in the other selected route in the best possible position. The *chain* neighborhood applies the ejection chain idea [9]. First two routes are randomly chosen and a sequence of requests is ejected from the first route and inserted in the best possible way in the second route. Then a sequence of requests which would decrease the evaluation function value *of that route* the most is ejected from the second route and moved to a third route (which may even be the first route). This last step is repeated a fixed number of times. The size of the sequences is also fixed. The third type of neighborhood is the *zero-split* neighborhood which is parameterless. Define a natural sequence to be one where the load at the beginning and end of the sequence is zero. Then the neighborhood is based on the idea that quite often multiple such natural sequences exist in routes. Thus a random number of such natural sequences are ejected from a route. Each of them is then inserted independently in a random route at their best insertion point. By varying the parameters of

the first two neighborhoods, along with the zero-split neighborhood, a sequence of 13 different neighborhoods is obtained.

Local Search. After the shaking step, a local search step is applied. Requests are sequentially removed from their current position and inserted in the first position that would improve the route's evaluation function value. If no such position exists for a request, then the request is kept at its original place. Since this procedure is time-consuming, it is only called if the solution after the shaking step is considered a promising solution, i.e., a solution that has a good possibility of becoming the new incumbent solution. Further details are in [11].

6 The LNS-FFPA Algorithm

The large neighborhood search algorithm (LNS-FFPA) is the main contribution of the paper. LNS-FFPA, where FFPA stands for First Feasible Probabilistic Acceptance, is inspired by the LNS algorithm described in [12,2,3] to minimize the travel distance for vehicle routing problems in [12] and pickup and delivery problems in [2]. However, LNS-FFPA contains some novel design decisions which are key to obtaining high-quality solutions on DARPs.

The Model. Each vertex v_i has a successor variable s_i. The routes are constructed by inserting a non-scheduled request r_i with pickup vertex v_i and delivery vertex v_j in the route. The pickup vertex is inserted in between two other vertices v_p and v_s which are parts of a route and similarly for the delivery vertex. In other words, each branching decision in LNS-FFPA corresponds to the insertion of a request in a route. Every time LNS-FFPA branches, it adds the following constraints for both the pickup and the delivery vertex of the request

$$b_i \geq b_p + d_p + t_{p,i}$$

$$b_s \geq b_i + d_i + t_{i,s}$$

where b_i is the serving time of the pickup or delivery vertex in question, d_i is the serving duration of a vertex and $t_{i,j}$ is the distance between two vertices as specified in Section 2. These constraints are removed upon backtracking.

The Feasibility Search. At a high level, LNS-FFPA is a constraint-programming search to find a feasible solution to DARPs, coupled with a large neighborhood algorithm to minimize travel distance. Algorithm 1 describes the algorithm for finding feasible solutions. The algorithm receives a partial solution, i.e., a set of partial routes for the vehicles. As long as there are unassigned customers, the algorithm selects such a request r (line 3). It then considers all its possible insertion points (line 4) and calls the algorithm recursively for each such insertion point p (line 6–8). If the recursive call finds a feasible solution, the algorithm returns. Otherwise, it removes the request and tries the remaining insertion points. Note that the insertion points are explored in increasing order of $e(r, p)$ which is defined as (α and β are positive constants)

$$\alpha \cdot costIncrease(r, p) - \beta \cdot slackAfterInsertion(r, p)$$

Algorithm 1. Tree-Search($PartialSolution$)

```
1: if no unassigned requests left then
2:     return PartialSolution
3: r ← GetUnassignedRequest()
4: for all feasible insertion points p for r in increasing order of e(r, p) do
5:     Insert r at point p in the PartialSolution
6:     ret = Tree-Search(PartialSolution)
7:     if ret is a solution then
8:         Return ret {Feasible solution found in sub-branch}
9:     Remove r from PartialSolution
10: return False {No feasible solution found for this sub-branch}
```

The Algorithm for Finding a Feasible Solution Given a Partial Solution.

Algorithm 2. GetUnassignedRequest()

1: $S_1 \leftarrow \{r : r$ is an unassigned request and the number of routes in which r can be inserted is minimized$\}$.
2: $S_2 \leftarrow \{r : r \in S_1$ and the number of insertion points for r is minimized$\}$.
3: $S_3 \leftarrow \{r : r \in S_2$ and the best insertion point for r increases $e(r, p)$ by the least amount$\}$.
4: **return** a randomly chosen element from S_3.

