View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Computers & Operations Research 40 (2013) 490–497

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Hybrid column generation and large neighborhood search for the dial-a-ride problem

Sophie N. Parragh [a], Verena Schmid [a,b,*]

[a] Department of Business Administration, University of Vienna, Vienna, Austria
[b] Centro para la Optimizacióny Probabilidad Aplicada (COPA), Departamento de Ingeniería Industrial, Universidad de los Andes, Cr 1E No. 19A-10, ML315, Bogotá, Colombia

ARTICLE INFO

Available online 10 August 2012

Keywords:
Large neighborhood search
Column generation
Hybrid algorithm
Variable neighborhood search
Dial-a-ride problem

ABSTRACT

Demographic change towards an ever aging population entails an increasing demand for specialized transportation systems to complement the traditional public means of transportation. Typically, users place transportation requests, specifying a pickup and a drop off location and a fleet of minibuses or taxis is used to serve these requests. The underlying optimization problem can be modeled as a dial-a-ride problem. In the dial-a-ride problem considered in this paper, total routing costs are minimized while respecting time window, maximum user ride time, maximum route duration, and vehicle capacity restrictions. We propose a hybrid column generation and large neighborhood search algorithm and compare different hybridization strategies on a set of benchmark instances from the literature.

© 2012 Elsevier Ltd. Open access under CC BY-NC-ND license.

1. Introduction

Demand responsive transportation services are needed, e.g. in remote rural areas, where no general public transportation systems exist, as a complementary service to available public transportation systems for the elderly or disabled, or in the area of patient transportation to and from hospitals and other medical facilities. All these services involve the transportation of persons who place transportation requests, specifying an origin and a destination location. The underlying optimization problem is usually modeled in terms of a dial-a-ride problem (DARP). The field of DARPs has received considerable attention in the literature. However, due to the application oriented character of this problem, the objectives considered as well as the constraints imposed vary considerably; rather recent surveys covering DAPRs and demand responsive transportation are due to Cordeau and Laporte [1] and Parragh et al. [2].

In the DARP under consideration in this paper, the objective corresponds to the minimization of the total routing costs. A homogeneous fleet of vehicles of size m has to serve a given set of transportation requests n. These are all known in advance of the planning. In the following, we will refer to the origin or pickup node of a request i by i, and to its destination or drop off node by $n+i$. Users specify time windows for either the origin or the destination. In addition, maximum user ride times, route duration limits, and vehicle capacity constraints have to be considered in the planning.

This version of the DARP has been considered by Cordeau and Laporte [3], who propose a tabu search algorithm and a set of 20 benchmark instances, by Parragh et al. [4], who develop a competitive variable neighborhood search (VNS) heuristic, and by Jain and Van Hentenryck [5], who propose a constraint programming based large neighborhood search algorithm. A formal definition of the problem can be found in [6], where a branch-a-cut algorithm is proposed that solves instances with up to 36 requests. Ropke et al. [7] propose two new two-index formulations and a number of additional valid inequalities which are used within branch-and-cut algorithms. Instances with up to 8 vehicles and 96 requests are solved to optimality. In [8], the same instances are solved by means of branch-and-cut-and-price.

Since we consider a route duration limit in combination with a time window at the depot and maximum user ride times in connection with time windows at either the pickup or the drop off location, the scheduling subproblem (and hence also the feasibility check) is more involved than in other routing problems. Cordeau and Laporte [6] propose an eight step evaluation scheme to determine the feasibility of a given route. We use this scheme in the revised version of [4]. The evaluation scheme is based on the forward time slack as introduced by Savelsbergh [9] which is first used to reduce the duration of the tour and then to reduce the individual ride time of each request on the tour. If l gives the length in terms of the number of nodes along the tour the forward time slack computation is of complexity $O(l)$ [1].

In recent years, the field of hybrid metaheuristics, and matheuristics in particular, has received more and more attention

* Corresponding author at: Department of Business Administration, University of Vienna, Vienna, Austria.
E-mail addresses: sophie.parragh@univie.ac.at (S.N. Parragh), verena.schmid@univie.ac.at (V. Schmid).

0305-0548 © 2012 Elsevier Ltd. Open access under CC BY-NC-ND license.
http://dx.doi.org/10.1016/j.cor.2012.08.004

[10,11]. In the field of vehicle routing, metaheuristic and column generation hybrids have shown to be especially successful: Prescott-Gagnon et al. [12], e.g., propose a branch-and-price based large neighborhood search algorithm for the vehicle routing problem with time windows; heuristic destroy operators are complemented by a branch-and-price based repair algorithm. Muter et al. [13], on the other hand, propose a hybrid tabu search heuristic, where the column pool is filled with feasible routes identified by the tabu search. The search is then guided by the current best lower and upper bound; the current best lower bound is obtained from solving the linear relaxation of a set covering type formulation on the current column pool; the current best upper bound is computed by imposing integrality on the decision variables. Both methods are tested on benchmark instances for the vehicle routing problem with time windows. Using related ideas, Pirkwieser and Raidl [14] propose variable neighborhood search ILP hybrid for the periodic vehicle routing problem with time windows: feasible solutions generated by the metaheuristic are used to populate the column pool. A set covering problem (SCP) is iteratively solved on this pool and in case of improvement, the current solution of the VNS is replaced by the solution of the SCP.

In line with these developments, we propose a hybrid method for the DARP. It integrates VNS into column generation and combines it with large neighborhood search (LNS). LNS, as a stand-alone method, has shown to work well when applied to routing problems in general [15], and for the pickup and delivery problem with time windows in particular [16] (a problem closely related to the DARP). Following recent developments in the column generation field [17] and given its success in solving difficult routing problems [18,19], we propose a VNS based column generator to identify additional routes.

The remainder of this paper is organized as follows. Section 2 is devoted to a detailed presentation of the proposed hybrid framework. This is followed by computational experiments, illustrating the merits of each of the components. Conclusions and directions for future research are given at the end of the paper.

2. Solution framework

The proposed hybrid framework consists of two main algorithmic components: LNS and column generation. These two components are described in the following. Thereafter, their combination is illustrated in further detail.