Request Selection Heuristic

where $costIncrease(r, p)$ denotes the increase in routing cost produced by inserting request r at insertion point p and $slackAfterInsertion(r, p)$ denotes the gap between the serving times of the pickup and delivery vertices and their successors and predecessors after the insertion. The gap for a vertex v_i is given by

$$servingTime[succ(v_i)] - servingTime[v_i] + servingTime[v_i] - servingTime[pred(v_i)]$$

and the gap for the pickup and delivery vertices is the sum of the gaps of the individual vertices. In other words, the insertion points are chosen to minimize the increase in the routing cost and maximize the available slack.

Algorithm 2 specifies which requests are inserted first, i.e., how line 3 in Algorithm 1 is implemented. It selects a request which can be inserted in the fewest vehicles (set S_1), which has the fewest insertion points (set S_2), and whose best insertion point produces the smallest amount in objective value.

Algorithm 1 is used both for finding an initial solution and for reinserting vertices during the large neighborhood search. In [12,2,3], the corresponding algorithm uses Limited Discrepancy Search (LDS) and limits the number of feasible insertion points explored at every search node. Moreover, such a neighborhood search is constrained to produce only improving solutions. In contrast, no such restrictions are imposed on Algorithm 1: It is a pure depth-first search algorithm, exploring all potential insertion points and returning the first feasible solution extending the input partial configuration.

For some instances with high (number of requests/number of vehicle) ratios, restarts improve performance: Algorithm 1 restarts after $max(\gamma \cdot m, \tau)$ failures where γ and τ are positive constants. This is only used for finding an initial feasible solution.

Algorithm 3. MinimizeRoutingCost($s, maxSize, range, numIter, timeLimit, d$)

```
 1:
 2: best ← s
 3: current ← s
 4: for i ← 2; i ≤maxSize-range; i ← i + 1 do
 5:    for j ← 0; j ≤range; j ← j + 1 do
 6:       for k ← 0; k ≤numIter; k ← k + 1 do
 7:          RelaxedSolution ← Randomly select i + j requests and
 8:          remove them from current
 9:          new ← Tree-Search(RelaxedSolution)
10:          pr ← random number between 0 and 1
11:          if f(new) < f(current) OR pr < d then
12:             current = new
13:             if f(current) < f(best) then
14:                best = current
15:          if timeLimit reached then
16:             return best
17: return best
```

The LNS-FFPA Algorithm for DARPs.

The Large Neighborhood Search **Algorithm 3** describes the LNS-FFPA algorithm to minimize the routing cost. It takes as input an initial feasible solution s and an upper bound on the number of requests that can be relaxed *maxSize*. To explore smaller neighborhoods first, LNS-FFPA uses a parameter *range* to increase the neighborhood size progressively. Finally, the procedure receives as inputs the number of iterations per neighborhood (t), the time limit for running the algorithm (*timeLimit*) and the probability d of accepting a worse solution. In addition, the function f used in the procedure returns the routing cost of a solution. The current solution is first initialized to the initial solution passed in to the procedure (line 3). Then the neighborhood is explored as given in lines 4-6. The number of requests that can be relaxed is steadily increased (line 4). Once it reaches the upper bound, it is effectively reset to 1. For a particular neighborhood size, a small range of neighborhoods starting from that size are explored (line 5). Every neighborhood size in that range is explored for *numIter* iterations (line 6). The number of requests equal to the neighborhood size are relaxed (line 7). The requests to relax are chosen at random. More sophisticated methods to select the requests to relax including the one used in [12,2,3] were tried but the random heuristic was significantly better for DARPs. Our conjecture is that the side constraints in DARPs, in particular the ride time, make it much harder to select a set of spatially related requests that could lead to a better solution than for more traditional VRPs without the ride constraint [12,2,3]. The search then attempts to complete the relaxed solution by calling Algorithm 1 to find a satisfying solution (line 9). The current solution is replaced by the new solution (which might be the same as the old solution) if either (1) the routing cost of the new solution is lower than the current solution; or (2) with some probability d (line 10). If the current solution is better than the best solution, then the best solution is updated (line 13). At the end or if the time limit is reached, the best solution found is returned (line 14-15 and line 16).