2.1. Large neighborhood search

LNS has been introduced by Shaw [20]. Its principle is relatively simple: in each iteration the incumbent solution is partially destroyed and then it is repaired again; that is, first a given number of elements are removed and then they are reinserted. Every time these operations lead to an improved solution, the new solution replaces the incumbent solution, otherwise it is discarded. Ropke and Pisinger [16] extend Shaw's idea and they propose to use a number of different destroy and repair operators. Given the success of this method, all our operators are either based on or correspond to the ones employed in [16]. In terms of destroy operators these are the random removal operator, the worst removal operator, and the related removal operator; in terms of repair operators, these are a greedy insertion heuristic, and k-regret insertion heuristics. In contrast to [16], we do not use an adaptive layer to guide the selection of the operators but we choose them randomly. The reason for this design decision is the following. Our aim is to keep each component of our hybrid framework as simple as possible and an adaptive layer comes at the price of a large number of additional parameters, compared to only small gains in solution quality.

In every iteration of the proposed LNS, before a removal operator is applied to the incumbent solution, the number of requests to be removed q has to be determined. In our case, in each iteration, q is chosen randomly between $0.1n$ and $0.5n$. Then, one of the destroy operators is selected randomly and applied to the current incumbent solution.

The random removal operator randomly removes q requests. The worst removal operator randomly removes requests while biasing the selection towards requests whose removal would improve the objective function value the most. Finally, in the related removal operator the selection of requests is biased towards related requests. We use a slightly different similarity measure than the one employed in [16]. It has the advantage of being parameter free but it is not only based on the distance between two requests, as proposed in [21]. Two requests i and j are said to be related if $(|B_i-B_j|+|B_{n+i}-B_{n+j}|+t_{ij}+t_{n+i,n+j})$ is small; t_{ij} denotes the distance between location i and j; and B_i the beginning of service at i. In every iteration of the related removal operator, we bias the choice towards those requests that are the most similar to all requests that have already been removed.

Removed requests are put into the request bank and, in a next step, they are reinserted using one of the repair operators. We randomly choose a repair operator among greedy insertion, 2-regret insertion, 3-regret insertion, 4-regret insertion and m-regret insertion.

Using the greedy insertion heuristic, in each iteration, the unserved request that deteriorates the objective function value the least is inserted at its best insertion position. The regret insertion heuristics work as follows: in each iteration the unserved request that is associated with the largest regret value is inserted at its best insertion position. Let $\Delta(i,r,s)$ denote the difference in objective function value if request i is inserted at its best position into route r of solution s. Now assume that $\Delta(i,1,s)$ is associated with the route where i can be inserted the cheapest, $\Delta(i,2,s)$ with the route where i can be inserted the second-cheapest, and so on. Then, the regret value $R(i)$ is computed as follows:

$$R(i)=\sum_{r=2}^{k}[\Delta(i,r,s)-\Delta(i,1,s)] \tag{1}$$

with $k\in\{2,3,4,m\}$, depending on the chosen version of the heuristic (2-regret, 3-regret, 4-regret, and m-regret). We refer to [16] for additional information.

In order to further diversify the search we allow solutions that deteriorate the incumbent solution by at most 3% to be accepted with a probability of 1%. In order to facilitate switching between LNS and other components, we refrain from using more sophisticated acceptance schemes. We note, however, that feasibility is maintained at all times: if requests cannot be inserted in a feasible way and thus remain in the request bank, the new solution is not considered for acceptance. Since deteriorating moves are allowed, besides the current incumbent solution, we also keep track of the best solution identified during the search.

Furthermore, following the findings of [16], in each iteration, we randomly choose if the selected repair operator is used in its deterministic or in its randomized version. If the randomized version is selected, every time the evaluation function is called, it randomly chooses a noise factor in [0.5, 1.5] and multiplies the original insertion costs by it.

Finally, like in [4], every time a new solution is generated and it is at most 5% worse than the current best solution, the new solution undergoes local search based improvement. Solutions

that are worse than that have a chance of 1% to undergo local search based improvement. The local search heuristic is of the intra-route, first improvement type. It proceeds sequentially along each route, considering one request at a time, trying to improve its positioning by moving the corresponding pickup and the corresponding delivery to a different position within their route; we refer to [4] for details on this procedure.

As indicated above, we hybridize LNS with ideas from column generation. These are described in the following.

2.2. Column generation

In order to use column generation based components, we formulate the DARP in terms of a SCP. Let Ω denote the set of feasible routes and let P denote the set of requests. For each route $\omega \in \Omega$, let c_ω be the cost of the route and let the constant $b_{i\omega}$ represent the number of times request $i \in P$ is traversed by ω. Binary variable y_ω takes value 1 if and only if route ω is used in the solution. The problem can thus be formulated as the following SCP:

$$\min \sum_{\omega \in \Omega} c_\omega y_\omega \tag{2}$$

subject to

$$\sum_{\omega \in \Omega} b_{i\omega} y_\omega \geq 1 \quad \forall i \in P, \tag{3}$$

$$\sum_{\omega \in \Omega} y_\omega \leq m, \tag{4}$$

$$y_\omega \in \{0,1\} \quad \forall \omega \in \Omega. \tag{5}$$

The objective function (2) minimizes total routing costs. Constraints (3) make sure that each request is covered by at least one route; and constraint (4) ensures that at most m vehicles are used in the solution. Replacing (5) by,

$$y_\omega \geq 0 \quad \forall \omega \in \Omega, \tag{6}$$

we obtain the linear relaxation of SCP denoted as LSCP.

Due to the large size of Ω, SCP cannot be solved directly. Instead, a restricted version of SCP, denoted as RSCP, considering only a small subset of columns $\Omega' \subset \Omega$, is solved. The set Ω' can be generated by means of column generation.

In column generation, in each iteration the linear relaxation of RSCP, denoted as RLSCP, is solved. The column or route that is associated with the smallest negative reduced cost value is searched and added to Ω'. The according problem is usually referred to as the subproblem whereas RLSCP is denoted as the master problem. The reduced cost of a newly found column can be computed as follows:

$$\bar{c}_\omega = c_\omega - \sum_{i \in P} b_{i\omega} \pi_i - \sigma, \tag{7}$$

where π_i denotes the dual variable associated with constraint (3) for request i, and σ the dual variable associated with constraint (4). The master and the subproblem are solved iteratively until no more negative reduced cost columns can be found. In this case, the optimal solution of LSCP has been found. For further information on column generation, we refer to the book by Desaulniers et al. [22].