LNS-FFPA has some unique features compared to the standard LNS algorithms. First, during the neighborhood exploration, LNS-FFPA does not search for a solution with a routing cost better than the current solution, just a feasible solution. This diversifies the search and, equally importantly, enables LNS-FFPA to explore many reinsertions effectively. Indeed, since the selection of the requests to relax is randomized, it is not very likely that the search can discover better solutions for a given reinsertion set. Hence, it is not cost-effective to explore the sub-neighborhood exhaustively in the hope of finding a better solution. We could limit the number of insertion points per requests as is done in [12,2,3] but the algorithm would still take significant time on unsuccessful searches, while reducing the probability of finding high-quality solutions. Instead, we simply let Algorithm 1 find the first feasible, but not necessarily improving, solution, its variable and value heuristics guiding the search towards good solutions. Since finding a feasible solution is fast, LNS-FFPA explores many reinsertions, while providing a good diversification. This aspect is critical and led to significant improvements in quality, as will be demonstrated shortly.

7 Numerical Results

This section presents the experimental results, justifies the design decisions underlying our LNS-FFPA algorithm, and compares the algorithm with prior algorithms.

The Algorithms. We compared our LNS-FFPA algorithm against the CP approach of Berbeglia et al. [5], the tabu search by Cordeau et al. [7], and the variable neighborhood search by Parragh et al. [11]. For the parameters for Algorithm 1, we set $\alpha = 80$ and $\beta = 1$. For the parameters for the restart strategy, we set $\gamma = 200$ and $\tau = 1000$. For the LNS Search, we set $maxSize = n/2$, where n is the number of requests, $range = 4$, $numIter = 300$, and $d = 0.07$.

The LNS-FFPA and Variable Neighborhood Search algorithms[2] were tested on a Intel Core 2 Quad Q6600 machine with 3 GB of RAM. The Comet language was used to implement the LNS-FFPA algorithm and is in general 3–5x slower than comparable C++ code. As the code for VNS and Tabu Search is implemented in C++, we conservatively divide the amount of time LNS-FFPA takes by a factor of 3 for this evaluation.

The code for the the tabu search was unavailable and hence, we can only compare with the tables given in [7] which report results for only one or two runs of the algorithm.[3] This comparison is much less reliable than the comparison with the most recent Variable Neighborhood Search [11] but is given for completeness.

The Instances. The Dial-a-Ride instances are taken from Parragh et al. [11] and Cordeau et al. [7]. They are based on realistic assumptions and data provided by the Montreal Transit Commission (MTC). Half of the requests are outbound and half inbound. They are divided into classes a and b, the difference being that class a instances have tighter time windows. In the instances, m denotes the number of vehicles and n is the number of requests.

[2] Many thanks to Parragh et al. for providing us with the code for the VNS search.

[3] As the tabu search was run on a 2 GHZ machine, when it is compared with the LNS-FFPA algorithm, the time for LNS-FFPA is divided by 2.5 instead of 3.

Table 1. The Benefits of LNS-FFPA

5 minute run					
Class a		**LNS**		**LNS-FFPA**	
m	n	Mean	Best	Mean	Best
3	24	191.14	190.02	190.77	190.02
4	36	302.77	296.36	292.86	291.71
5	48	318.50	312.12	304.45	303.03
6	72	537.10	526.54	505.15	494.91
7	72	577.13	547.69	547.39	542.83
8	108	768.08	736.14	711.60	696.51
9	96	660.67	636.50	595.05	588.80
10	144	987.70	950.01	911.18	891.98
11	120	722.87	696.95	662.56	653.57
13	144	915.34	905.03	832.74	816.79
Avg.		598.13	579.74	555.38	547.02

5 minute run					
Class b		**LNS**		**LNS-FFPA**	
m	n	Mean	Best	Mean	Best
3	24	170.29	167.78	164.46	164.46
4	36	263.27	252.99	248.31	248.21
5	48	318.51	308.51	301.67	299.27
6	72	509.89	494.97	477.75	469.73
7	72	548.22	530.45	504.69	494.01
8	108	682.50	647.85	633.51	620.54
9	96	611.66	595.32	566.48	557.61
10	144	952.60	918.76	857.95	838.65
11	120	671.87	650.27	610.33	602.19
13	144	870.30	846.16	785.13	771.69
Avg.		559.91	541.31	515.03	506.64

The Benefits of LNS-FFPA. Before comparing LNS-FFPA with prior art, it is useful to evaluate our main design decision and compare LNS and LNS-FFPA. Standard LNS algorithms (e.g., [12,3,2]) always search for an improving solution and limit the number of insertion points to explore the "good" parts of the subproblems. In contrast, LNS-FFPA does not require the subproblem to find an improving solution: It simply searches for the first feasible solution to the subproblem using the heuristic to drive the search toward a high-quality solution. Moreover, LNS-FFPA may accept the solution to the subproblem even if it degrades the best-known solution, using a Probabilistic acceptance criterion.