The subproblem we have to solve minimizes $\bar{c}_\omega$ while taking into consideration pairing and precedence, time window, ride time, capacity and duration constraints. We propose to generate columns of negative reduced cost heuristically, by means of VNS. It is described in the following.

2.3. Variable neighborhood search

VNS has been introduced by Hansen and Mladenovic [18]. It is an iterative procedure, where within a *shaking* phase an incumbent solution is modified using a neighborhood structure $\mathcal{N}_k$ (where $k = 1, \ldots, k_{max}$). The resulting solution is further improved using local search. The neighborhoods are typically ordered in a nested way, i.e. the first (last) neighborhoods result in a comparatively small (large) modification respectively. If the current solution improves the incumbent solution it replaces the incumbent solution and the procedure starts over again with the first neighborhood. Otherwise the current solution is discarded and the procedure continues with the next neighborhood. The procedure stops as soon as a predefined termination criterion is met.

We apply VNS to individual routes: each route associated with a column that is part of the current basic solution of RLSCP (i.e. where $y_\omega > 0$) is considered consecutively. The objective is to find new columns with negative reduced cost which enrich the column pool Ω'. Throughout the application of VNS all obtained routes are evaluated and compared in terms of their reduced costs, where lower ones are favored.

The proposed neighborhood structures $\mathcal{N}_k$ (where $k = 1, \ldots, 9$) for the shaking phase may modify any given route in three different ways. In any iteration a given percentage of requests is *added*, *exchanged*, or *removed*. The selection probabilities for requests to be chosen in each neighborhood are biased with respect to the dual information of the underlying RLSCP. The selection probability is proportional to π_i (we will refer to this strategy as π afterwards). Any neighborhood affects the route by $20\lceil k/3 \rceil\%$, with respect to the current number of request pairs scheduled on it. Neighborhoods $k = 1,4,7$ *add* up to 20%, 40%, 60% of pairs currently not on route, neighborhoods $k = 2,5,8$ *exchange* the corresponding fraction of pairs currently scheduled on it. Finally neighborhoods $k = 3,6,9$ *remove* the required number of pairs from the route. By means of any shaking step we try to identify routes with negative reduced costs. When adding a request to a given route (i.e., in neighborhoods $k = 1,4,7$), the selection probability for any request will be directly proportional to its dual value π_i, hence favoring requests with a high π_i value. When removing requests ($k = 3,6,9$), the selection probability will be inversely proportional. In neighborhoods $k = 2,5,8$ a request will be removed in exchange for another one to be added; again, the selection probabilities will be determined according to the rules specified before. Only changes resulting in a feasible route are considered.

After having generated a route from the current neighborhood randomly it is further improved by means of the intra-route local search algorithm proposed in [4]. In case the resulting route improves the best currently known in terms of negative reduced cost, we start over with the first neighborhood $\mathcal{N}_1$, otherwise the route is discarded and we continue with the next neighborhood $k+1$. The procedure terminates as soon as N^{new} new columns with negative reduced cost have been found or exploiting the last neighborhood k_{max} did not result in a column with negative reduced cost (we will refer to this strategy as *addFirst* afterwards). Upon termination of VNS the obtained routes with negative reduced costs are added to Ω'.

2.4. Hybridization scheme

A sketch of the proposed hybridization scheme is outlined in Algorithm 1. In a first step, the column pool Ω' is initialized with single request routes (one for each request) that do not consume a vehicle resource. Each of these columns is associated with a cost of big M. Then, a feasible starting solution is generated by means of the 2-regret insertion heuristic. In the case where requests

remain in the request bank, a truncated version of the described LNS is run: destroy and repair operators are iterated until a first feasible solution is attained. This solution constitutes the starting solution and the according columns are added to the pool Ω'. Then, in each iteration, we first solve RLSCP on Ω' and we retrieve the according columns and the dual information. Thereafter, the above described VNS is used to identify new columns of negative reduced cost. It takes each route or column ω part of the current basis s' as a starting point and generates at most N^{new} new columns for each $\omega \in s'$. These are added to Ω'. Every $N^{interval}$ iterations, we also solve the RSCP on Ω' to obtain the best feasible DARP solution s^*, currently available. In case of duplicate requests, which is possible because we solve a set covering and not a set partitioning problem, the obtained solution is repaired. Duplicate requests are removed in a greedy way, i.e. we check each request at a time and in case it exists more than once, we keep the one that is inserted best and remove all others. The obtained solution serves as starting solution for LNS. Thus, compared to the column generation component based on VNS, LNS works on complete solutions rather than on individual routes or columns, hence taking into account a more global view on the solution. It is run for N^{LNS} iterations. All routes generated during the execution of LNS are transformed into columns and they are added to the pool Ω'. The above is repeated for $N^{iterations}$ iterations. In the end, RSCP is solved on the obtained column pool Ω' and the according result is returned.

Algorithm 1. Hybrid framework.

```
initialize column pool Ω'
generate a feasible starting solution s
add all ω ∈ s to Ω'
s_best := s; i := 0;
while i < N^iterations do
    solve RLSCP on Ω' yielding s'
    for each ω ∈ s' do
        apply VNS and add up to N^new columns with negative reduced costs to Ω'
    end for
    if (i mod N^interval) = 0 then
        solve RSCP on Ω' yielding s*
        apply LNS to s* for N^LNS iterations yielding s''
        s_best = s''
    end if
    i := i+1
end while
solve RSCP on Ω' yielding s_best
return s_best
```

3. Results

The algorithms were implemented in C++ and for the solution of the SCPs CPLEX 12.1 together with Concert Technology 2.9 was used. All tests were carried out on a Xeon CPU at 2.67 GHz with 24 GB of RAM (shared with 7 other CPUs) and CPLEX was restricted to one thread. In addition, each call to CPLEX was limited to 60 s. This may entail that the attained solution of RSCP is worse than the best solution found so far. In this case the search proceeds with the best found solution.