Table 1 compares LNS and LNS-FFPA and reports the mean and best solutions found over 10 different 5-minute runs for each instance. In the table, m denotes the number of vehicles and n the number of requests. The experimental results indicate that LNS-FFPA leads to solutions of significantly higher quality and to a more robust algorithm. The average improvement for the Class a instances is about 8% and is higher for the larger instances. For Class b, the average improvement is also around 8% and, for larger instances, improvement of almost 10% are observed. It is also important to stress how robust LNS-FFPA is, since the difference in quality between the best and the average solutions is rather small.

Table 2 evaluates the impact of the two novel aspects of LNS-FFPA: It reports the results of LNS-FF which never accepts any worse solution. The results show that the two additional components of LNS-FFPA are complementary but with the First Feasible criterion having a slighly larger effect. Indeed, without Probabilistic acceptance criterion, the average improvement drops from 8% to 4.2% on Class a instances and from 8% to 4.5% on Class b.

It is also important to mention that simply adding the Probabilistic criterion to the standard LNS was not effective. In other words, accepting the best solution found in the neighborhood with a Probabilistic criterion actually deteriorated performance, indicating that it is the combination of stopping at the first feasible solution and using the Probabilistic criterion which is key to obtain enough search diversity. A potential

Table 2. The Impact of the Acceptance Criterion

5 minute run					
Class a		**LNS**		**LNS-FF**	
m	n	Mean	Best	Mean	Best
3	24	191.14	190.02	190.224	190.019
4	36	302.77	296.36	297.419	293.038
5	48	318.50	312.12	307.449	304.051
6	72	537.10	526.54	518.945	507.672
7	72	577.13	547.69	556.872	546.893
8	108	768.08	736.14	736.739	714.860
9	96	660.67	636.50	626.956	598.675
10	144	987.70	950.01	937.208	912.302
11	120	722.87	696.95	685.016	670.116
13	144	915.34	905.03	875.007	849.761
Avg.		598.13	579.74	573.18	558.74

5 minute run					
Class b		**LNS**		**LNS-FF**	
m	n	Mean	Best	Mean	Best
3	24	170.29	167.78	166.57	164.46
4	36	263.27	252.99	257.02	255.96
5	48	318.51	308.51	309.43	299.02
6	72	509.89	494.97	488.90	478.19
7	72	548.22	530.45	526.87	511.35
8	108	682.50	647.85	645.20	624.61
9	96	611.66	595.32	593.90	574.23
10	144	952.60	918.76	903.58	883.29
11	120	671.87	650.27	640.77	615.96
13	144	870.30	846.16	811.58	795.23
Avg.		559.91	541.31	534.38	520.28

explanation is that LNS-FFPA can exploit the diversification of accepting a worse solution a lot better since it explores a lot more neighborhoods whereas the standard LNS algorithm spends too much time trying to find a better solution in fewer neighborhoods.

Overall, these results show that LNS-FFPA is a critical aspect of this research. For Dial-a-Ride problems, more diversification is key to improving quality. This diversification can be obtained either by accepting worse solutions or by exploring more neighborhoods since the search terminates as soon as a feasible solution is found.

Comparison with the Variable Neighborhood Search. We now compare LNS-FFPA with the state-of-the-art Variable Neighborhood Search (VNS) of Parragh et al. [11]. Table 3 depicts the results for class a instances.[4] Except for small instances with three vehicles, LNS-FFPA produces results that are consistently better on average and frequently better in terms of the best solutions. As the n/m ratio rises, the difficulty and the instance size increase and the LNS-FFPA produces increasing benefits. For the 1.6 min runs, LNS-FFPA improves the quality of the solution by 6.1% in average and by 24.2% in the best case. For the 5 min runs, LNS-FFPA produces improvement of about 3% in average and 14% in the best case.

These results indicates that LNS-FFPA is a very effective approach to find high-quality solutions under severe time constraints to complex Dial-a-Ride problems.