In the following we describe the test data set and then the results obtained. We first evaluate different parameter settings and hybridization schemes. In hybridization step one, we evaluate combinations of LNS with RSCP. In hybridization step two, we evaluate combinations of all three components: LNS, R(L)SCP and VNS. This is followed by a comparison of the results computed by the proposed hybrid framework to the best known results from the literature.

3.1. Test instances

Cordeau and Laporte [3] proposed a data set of 20 randomly generated instances. They contain 24–144 requests. In each instance, the first $n/2$ requests have a time window on the destination, while the remaining $n/2$ instances have a time window on the origin. For each vertex a service time $d_i=10$ is considered and the number of persons transported per request is set to 1. Routing costs and travel times from a vertex i to a vertex j correspond to the Euclidean distance between these two vertices. The route duration limit is 480, the vehicle capacity is 6, and the maximum ride time is 90, in all instances. In the first 10 instances, narrow time windows are considered. For the second 10 instances, wider time windows are given. The first six instances out of these 10 instances are characterized by a larger vehicle fleet, the remaining four instances have a smaller vehicle fleet. Based on this information, the data set can be subdivided into four sets with similar characteristics: R1a–R6a (narrow time windows, larger vehicle fleet); R7a–R10a (narrow time windows, smaller vehicle fleet); R1b–R6b (wider time windows, larger vehicle fleet); R7b–R10b (wider time windows, smaller vehicle fleet). In the following, we refer to these sets by A, B, C, and D respectively.

3.2. Evaluating hybridization step one

In a first step, we evaluate the performance of LNS in combination with intermediate calls to CPLEX, solving RSCP on the columns generated by LNS (LNS+SC) and using it as a stand-alone method, with and without local search. Following the findings of Ropke and Pisinger [16] in the context of the pickup and delivery problem with time windows, we decided that a total of 25,000 LNS iterations shall be attained in each setting. In our framework (with $N^{interval}=1$) there are two parameters which influence the total number of LNS iterations executed during one run. These are $N^{iterations}$ and N^{LNS}. In order to obtain the desired 25,000 iterations we test the following parameter pairs (given in the form $(N^{iterations},N^{LNS})$): (1, 25,000), (5, 5000), (10, 2500), (25, 1000), (50, 500), and (100, 250). Table 1 provides the according average results aggregated across per-instance average values over five random runs as well as average CPU times in minutes. LNS+SC indicates that LNS is run without the additional local search step and LNS+LS+SC indicates that it is run as described in Section 2.1. Row-wise best results are indicated in bold. The best results for LNS+SC without LS are obtained with setting (50, 500) with an average run time of 20.08 min. The best results across all experiments reported in Table 1 are obtained with LNS+LS+SC and setting (25, 1000) with an average run time of 16.25 min. The latter $(N^{iterations},N^{LNS})$ setting appears to provide a good trade-off between solution quality and run time. It is also used in all subsequent experiments.

Table 1
Identifying the best parameter setting for hybridization step one.

$N^{iterations}$ / N^{LNS}		1 / 25,000	5 / 5000	10 / 2500	25 / 1000	50 / 500	100 / 250
(LNS + SC)	Avg.	525.43	523.41	520.77	520.54	**520.48**	520.88
	CPU[a]	**12.33**	13.99	14.42	16.79	20.08	30.75
(LNS + LS + SC)	Avg.	526.76	521.15	520.96	**519.79**	520.91	520.02
	CPU[a]	13.98	**13.28**	14.23	16.25	20.40	28.58

[a] Average run times in minutes on a Xeon computer with 2.67 GHz.

In Table 2, LNS+SC is compared to pure LNS. For both methods, results for a version with (denoted as LNS+LS+SC and LNS+LS) and without the additional local search step in the LNS (denoted as LNS+SC and LNS) are reported. The values given in the table correspond to average values across data sets with similar characteristics (A, B, C, and D). For each setting, average solutions over five random runs per instance, the best out of these five random runs and the average CPU time in minutes are given. We performed paired t-tests to determine if the use of local search and hybridization step one is effective. All tests were performed at a level of confidence of 99%. Hybridization step one leads to better results, where the improvement is statistically significant. Using the same number of LNS iterations (25,000), LNS+SC improves LNS by about 1.3% and LNS+LS+SC improves LNS+LS by about 1.7%. This improvement comes at the price of less than 3 min of additional computation time, on average (see Table 2).

In order to further validate hybridization step one, we ran LNS and LNS+LS also for 37,000 iterations, leading to average CPU times of 16.77 and 17.24 min, respectively. We then compare the obtained average results to those of LNS+SC and LNS+LS+SC as given in Table 2, i.e. using 25,000 iterations, with average CPU times of 16.79 and 16.25 min, respectively. Within comparable CPU times, LNS+SC improves the results of LNS by 0.93% on average and LNS+LS+SC improves the results of LNS+LS by 1.06% on average.

Hence we conclude that the proposed hybridization step is effective and we will continue to do so in the following experiments.

The use of the local search step (when comparing LNS vs. LNS+LS and LNS+SC vs. LNS+LS+SC) does not lead to results that are statistically different. However, further test runs have shown the inclusion of LS does indeed lead to improved results, without increasing computation times. The latter is due to the fact that within LNS, local search is only applied to solutions that are already close to the best solutions found so far (see Section 2.1). Therefore, we decided to use LNS+LS within the proposed hybrid framework and in all subsequent experiments.

3.3. Evaluating hybridization step two

In a second step, we evaluate the incorporation of the proposed VNS based heuristic column generator. In line with the above experiments, a total of 25,000 iterations shall be attained by the LNS in each of the experiments. Setting $N^{LNS}=1000$ and fixing $N^{new}=2$, we vary parameter pair ($N^{interval}$, $N^{iterations}$) such that $N^{iterations}/N^{interval}=25$ which leads to the desired 25,000 total LNS iterations. The tested parameter pairs are (1,25), (2,50), (5,125), (10,250), and (25,625). In Table 3 we compare again average and best solution values, as well as the average CPU times required, over five random runs, aggregated across all instances. The best setting in terms of both best and average values is (2,50) (indicated in bold): RLSCP together with the VNS based column generator is called twice before the (new) best complete solution is again improved by means of LNS.