Comparison with the Tabu Search. Table 4 compares LNS-FFPA and the tabu search of [7] on short runs for the classes a and b. As mentioned earlier, Cordeau did not release his algorithm whose results seem very hard to reproduce. The table reports the tabu-search results as given in [7] where the quality of a single solution, and the time to obtain it, are given. The results for LNS-FFPA are obtained by snapshots of the execution, selecting the best-found solution within the time reported by the tabu search. The tabu

[4] A comparison against the class b instances was not possible as the solver seemed to require some user interaction during the search on those instances.

Table 3. Comparing VNS and LNS-FFPA

1.6 minute run					
Class a		**VNS**		**LNS-FFPA**	
m	n	Mean	Best	Mean	Best
3	24	190.02	190.02	191.02	190.79
4	36	294.42	291.71	294.00	291.71
5	48	306.10	302.45	305.30	303.39
6	72	507.54	501.31	506.65	494.91
7	72	553.85	536.23	548.76	542.94
8	108	843.64	783.24	723.64	699.95
9	96	611.86	592.91	607.06	597.98
10	144	1223.18	1189.36	926.98	909.51
11	120	724.52	681.1	667.45	655.16
13	144	991.18	976.85	856.84	846.56
Avg.		624.63	604.48	562.77	553.29

5 minute run					
Class a		**VNS**		**LNS-FFPA**	
m	n	Mean	Best	Mean	Best
3	24	190.02	190.02	190.77	190.02
4	36	293.77	291.71	292.86	291.71
5	48	305.84	302.45	304.45	303.03
6	72	507.21	501.31	505.15	494.91
7	72	552.54	536.23	547.39	542.83
8	108	730.48	701.71	711.60	696.51
9	96	610.30	602.40	595.05	588.80
10	144	1059.52	1021.72	911.18	891.98
11	120	686.11	672.23	662.56	653.57
13	144	885.67	869.56	832.74	816.79
Avg.		582.15	568.93	555.38	547.02

Table 4. Comparing Tabu Search and LNS-FFPA

Class a		**Tabu**		**LNS-FFPA**	
m	n	1 Run	Time	Mean	Mean Time
3	24	191.05	0.19	191.13	0.40
4	36	292.80	0.44	296.99	0.40
5	48	304.04	0.81	306.73	0.80
6	72	506.62	2.4	506.65	2.40
7	72	550.48	1.72	549.84	1.60
8	108	732.12	5.51	711.60	5.20
9	96	597.32	2.88	602.15	2.80
10	144	933.22	8.75	909.17	8.40
11	120	691.55	4.62	664.58	4.40
13	144	870.66	5.39	834.40	5.20
Avg.		566.99	3.27	557.32	3.16

Class b		**Tabu**		**LNS-FFPA**	
m	n	1 Run	Time	Mean	Mean Time
3	24	165.31	0.19	167.67	0.40
4	36	253.04	0.42	255.45	0.40
5	48	304.73	0.83	305.99	0.80
6	72	495.31	2.29	480.27	2.00
7	72	510.86	1.85	509.38	1.60
8	108	657.96	5.13	632.89	4.80
9	96	563.24	3.12	570.02	2.80
10	144	909.58	9.24	857.49	9.20
11	120	615.36	5.43	616.31	5.20
13	144	810.65	7.37	788.33	7.20
Avg.		528.60	3.59	518.38	3.44

search is slightly better on the small instances. However, as the instances get larger and harder with the n/m ratio increasing, LNS-FFPA gives much better results. In the best case, LNS-FFPA produces a 5.7% improvement while producing a 1.1% improvement on average. However, when restricting attention to larger instances ($m > 5$), LNS-FFPA algorithm produces an improvement of about 2% which becomes 3% if the largest three instances are considered. Given the high-quality and sophistication of both the VNS and the tabu-search algorithm, these improvements are significant.

These results are particularly appealing given the severe time constraints. For many problems, a dedicated local search produces solutions of higher quality than LNS early on, since LNS is a general-purpose technique on top of an existing optimization algorithm and does not have dedicated neighborhood operators. On Dial-a-Ride problems however, LNS-FFPA produces better solutions than highly-tuned tabu search or variable neighborhood search within short time limits, especially on the large instances.

This gives LNS-FFPA a significant advantage in dynamic settings since high-quality solutions would need to be found quickly in that scenario. A potential explanation is that the complexity of the side-constraints increases the cost of local moves in tabu and variable neighborhood searches, making LNS-FFPA very competitive.