In a next step, we evaluate different parameter settings within the proposed VNS, in comparison to the one described in Section 2.3. Instead of adding the first $N^{new}=2$ (addFirst) negative reduced cost columns derived from each column of the current basis, we test adding the best N^{new} (addBest), and adding all that we find (addAll). Note that in the latter case $N^{new}=\infty$. These three settings are evaluated in combination with different selection rules used to identify those requests in the VNS that are inserted, removed or exchanged. We test our proposed selection rule π against three other benchmark policies: random selection (random); biased selection based on the resulting negative reduced costs (NRC) (i.e., as opposed to strategy π also taking into account the actual routing costs); and a combination of all three (all), where one of the three rules is chosen randomly in every iteration of VNS. The results displayed in Table 4 confirm that π dominates all other selection rules. It obtains the best results in combination with addFirst. There is no obvious explanation why addFirst works slightly better than the other two strategies in this case (for the other selection rules it does not always obtain the best results). Our results confirm, however, that using the fastest strategy (addFirst requires less computation time than addBest and addAll), does not lead to results of reduced quality.

As a final step, we evaluate the robustness of the proposed algorithm with respect to the choice of N^{new}, which is varied between 1 and 10. The algorithm is applied using the best parameter setting identified previously (i.e. $N^{interval}=2$, $N^{iterations}=50$, addFirst). The results are shown in Table 5 and confirm that setting $N^{new}=2$ leads to the best results, both in terms of the average and best solutions obtained.

Summarizing, the merit of hybridization step two is an improvement in both average and best solution quality of about

Table 3
Identifying the best parameter setting for hybridization step two.

$N^{interval}$	1	2	5	10	25
$N^{iterations}$	25	50	125	250	625
Best	513.96	**512.44**	515.50	513.90	515.29
Avg.	519.73	**517.83**	519.36	519.23	520.81
CPU[a]	23.61	21.84	32.61	39.18	58.34

[a] Average run times in minutes on a Xeon computer with 2.67 GHz.

Table 4
Identifying the best VNS setting.

Strategy	Best			Avg.		
	addFirst	addBest	addAll	addFirst	addBest	addAll
Random	512.97	513.86	514.83	518.58	518.79	519.85
π	**512.44**	513.33	513.44	**517.83**	518.69	518.13
NRC	514.37	514.28	514.06	519.19	519.06	519.09
All	514.15	514.57	513.77	519.16	520.63	519.44

Table 2
Comparing LNS and hybridization step one with and without local search.

Data set	LNS			LNS+LS			LNS+SC			LNS+LS+SC		
	Avg.	Best	CPU[a]	Avg.	Best	CPU[a]	Avg.	Best	CPU[a]	Avg.	Best	CPU[a]
A	517.72	512.27	7.92	515.27	511.48	7.99	509.69	506.72	12.27	*508.54*	**504.06**	11.32
B	597.64	587.78	9.94	598.82	589.14	10.02	590.05	581.81	15.18	*588.34*	**581.44**	14.00
C	479.36	473.60	17.20	481.22	474.94	18.08	*473.53*	**469.06**	19.16	474.34	469.97	19.33
D	543.54	534.75	17.84	550.30	544.68	18.13	537.83	**527.94**	21.65	*536.32*	529.64	21.26
Avg	527.36	520.27	13.09	528.77	522.69	13.45	520.54	514.69	16.79	*519.79*	**514.42**	16.25

[a] Average run times in minutes on a Xeon computer with 2.67 GHz.

0.4%. This improvement comes at the price of about 5 min of additional computation time, on average.

3.4. Comparison to existing results

Based on the above, we identify the following parameter setting as the best one: $N^{iterations}=50$, $N^{new}=2$, $N^{interval}=2$ and $N^{LNS}=1000$. In Table 6, we compare the results of our hybrid framework, using the identified parameter setting, to the best known results and the VNS of [4] on a per-instance level. Columns headed by Best (Avg.) report the best (average) solution values. Average run times are reported in columns headed by CPU. In column four we report the best known solutions, as reported in [4]. The given deviations are deviations (columns (%)) from the best known results. Negative deviations indicate improvements. All results that tie with or improve the best known results of [4] are printed in bold. Results that improve the best known solutions available so far are highlighted by an asterisk. The proposed hybrid framework obtains solutions of very high quality. Comparing per-instance average results to the best known values, an average deviation of only 1.25% is observed. Comparing the best out of five random runs to the best known values, the deviation reduces to 0.31%, on average, and four new best solutions can be identified. Comparing the best solution values identified during all parameter tuning tests, we are able to improve nine out of 20 best known results and we tie for another nine. Furthermore, run on the same computer, the proposed framework requires only a fraction of the run time of the previous best method: on average about 22 min compared to 73 in case of the VNS. These results clearly indicate that the proposed framework is a highly competitive method for the DARP. (All new best results are available at http://prolog.univie.ac.at/research/DARP/.)

In a next step, we apply the proposed hybrid LNS to two sets of instances for which optimal solutions are known. These instances were proposed by Ropke et al. [7] and consist of 2–8 vehicles and 16–96 requests. The first set ("a" instances) represents the context where (smaller) vehicles ($Q=3$) are used for the transportation of individuals whereas the second set ("b" instances) includes (larger) vehicles ($Q=6$) for the transportation of individuals or groups of individuals. The generation of these instances and their main characteristics are described in detail in [6,7].

In Table 7 we compare per-instance average and best results (out of five random runs) obtained by our method to the optimal values as reported in [7]. We were able to find the optimal solution for 13 out of 21 instances per set. Those results are printed in bold. Comparing per-instance average results to the optimal solutions, an average deviation of only 0.16% (0.12%) is observed for instances in set a (b). When comparing the best out of five random runs to the optimal solutions, the average deviation reduces to 0.05% (0.06%). The average run time required is 2.26 min.

In a final step, we also compare the proposed hybrid LNS to results reported by Jain and Van Hentenryck [5] for the instances of Cordeau and Laporte [3] using a method they name large neighborhood search with first feasible probabilistic acceptance (LNS–FFPA). It combines constraint programming with LNS. The authors report results for a run time limit of 5 min, whereas they argue that since the modeling framework COMET is used, their run times have to be divided by a factor of 3 in order to obtain comparable run times to a program based on C++ code. Thus, their algorithm is in fact run for 15 min on a Core 2 Quad Q6600 CPU. In order to obtain comparable results, we run the proposed hybrid LNS with a run time limit of 5 min: we iterate between SC and the column generation component based on VNS, with the best settings identified above, and LNS, run for 1000 iterations at each call, until the run time limit is reached. We compare the proposed hybrid LNS also to pure LNS (i.e. without any of the hybridization

Table 5
Identifying the best parameter setting for hybridization step two.

N^{new}	1	2	3	4	5	7	10
Best	514.28	**512.44**	514.78	512.98	513.92	514.82	513.67
Avg.	519.63	**517.83**	519.00	519.15	519.17	520.74	519.56
CPU[a]	21.74	21.84	22.79	23.92	25.86	26.96	25.09

[a] Average run times in minutes on a Xeon computer with 2.67 GHz.

Table 6
Proposed hybrid LNS vs. best known results.

Instance	m	n	VNS [4]				Hybrid LNS					All tests	
			BKS[a]	Best	Avg.	CPU[b]	Best	(%)	Avg.	(%)	CPU[c]	Best	(%)
R1a	3	24	**190.02**	**190.02**	**190.02**	5.18	**190.02**	0.00	**190.02**	0.00	0.54	**190.02**	0.00
R2a	5	48	**301.34**	**301.34**	301.78	12.06	**301.34**	0.00	302.53	0.40	2.76	**301.34**	0.00
R3a	7	72	**532.00**	533.01	536.08	19.31	535.28	0.62	538.21	1.17	5.11	**532.00**	0.00
R4a	9	96	**570.25**	573.92	578.21	24.13	571.09	0.15	576.26	1.05	16.29	**570.25**	0.00
R5a	11	120	**628.11**	636.77	637.72	87.69	629.52	0.22	637.59	1.51	26.70	**627.68***	−0.07
R6a	13	144	794.06	802.12	809.27	94.10	**788.88***	−0.65	800.35	0.79	48.48	**785.26***	−1.11
R7a	4	36	**291.71**	**291.71**	294.26	5.09	**291.71**	0.00	292.56	0.29	1.02	**291.71**	0.00
R8a	6	72	**487.84**	490.58	495.70	44.29	491.93	0.84	495.20	1.51	5.92	489.33	0.30
R9a	8	108	**658.31**	666.96	673.18	103.97	661.47	0.48	676.09	2.70	24.89	659.85	0.23
R10a	10	144	**857.11**	866.97	873.15	211.29	872.31	1.77	878.93	2.55	47.17	**855.15***	−0.23
R1b	3	24	**164.46**	**164.46**	**164.46**	6.24	**164.46**	0.00	166.06	0.97	0.61	**164.46**	0.00
R2b	5	48	**295.66**	296.65	299.19	15.71	295.96	0.10	298.88	1.09	3.00	**295.66**	0.00
R3b	7	72	486.57	490.16	494.23	28.83	**484.83***	−0.36	491.29	0.97	8.19	**484.83***	−0.36
R4b	9	96	**530.70**	533.15	540.50	72.66	534.84	0.78	541.19	1.98	22.58	**529.33***	−0.26
R5b	11	120	**578.61**	583.12	588.48	131.85	587.67	1.57	590.22	2.01	44.09	**577.98***	−0.11
R6b	13	144	740.35	742.28	751.55	227.25	**738.01***	−0.32	743.64	0.45	71.50	**737.69***	−0.36
R7b	4	36	**248.21**	**248.21**	**248.21**	8.77	**248.21**	0.00	**248.21**	0.00	1.30	**248.21**	0.00
R8b	6	72	**461.39**	462.50	468.97	39.10	463.67	0.49	470.25	1.92	9.54	**461.39**	0.00
R9b	8	108	597.75	600.18	607.94	75.70	**593.49***	−0.71	606.25	1.42	27.49	**593.49***	−0.71
R10b	10	144	**795.16**	796.90	805.93	250.20	804.22	1.14	812.81	2.22	69.57	**793.21***	−0.24
Avg			510.48	513.55	517.94	73.17	512.44	0.31	517.83	1.25	21.84	509.44	−0.15

[a] Best known solutions computed by Parragh et al. as reported in [4].
[b] Average run times (5 independent random runs per instance) in minutes on a Xeon computer with 2.67 GHz.
[c] Average run times in minutes on a Xeon computer with 2.67 GHz.

Table 7
Proposed hybrid LNS vs. optimal results.