Comparison with the Constraint-Programming Approach. We conclude this section with a comparison to the constraint-programming approach of Berbeglia et al. [5] who report the time taken to find a satisfying solution for their CP solver and for the tabu-search solver. These feasibility problems are tested on different instances: They have vertices located in a $[-20, 20]^2$ square, over a time horizon of 12 hours, with time windows of 15 minutes, and vehicle capacity of 3 for instances from set a and 6 for instances from set b. The ride time is 30 minutes. The time taken by Algorithm 1 to find a satisfying solution is 0.5-2 seconds which is comparable to the time taken by tabu search and, on average, 12 times faster than the CP approach of Berbeglia et al [5]. The exception is instance b5-40 where it cannot find a solution for a time limit of 60s. Algorithm 1 can also detect infeasibility in less than a second for all the infeasible instances described in [5] which were obtained by reducing the maximum ride time.

8 Conclusions and Future Work

This paper considered Dial-a-Ride applications, which are complex multiple-vehicle routing problems with pickups and deliveries, time windows, and constraints on the ride time. Moreover, these applications are typically dynamic, as customers dial for rides. As a result, optimization algorithms must return high-quality solutions quickly.

The paper presented a novel large neighborhood search LNS-FFPA, which contains two key technical contributions. First, LNS-FFPA does not search the neighborhoods for improving solutions and rather returns the first feasible solution. Second, such a feasible solution is accepted if it improves the existing incumbent solution or using a Probabilistic criterion.

Experimental results for benchmarks based on realistic assumptions and data provided by the Montreal Transit Commission (MTC) show the effectiveness of LNS-FFPA. On short runs (of 1.6 and 5.0 minutes), LNS-FFPA significantly outperforms the state-of-the-art VNS and tabu search algorithms which are both rather sophisticated. LNS-FFPA also compares very favourably with the constraint-programming approach for finding feasible solutions, often producing significant improvements in efficiency. The experimental results also demonstrate the benefits of LNS-FFPA over a traditional LNS approach, as it improves solution quality by about 8%. Finally, LNS-FFPA was particularly effective on the largest instance, where its benefits are larger.

Future work will study if the spatial and temporal structure of Dial-A-Ride applications can be exploited in the choice of the neighborhood instead of relying on pure random selections. Moreover, it would be interesting to study the dynamic problem in the framework of online stochastic optimization to evaluate if stochastic information would be valuable in this setting.

References

1. Baugh, J., John, W.K., Reddy, G.K., Stone, J.R.: Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. Engineering Optimization 30, 91–123 (1998)
2. Bent, R., Hentenryck, P.V.: A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Comput. Oper. Res. 33, 875–893 (2006)
3. Bent, R., Van Hentenryck, P.: A two-stage hybrid local search for the vehicle routing problem with time windows. Transportation Science 38, 515–530 (2004)
4. Berbeglia, G., Cordeau, J.-F., Laporte, G.: A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. Submitted to INFORMS Journal on Computing (2010)
5. Berbeglia, G., Pesant, G., Rousseau, L.-M.: Checking the feasibility of dial-a-ride instances using constraint programming. Transportation Science (2010)
6. Berbeglia, G., Pesant, G., Rousseau, L.-M.: Feasibility of the pickup and delivery problem with fixed partial routes: A complexity analysis. Submitted to Transportation Science (2010)
7. Cordeau, J.-F., Laporte, G.: A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B: Methodological 37(6), 579–594 (2003)
8. Cordeau, J.-F., Laporte, G.: The dial-a-ride problem: models and algorithms. Annals of Operations Research 153, 29–46 (2007), 10.1007/s10479-007-0170-8
9. Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics 65(1-3), 223–253 (1996); First International Colloquium on Graphs and Optimization
10. Jorgensen, R.M., Larsen, J., Bergvinsdottir, K.B.: Solving the dial-a-ride problem using genetic algorithms. Journal of the Operational Research Society 58(11), 1321–1331 (2007)
11. Parragh, S.N., Doerner, K.F., Hartl, R.F.: Variable neighborhood search for the dial-a-ride problem. Computers & Operations Research 37(6), 1129–1138 (2010)
12. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M.J., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)
13. Toth, P., Vigo, D.: Heuristic algorithms for the handicapped persons transportation problem. Transportation Science, 60–71 (1997)