m	n	Set a						Set b					
		Instance	BUC [7]		Hybrid LNS			Instance	B&C[7]		Hybrid LNS		
			Opt[a]	CPU[b]	Best	Avg.	CPU[c]		Opt[a]	CPU[b]	Best	Avg.	CPU[c]
2	16	a2-16	294.25	0.01	**294.25**	**294.25**	0.12	b2-16	309.41	0.04	**309.41**	**309.41**	0.15
2	20	a2-20	344.83	0.05	**344.83**	**344.83**	0.28	b2-20	332.64	0.01	**332.64**	**332.64**	0.21
2	24	a2-24	431.12	0.12	**431.12**	**431.12**	0.35	b2-24	444.71	0.06	**444.71**	444.83	0.40
3	24	a3-24	344.83	1.31	**344.83**	**344.83**	0.29	b3-24	394.51	0.55	**394.51**	**394.51**	0.31
3	30	a3-30	494.85	15.48	**494.85**	495.27	0.50	b3-30	531.44	3.20	531.45	531.45	0.48
3	36	a3-36	583.19	8.13	**583.19**	583.25	0.83	b3-36	603.79	37.57	**603.79**	604.13	0.74
4	32	a4-32	485.50	0.12	**485.50**	485.71	0.55	b4-32	494.82	75.42	**494.82**	**494.82**	0.43
4	40	a4-40	557.69	0.51	**557.69**	**557.69**	0.78	b4-40	656.63	0.15	**656.63**	**656.63**	1.00
4	48	a4-48	668.82	1.01	**668.82**	669.04	1.62	b4-48	673.81	0.63	**673.81**	674.93	1.73
5	40	a5-40	498.41	0.24	**498.41**	**498.41**	0.85	b5-40	613.72	0.59	**613.72**	613.73	0.78
5	50	a5-50	686.62	2.67	686.63	686.87	1.60	b5-50	761.40	0.92	**761.40**	762.12	1.49
5	60	a5-60	808.42	2.59	808.48	809.34	2.51	b5-60	902.04	1.96	902.52	903.46	3.00
6	48	a6-48	604.12	0.90	**604.12**	604.54	1.14	b6-48	714.83	0.27	**714.83**	**714.83**	1.07
6	60	a6-60	819.25	3.94	820.30	821.91	2.29	b6-60	860.07	0.86	**860.07**	860.15	2.07
6	72	a6-72	916.05	8.16	916.66	919.77	4.43	b6-72	978.47	127.71	979.61	980.27	4.43
7	56	a7-56	724.04	5.36	**724.04**	**724.04**	1.67	b7-56	823.97	56.65	**823.97**	824.17	1.82
7	70	a7-70	889.12	6.13	889.58	893.50	2.88	b7-70	912.62	4.66	913.11	914.97	3.70
7	84	a7-84	1033.37	29.43	1036.17	1040.39	7.04	b7-84	1203.37	11.51	1211.27	1213.55	6.72
8	64	a8-64	747.46	6.48	**747.46**	747.99	2.14	b8-64	839.89	8.11	840.63	840.63	2.56
8	80	a8-80	945.73	26.14	948.69	951.36	5.73	b8-80	1036.34	4.04	1036.65	1038.28	3.84
8	96	a8-96	1232.61	1210.56	1234.78	1236.27	9.92	b8-96	1185.55	847.41	1187.80	1190.74	10.32
Avg			671.92	63.30	672.40	673.35	2.26		727.33	56.30	727.97	728.58	2.25
Avg (%)					0.05	0.16					0.06	0.12	

[a] Optimal solutions computed by Ropke et al. as reported in [7].
[b] Run times in minutes on a AMD Opteron 250 computer with 2.4 GHz.
[c] Average run times in minutes on a Intel Xenon computer with 2.67 GHz.

Table 8
Proposed hybrid LNS vs. LNS-FFPA [5], VNS [4] and pure LNS (run time limit: 5 min).

Instance	m	n	LNS-FFPA [5][a]	VNS [4][b]		LNS		Hybrid LNS	
			Avg.	Avg.	(%)	Avg.	(%)	Avg.	(%)
R1a	3	24	190.77	**190.02**	−0.39	**190.02**	−0.39	**190.02**	−0.39
R2a	5	48	304.45	304.49	0.01	303.82	−0.21	**303.63**	−0.27
R3a	7	72	547.39	540.63	−1.24	539.11	−1.51	**536.77**	−1.94
R4a	9	96	595.05	582.42	−2.12	594.83	−0.04	**581.41**	−2.29
R5a	11	120	662.56	658.45	−0.62	653.95	−1.30	**643.72**	−2.84
R6a	13	144	**832.74**	844.77	1.44	834.74	0.24	837.01	0.51
R7a	4	36	292.86	295.75	0.99	293.30	0.15	**291.93**	−0.32
R8a	6	72	505.15	504.07	−0.21	502.42	−0.54	**498.44**	−1.33
R9a	8	108	711.60	701.24	−1.46	697.90	−1.93	**697.22**	−2.02
R10a	10	144	911.18	975.34	7.04	918.13	0.76	**908.12**	−0.34
Avg			555.38	559.72	0.34	552.82	−0.48	548.83	−1.12
R1b	3	24	**164.46**	164.54	0.05	165.52	0.65	**164.46**	0.00
R2b	5	48	301.67	299.53	−0.71	300.85	−0.27	**297.99**	−1.22
R3b	7	72	504.69	498.41	−1.24	501.92	−0.55	**495.02**	−1.92
R4b	9	96	566.48	562.05	−0.78	562.95	−0.62	**540.76**	−4.54
R5b	11	120	610.33	619.32	1.47	614.38	0.66	**607.17**	−0.52
R6b	13	144	785.13	820.17	4.46	789.75	0.59	**772.21**	−1.65
R7b	4	36	248.31	250.14	0.74	**248.21**	−0.04	**248.21**	−0.04
R8b	6	72	477.75	475.52	−0.47	476.18	−0.33	**473.19**	−0.95
R9b	8	108	633.51	**618.54**	−2.36	628.90	−0.73	620.39	−2.07
R10b	10	144	**857.95**	936.09	9.11	862.31	0.51	874.36	1.91
Avg			515.03	524.43	1.03	515.10	−0.01	509.38	−1.10
Total avg			535.20	542.07	0.69	533.96	−0.24	529.10	−1.11

[a] Results of [5] on a Core 2 Quad Q6600 and a run time limit of 15 min (the run times given in the paper [5] correspond to run times divided by a factor of 3).
[b] Computed with the code of [4] and a run time limit of 5 min on a Xeon Computer with 2.67 GHz.

components) and to the VNS of [4]. For the latter, since the acceptance scheme of the VNS of [4] is based on the number of iterations we first identify how many iterations are necessary to obtain the run time limit of 5 min for each instance. Then, we add 10^5 iterations to this limit and run the method until the desired 5 min are reached. For all four methods, average results per instance over five random runs are reported in Table 8. For VNS, LNS, and the proposed hybrid LNS, per instance percentage

deviations from the results of [5] for LNS–FFPA are given. On average, VNS obtains results that are 0.69% worse than LNS–FFPA. The main issue of the VNS of [4] is that, in case of the largest instances, within 5 min only very few iterations (fewer than 10^5) can be completed, leading to a rather poor performance for instances R10a and R10b. Pure LNS obtains results that improve the results of LNS–FFPA by 0.24% on average and the proposed hybrid LNS is able to improve the results of [5] by 1.11% on average, within the run time limit of 5 min. Only in two cases LNS–FFPA is able to converge to better solutions more quickly. Note that in our hybrid framework we use a parameter setting that is tailored to longer run times, especially for larger instances. Therefore, it can be assumed that even better results could be obtained for this more restrictive run time limit with a different parameter setting tailored to short computation times. Summarizing, these results clearly show that, in general, the proposed hybrid LNS converges faster to good solutions than the other three methods. Furthermore, it emphasizes that integrating ideas from column generation help to increase the convergence speed and thus lead to improved solutions within shorter computation times.

4. Discussion and conclusions

In this paper, we have implemented and tested a hybrid solution framework for the dial-a-ride problem. It combines ideas from large neighborhood search and column generation, integrating a variable neighborhood search based column generator, in an efficient way: we were able to find new best solutions for 9 out of 20 instances. On the same computer, it requires less than one third of the run time of the current state-of-the-art method.

The proposed hybridization scheme extends the one of Pirkwieser and Raidl [14] who iteratively solve a set covering problem on the routes generated by a variable neighborhood search algorithm which works on complete solutions. In our approach, a large neighborhood search algorithm works on complete solutions and additional routes are generated by a variable neighborhood search based heuristic column generator, relying on dual information. The obtained results indicate that using dual information in this way, within a column generation metaheuristic hybrid, leads to improved solution quality.

Also other authors have experimented with using dual information within a set covering–metaheuristic hybrid. Muter et al. [13], e.g., evaluate the integration of dual information to guide the selection of customers within the tabu search algorithm of their hybrid framework and to control the size of the column pool. Neither strategy leads to improving results. Experiments in the context of our hybrid large neighborhood search algorithm with a destroy and a repair operator that exploit dual information in the request selection decision (results are reported in [23]) support this finding: the additional operators did not lead to improved solutions.

The conclusions we draw from these observations are that dual information incorporated into a metaheuristic working on complete solutions rather than on individual routes does not have a positive impact on the performance of the entire method; but, dual information can be useful when exploited in the standard way, that is in the generation of additional individual routes or columns. In our case, these additional columns help to further diversify the column pool and thus to reach solutions of improved quality.

Future research will address the application of the proposed hybrid framework to other (real-world) routing problems.

Acknowledgments

We wish to thank the Austrian Research Promotion Agency (FFG, Grant #826151, program IV2Splus) and the Austrian Science Fund (FWF): T-514-N13 for sponsoring this work. Thanks are due to the two anonymous referees for their valuable comments.

References

[1] Cordeau J-F, Laporte G. The dial-a-ride problem: models and algorithms Annals of Operations Research 2007;153:29–46.
[2] Parragh SN, Doerner KF, Hartl RF. Demand responsive transportation. In: Wiley encyclopedia of operations research and the management sciences; 2010.
[3] Cordeau J-F, Laporte G. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B Methodological. 2003; 37:579–94.
[4] Parragh SN, Doerner KF, Hartl RF. Variable neighborhood search for the dial-a-ride problem. Computers & Operations Research 2010;37:1129–38.
[5] Jain S, Van Hentenryck P. Large neighborhood search for dial-a-ride problems. In: Principles and practice of constraint programming—CP 2011, Lecture Notes in computer science, vol. 6876. Springer; 2011. http://dx.doi.org/10.1007/978-3-642-23786-7_31.
[6] Cordeau J-F. A branch-and-cut algorithm for the dial-a-ride problem. Operations Research 2006;54:573–86.
[7] Ropke S, Cordeau J-F, Laporte G. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks 2007;49(4): 258–72.
[8] Ropke S. Heuristic and exact algorithms for vehicle routing problems. PhD thesis, Computer Science Department at the University of Copenhagen (DIKU); 2005.
[9] Savelsbergh MWP. The vehicle routing problem with time windows: minimizing route duration ORSA. Journal on Computing 1992;4:146–54.
[10] Raidl GR, Puchinger J, Blum C. Metaheuristic hybrids. In: Gendreau M, Potvin JY, editors. Handbook of metaheuristics. 2nd ed. Springer; 2010. p. 469–96.
[11] Blum C, Blesa Aguilera MJ, Roli A, Sampels M, editors. Hybrid metaheuristics. In: Studies in computational intelligence, vol. 114. Berlin: Springer; 2008.
[12] Prescott-Gagnon E, Desaulniers G, Rousseau L-M. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Networks 2009;54(4):190–204.
[13] Muter I, Birbil SI, Sahin G. Combination of metaheuristic and exact algorithms for solving set covering-type optimization problems. INFORMS Journal on Computing 2010;22(4):603–19.
[14] Pirkwieser S, Raidl GR. Boosting a variable neighborhood search for the periodic vehicle routing problem with time windows by ILP techniques. In: Voss S, Caserta M, editors, Proceedings of the eighth metaheuristic international conference (MIC 2009), Hamburg, Germany; 13–16 July 2009.
[15] Pisinger D, Ropke S. Large neighborhood search. In: Gendreau M, Potvin J-Y, editors. Handbook of metaheuristics. 2nd ed. Springer; 2010. p. 399–419.
[16] Ropke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 2006;40:455–72.
[17] Desaulniers G, Lessard F, Hadjar A. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. Transportation Science 2008;42:387–404.
[18] Mladenović N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24:1097–100.
[19] Hansen P, Mladenović N. Variable neighborhood search: principles and applications European Journal of Operational Research 2001;130(3):449–67.
[20] Shaw P. Using constraint programming and local search methods to solve vehicle routing problems. In: Proceedings CP-98 (fourth international conference on principles and practice of constraint programming); 1998.
[21] Ropke S, Pisinger D. A unified heuristic for a large class of vehicle routing problems with backhauls. European Journal of Operational Research 2006; 171:750–75.
[22] Desaulniers G, Desrosiers J, Solomon MM, editors. Column generation. Number 5 in GERAD 25th anniversary. Springer; 2005.
[23] Parragh SN, Schmid V. Hybrid large neighborhood search for the dial-a-ride problem. In: Proceeding of the VII ALIO/EURO—workshop on applied combinatorial optimization, Porto, Portugal, May 4–6; 2011.